LNCS 7183

Adrian-Horia Dediu
Carlos Martín-Vide (Eds.)

# Language and Automata Theory and Applications

6th International Conference, LATA 2012
A Coruña, Spain, March 2012
Proceedings

Springer

Adrian-Horia Dediu   Carlos Martín-Vide (Eds.)

# Language and Automata Theory and Applications

6th International Conference, LATA 2012
A Coruña, Spain, March 5-9, 2012
Proceedings

Springer

Volume Editors

Adrian-Horia Dediu
Carlos Martín-Vide
Universitat Rovira i Virgili, Research Group on Mathematical Linguistics
Avinguda Catalunya, 35, 43002 Tarragona, Spain
E-mail: {adrian.dediu; carlos.martin}@urv.cat

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-28331-4 e-ISBN 978-3-642-28332-1
DOI 10.1007/978-3-642-28332-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012931003

CR Subject Classification (1998): F.1, F.4, F.2-3, I.2, I.4, I.5

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

These proceedings contain the papers that were presented at the 6th International Conference on Language and Automata Theory and Applications (LATA 2012), held in A Coruña, Spain, during March 5–9, 2012.

The scope of LATA is rather broad, including: algebraic language theory; algorithms for semi-structured data mining; algorithms on automata and words; automata and logic; automata for system analysis and program verification; automata, concurrency and Petri nets; automatic structures; cellular automata; combinatorics on words; computability; computational complexity; computational linguistics; data and image compression; decidability questions on words and languages; descriptional complexity; DNA and other models of bio-inspired computing; document engineering; foundations of finite-state technology; foundations of XML; fuzzy and rough languages; grammars (Chomsky hierarchy, contextual, multidimensional, unification, categorial, etc.); grammars and automata architectures; grammatical inference and algorithmic learning; graphs and graph transformation; language varieties and semigroups; language-based cryptography; language-theoretic foundations of artificial intelligence and artificial life; parallel and regulated rewriting; parsing; pattern recognition; patterns and codes; power series; quantum, chemical and optical computing; semantics; string and combinatorial issues in computational biology and bioinformatics; string processing algorithms; symbolic dynamics; symbolic neural networks; term rewriting; transducers; trees, tree languages and tree automata; and weighted automata.

LATA 2012 received 114 submissions. Each one was reviewed by three Program Committee members, many of whom consulted with external referees. After a thorough and lively discussion phase, the committee decided to accept 41 papers (which represents an acceptance rate of 35.96%). The conference program also included three invited talks and two invited tutorials. Part of the success in the management of such a large number of submissions is due to the excellent facilities provided by the EasyChair conference management system.

We would like to thank all invited speakers and authors for their contributions, the Program Committee and the reviewers for their cooperation, and Springer for its very professional publishing work.

December 2011

Adrian-Horia Dediu
Carlos Martín-Vide

## Organization

LATA 2012 was organized by the Language in the Information Society Research Group—LYS—from the University of A Coruña, Spain, and the Research Group on Mathematical Linguistics—GRLMC—from the University Rovira i Virgili, Tarragona, Spain.

### Program Committee

| | |
|---|---|
| Eric Allender | Piscataway, USA |
| Miguel Á. Alonso | A Coruña, Spain |
| Amihood Amir | Ramat-Gan, Israel |
| Dana Angluin | New Haven, USA |
| Franz Baader | Dresden, Germany |
| Patricia Bouyer | Cachan, France |
| John Case | Newark, USA |
| Volker Diekert | Stuttgart, Germany |
| Paul Gastin | Cachan, France |
| Reiko Heckel | Leicester, UK |
| Sanjay Jain | Singapore |
| Janusz Kacprzyk | Warsaw, Poland |
| Victor Khomenko | Newcastle, UK |
| Bakhadyr Khoussainov | Auckland, New Zealand |
| Claude Kirchner | Paris, France |
| Maciej Koutny | Newcastle, UK |
| Gregory Kucherov | Marne-la-Vallée, France |
| Salvador Lucas | Valencia, Spain |
| Sebastian Maneth | Sydney, Australia |
| Carlos Martín-Vide (Chair) | Tarragona, Spain |
| Giancarlo Mauri | Milan, Italy |
| Aart Middeldorp | Innsbruck, Austria |
| Faron Moller | Swansea, UK |
| Angelo Montanari | Udine, Italy |
| Joachim Niehren | Lille, France |
| Mitsunori Ogihara | Miami, USA |
| Enno Ohlebusch | Ulm, Germany |
| Dominique Perrin | Marne-la-Vallée, France |
| Alberto Policriti | Udine, Italy |
| Alexander Rabinovich | Tel Aviv, Israel |
| Mathieu Raffinot | Paris, France |
| Jörg Rothe | Düsseldorf, Germany |
| Olivier H. Roux | Nantes, France |

| | |
|---|---|
| Yasubumi Sakakibara | Yokohama, Japan |
| Eljas Soisalon-Soininen | Espoo, Finland |
| Frank Stephan | Singapore |
| Jens Stoye | Bielefeld, Germany |
| Howard Straubing | Boston, USA |
| Masayuki Takeda | Fukuoka, Japan |
| Wolfgang Thomas | Aachen, Germany |
| Sophie Tison | Lille, France |
| Jacobo Torán | Ulm, Germany |
| Tayssir Touili | Paris, France |
| Esko Ukkonen | Helsinki, Finland |
| Frits Vaandrager | Nijmegen, The Netherlands |
| Manuel Vilares | Vigo, Spain |
| Todd Wareham | St. John's, Canada |
| Pierre Wolper | Liège, Belgium |
| Hans Zantema | Eindhoven, The Netherlands |
| Thomas Zeugmann | Sapporo, Japan |

## External Reviewers

Aizikowitz, Tamar
Alekseyev, Arseniy
Allauzen, Cyril
Atig, Mohamed Faouzi
Bapodra, Mayur
Bauer, Sebastian
Bengtson, Jesper
Bocchi, Laura
Bodlaender, Hans
Boker, Udi
Bollig, Benedikt
Bolus, Stefan
Borgwardt, Stefan
Brenguier, Romain
Bresolin, Davide
Butz, Martin V.
Calin, Georgel
Cassez, Franck
Cavaliere, Matteo
Chaturvedi, Namit
Choffrut, Christian
Chowdhury, Rezaul
D'Agostino, Giovanna
Darondeau, Philippe
Darriba, Víctor
de la Higuera, Colin
Delfieu, David
Della Monica, Dario
Donyina, Adwoa
Dovier, Agostino
Durand-Lose, Jérôme
Dörr, Daniel
Emmi, Michael
Enea, Constantin
Engelfriet, Joost
Escobar, Santiago
Faucou, Sébastien
Felgenhauer, Bertram
Felscher, Ingo
Fernández-González, Daniel
Filiot, Emmanuel
Fridman, Wladimir
Frougny, Christiane
Frutos Escrig, David de
Gagie, Travis
Gentilini, Raffaella
Gilleron, Rémi
Gog, Simon

Golubcovs, Stanislavs
Gurski, Frank
Gómez-Rodríguez, Carlos
Hachiya, Tsuyoshi
Heidarian, Faranak
Heinz, Jeffrey
Hertrampf, Ulrich
Holík, Lukáš
Hölldobler, Steffen
Inokuchi, Shuichi
Jacquemard, Florent
Janowski, Sebastian Jan
Jeż, Artur
Josuat-Vergès, Matthieu
Khan, Ajab
Kilpeläinen, Pekka
Klein, Shmuel
Koppel, Moshe
Krebs, Andreas
Kärkkäinen, Juha
Laroussinie, François
Legay, Axel
Lemay, Aurélien
Lime, Didier
Lippmann, Marcel
Lohrey, Markus
Lombardy, Sylvain
Maass, Alejandro
McKenzie, Pierre
Mäkinen, Erkki
Mokhov, Andrey
Monmege, Benjamin
Mousavi, Mohammad Reza
Murano, Aniello
Nowotka, Dirk
Palomino, Miguel
Pebesma, Edzer
Peñaloza, Rafael
Porello, Daniele
Puppis, Gabriele
Puzynina, Svetlana
Pérez, Jorge A.
Rampersad, Narad
Rezine, Ahmed
Ribadas Pena, Francisco José
Rogalewicz, Adam
Roichman, Yuval
Roos, Yves
Rouillard, José
Royer, James
Sakamoto, Hiroshi
Sala, Pietro
Salmela, Leena
Sangnier, Arnaud
Sato, Kengo
Schmitz, Sylvain
Schulz, Stefan
Segoufin, Luc
Serre, Olivier
Servières, Myriam
Sippu, Seppo
Soloviev, Sergei
Starikovskaya, Tatiana
Sternagel, Christian
Sundermann, Linda
Thiemann, René
Tomescu, Alexandru I.
Tomiura, Yoichi
Ulidowski, Irek
Ummels, Michael
Ventura, Enric
Verwer, Sicco
Vilares, Jesús
Waldmann, Johannes
Wei, Fang
Winkler, Sarah
Wittler, Roland
Zeitoun, Marc
Zielonka, Wiesław

## Organizing Committee

Miguel Á. Alonso, A Coruña (Co-chair)
Adrian-Horia Dediu, Tarragona
Carlos Gómez Rodríguez, A Coruña
Jorge Graña, A Coruña
Carlos Martín-Vide, Tarragona (Co-chair)
Bianca Truthe, Magdeburg
Jesús Vilares, A Coruña
Florentina-Lilica Voicu, Tarragona

# Table of Contents

# Measuring Information in Timed Languages[*]

Eugene Asarin

LIAFA – Université Paris Diderot and CNRS, France
eugene.asarin@liafa.jussieu.fr

Timed automata and timed languages [1] constitute a beautiful discovery that opened new perspectives to automata and language theory, as well as new applications to computer-aided verification. However the theory of timed regular languages is far from being achieved. Seven years ago, in [2], I argued that developing such a theory constituted an important research challenge, and I sketched a research program in this direction. Unfortunately, when listing research tasks on timed languages I have overlooked one interesting topic: measuring size of and information content in such languages. Catching up this omission became the focus of my research and the theme of this talk.

In this talk, I survey results obtained in [3–7] by my co-workers and myself on volume and entropy of timed regular languages.

To define size measures of timed languages we proceed as follows. A "slice" $L_n$ of a timed language $L$, limited to words with $n$ events, can be seen as subset of $\mathbb{R}^n \times \Sigma^n$, and its volume $V_n$ can be defined in a natural way. We call the exponential growth rate of $V_n$ volumic entropy $H$ of the timed language $L$.

For a timed regular $L$, volumes $V_n$ are rational, and relatively easy to compute for a fixed $n$. What is more interesting, computation of entropy $H$ is based on functional analysis. Namely, to a timed automaton can be associated a positive linear operator $\Psi$ (given by a matrix of integrals) on some Banach space. Then we prove that $H = \log \rho(\Psi)$, where $\rho$ denotes the spectral radius. We discuss exact and approximate methods of computing the entropy.

Entropy $H$ can be seen as a size of the language $L$ or as information contents in its typical timed words. We formalize this last observation in terms of Kolmogorov complexity and in terms of symbolic dynamics.

Entropy brings new insights into classical questions on timed automata. We show that having a not-too-small entropy suffices to rule out Zeno pathologies in timed automata. Thus "non-vanishing" timed regular languages satisfy a weak pumping lemma and behave well under discretization.

For "degenerate" timed automata, it is often the case that the slice $L_n$ is a union of polyhedra of different dimensions $\leq n$. We argue for a reasonable way of measuring volumes and entropies in this degenerate case.

We conclude this introduction to quantitative theory of timed languages by some open problems and speculate on future theoretical and practical applications.

[*] The support of Agence Nationale de la Recherche under the project EQINOCS (ANR-11-BS02-004) is gratefully acknowledged.

## References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
2. Asarin, E.: Challenges in timed languages: from applied theory to basic theory (column: Concurrency). Bulletin of the EATCS 83, 106–120 (2004)
3. Asarin, E., Degorre, A.: Volume and Entropy of Regular Timed Languages: Analytic Approach. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 13–27. Springer, Heidelberg (2009)
4. Asarin, E., Degorre, A.: Volume and Entropy of Regular Timed Languages: Discretization Approach. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 69–83. Springer, Heidelberg (2009)
5. Asarin, E., Degorre, A.: Two size measures for timed languages. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS. LIPIcs, vol. 8, pp. 376–387. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010)
6. Basset, N.: Dynamique Symbolique et Langages Temporisés. Master's thesis, Master Parisien de la Recherche Informatique (2010)
7. Basset, N., Asarin, E.: Thin and Thick Timed Regular Languages. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 113–128. Springer, Heidelberg (2011)

# Automata-Based Symbolic Representations of Polyhedra*

Bernard Boigelot, Julien Brusten, and Jean-François Degbomont

Institut Montefiore, B28,
Université de Liège,
B-4000 Liège, Belgium
{boigelot,brusten,degbomont}@montefiore.ulg.ac.be

**Abstract.** This work describes a data structure, the *Implicit Real-Vector Automaton (IRVA)*, suited for representing symbolically polyhedra, i.e., regions of $n$-dimensional space defined by finite Boolean combinations of linear inequalities. IRVA can represent exactly arbitrary convex and non-convex polyhedra, including features such as open and closed boundaries, unconnected parts, and non-manifold components. In addition, they provide efficient procedures for deciding whether a point belongs to a given polyhedron, and determining the polyhedron component (vertex, edge, facet, ...) that contains a point. An advantage of IRVA is that they can easily be minimized into a canonical form, which leads to a simple and efficient test for equality between represented polyhedra. We also develop an algorithm for computing Boolean combinations of polyhedra represented by IRVA.

## 1 Introduction

The problem of designing a good data structure for representing and handling *polyhedra*, i.e., regions of $n$-dimensional space delimited by finitely many planar boundaries, has important applications in several areas of computer science. The precise class of polyhedra that needs to be covered and the range of necessary manipulation operations actually differ according to the application field.

Our historical motivation for studying this problem is related to *computer-aided verification*, where polyhedra are used for representing sets of system configurations that are manipulated during symbolic state-space exploration of hybrid automata [11,6]. In this setting, polyhedra are defined as finite Boolean combinations of strict and/or non-strict linear inequalities. The operations that need to be performed on polyhedra include unions, intersections, projections, and linear transformations (for applying the transition relation of the system under study), and tests of inclusion or equality (for detecting that a fixed point has been reached).

* This work is partially supported by the *Interuniversity Attraction Poles* program *MoVES* of the Belgian Federal Science Policy Office, and by the grants 2.4607.08 and 2.4545.11 of the Belgian Fund for Scientific Research (F.R.S.-FNRS).

The efficient manipulation of polyhedra is also essential to *computed-aided design*, where they yield convenient approximations of the shape of arbitrary objects. In this framework, the spatial dimension $n$ is usually limited to 2 or 3, and polyhedra are often *regularized*, meaning that they are made equal to the topological closure of their interior. Intuitively, the regularization operation gets rid of polyhedron features that are considered to be negligible and problematic, such as isolated points, dangling facets or edges, and open boundaries. The operations applied on polyhedra include Boolean combinations in order to construct complex objects from elementary building blocks, geometric transformations and measurements, two and three-dimensional visualization, checking whether a point belongs to a given polyhedron (the *point location* problem), and computing the polyhedron component (vertex, edge, facet, $\ldots$) that contains a point (the *point classification* problem).

Finally, as last examples of applications, polyhedra are also used in *optimization theory* and *constraint programming* [16] for specifying systems of constraints. In those applications, the spatial dimension $n$ corresponds to the number of variables implicated in the constraints, which is usually large, and the considered polyhedra are often convex, meaning that they can be expressed as finite conjunctions of linear inequalities. Typical problems there consist in searching inside a polyhedron for a point that maximizes a given objective function, and deciding whether a polyhedron is empty or not.

In this work, we consider the polyhedra defined as a finite Boolean combination of open and closed linear constraints, which are also known as *Nef polyhedra* [1,10]. This class covers polyhedra with combinations of open and closed boundaries, non-convex or unconnected parts, and non-manifold components. Our aim is to obtain a data structure that is able to represent exactly those polyhedra, and for which efficient algorithms can be derived for computing their Boolean combinations, checking inclusion, equality, and emptiness, and solving the point location and point classification problems.

Several approaches have been proposed for tackling those problems. A first possibility is to represent a polyhedron by a logical formula expressed in the quantifier-free fragment of linear arithmetic, for which powerful solvers are available [9]. This solution has the advantage of being able to deal with large spatial dimensions, but does not provide efficient algorithms for checking set equality or inclusion, or for simplifying the representation of a polyhedron obtained as the result of complex operations. In the restricted case of *convex polyhedra*, formula-based representations can be augmented with redundant structural information (the so-called *vertices*, *extremal rays* and *lines* of polyhedra), which substantially simplifies comparison operations [13]. In computer-aided design applications, the main approaches consist in representing a solid object as an explicit Boolean combination of elementary primitives (*Constructive Solid Geometry (CSG)*) [15], or by a geometrical description of their boundary (*Boundary representations (B-rep)*). CSG methods can be generalized to non-polyhedral primitives such as spheres, toruses and shapes bounded by polynomial surfaces. They provide direct implementations of Boolean operators and an easy solution to the point location problem. However, they are usually restricted to regularized shapes, and do not make it possible to check easily inclusion or equality of objects. On the other hand, B-rep techniques are able to represent accurately features such as open and/or closed boundaries and non-manifold components, but do not admit efficient algorithms for applying Boolean operators or solving the point location problem. These drawbacks are addressed by *Selective Nef Complexes (SNC)*, which combine a geometrical description of the vertices, edges and facets that compose a polyhedron with a topological representation of the incidence relation between them. SNC data structures have the same expressive power as B-rep ones, but can be combined by means of Boolean operators. Algorithms have also been developed for solving the point location and point classification problems over these structures [10] in the case of small spatial dimensions ($n = 2$ or $n = 3$).

A different approach is to represent polyhedra using *Real-Vector Automata (RVA)*, which are a particular form of infinite-word automata recognizing encodings of points in $\mathbb{R}^n$ [2,5]. It has been established that RVA are expressive enough for representing arbitrary polyhedra. The advantages of automata-based representations are that computing Boolean combinations of polyhedra reduce to carrying out similar operations on the languages accepted by the automata, for which simple algorithms are known. Furthermore, RVA can easily be minimized into a canonical form [14]. This leads to efficient comparison operations between represented sets, and allows to simplify the results of long chains of operations. RVA also provide a very efficient algorithm for solving the point location and classification problems. The main drawback of RVA is their size that can grow linearly with the coefficients of linear constraints, which makes those symbolic representations unmanageable in some applications. This drawback is alleviated by *Implicit Real-Vector Automata (IRVA)*, which intuitively operate on similar principles as RVA, but replace some of their unnecessarily large internal structures by more concise algebraic objects [3]. Interestingly enough, it has been shown that the RVA structures replaced in IRVA closely match the internal components of SNC representations of polyhedra, and that their reachability properties represent the incidence relation between them. The advantages of IRVA over SNC representations are threefold. First, they inherit the canonicity properties of RVA, which reduces equality testing between polyhedra to a simple isomorphism check. Second, like RVA, they admit very efficient algorithms for the point location and classification problems, which proceed by following a single path in a decision structure. Finally, IRVA are applicable to any spatial dimension $n$.

## 2 Basic Notions and Notations

Let $n \in \mathbb{N}$ be a dimension. A *linear constraint* over points $\boldsymbol{x} \in \mathbb{R}^n$ is a constraint of the form $\boldsymbol{a}.\boldsymbol{x}\#b$, with $a \in \mathbb{Z}^n$, $b \in \mathbb{Z}$, and $\# \in \{<, \leq, =, \geq, >\}$. A finite Boolean combination of such constraints defines a *polyhedron*. A polyhedron $\Pi$ is *convex* if for every $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \Pi$ and $\lambda \in [0, 1]$, one has $\lambda\boldsymbol{x}_1 + (1 - \lambda)\boldsymbol{x}_2 \in \Pi$, i.e., the line segment joining $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is a subset of $\Pi$. Every convex polyhedron can be expressed as a finite conjunction of linear constraints. A polyhedron defined by a finite conjunction of linear equalities, i.e., constraints of the form $\boldsymbol{a}.\boldsymbol{x} = b$, is an *affine space*. An affine space that contains $\mathbf{0}$ is a *vector space*. The *dimension* $\dim(S) \leq n$ of an affine or vector space $S \subseteq \mathbb{R}^n$ is the largest number of linearly independent vectors it contains. A set $S \subseteq \mathbb{R}^n$ is *conical* with respect to the *apex* $\boldsymbol{v} \in \mathbb{R}^n$ if for all $\boldsymbol{x} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}_{>0}$, one has $\boldsymbol{x} \in S$ iff $\boldsymbol{v} + \lambda(\boldsymbol{x} - \boldsymbol{v}) \in S$, which intuitively states that the set $S$ is not affected by a scaling transformation centered on the point $\boldsymbol{v}$. A polyhedron that is conical is a *pyramid*. The set of apexes of a pyramid always forms an affine space [1].

## 3 Polyhedra

### 3.1 Topological Components

The main idea behind the data structure discussed in this work is to exploit the specific topological properties of polyhedra. It has been observed that the structure of a polyhedron $\Pi \subseteq \mathbb{R}^n$ is pyramidal in arbitrarily small neighborhoods of any point $\boldsymbol{v} \in \mathbb{R}^n$ [1,4]. This property can be formalized as follows.

**Definition 1.** *Let* $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n$ *and* $\varepsilon \in \mathbb{R}_{>0}$. *The* cubic closed neighborhood *of size* $\varepsilon$ *of* $\boldsymbol{v}$ *is the set*

$$N_\varepsilon(\boldsymbol{v}) = [v_1 - \frac{\varepsilon}{2}, v_1 + \frac{\varepsilon}{2}] \times [v_2 - \frac{\varepsilon}{2}, v_2 + \frac{\varepsilon}{2}] \times \quad \cdots \quad \times [v_n - \frac{\varepsilon}{2}, v_n + \frac{\varepsilon}{2}].$$

**Theorem 2.** *Let* $\Pi \subseteq \mathbb{R}^n$ *be a polyhedron. For every point* $\boldsymbol{v} \in \mathbb{R}^n$*, there exists* $\varepsilon \in \mathbb{R}_{>0}$ *such that* $\Pi$ *coincides over* $N_\varepsilon(\boldsymbol{v})$ *with a pyramid of apex* $\boldsymbol{v}$*.*

Note that if $\Pi$ coincides with a pyramid $P$ in the neighborhood $N_\varepsilon(\boldsymbol{v})$ of a point $\boldsymbol{v}$, then the same pyramid $P$ also describes its structure in all neighborhoods $N_{\varepsilon'}(\boldsymbol{v})$ such that $0 < \varepsilon' \leq \varepsilon$, since a pyramid is invariant by scaling transformations. It has additionally been established that a finite number of distinct pyramids suffices for describing the structure of $\Pi$ in the neighborhood of all points in $\mathbb{R}^n$ [1,4]. We have the following definition and theorem.

**Definition 3.** *Let* $\Pi \subseteq \mathbb{R}^n$ *be a polyhedron, and* $\boldsymbol{v} \in \mathbb{R}^n$ *be an arbitrary point. The* local pyramid *of* $\Pi$ *with respect to* $\boldsymbol{v}$ *is the pyramid* $P_\Pi(\boldsymbol{v})$ *that coincides with* $\Pi$ *over sufficiently small neighborhoods* $N_\varepsilon(\boldsymbol{v})$ *of* $\boldsymbol{v}$*.*

**Theorem 4.** *For each polyhedron* $\Pi \subseteq \mathbb{R}^n$*, the set* $\{P_\Pi(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathbb{R}^n\}$ *is finite.*

This theorem states that a polyhedron $\Pi$ partitions the space $\mathbb{R}^n$ into a finite number of equivalence classes, each described by a local pyramid. We call the equivalence class that contains a point $\boldsymbol{v}$ the *polyhedral component*, or more simply *component*, of $\Pi$ associated to $\boldsymbol{v}$. Such a component is thus uniquely characterized by the local pyramid $P_\Pi(\boldsymbol{v})$. The set of apexes of this pyramid forms the *characteristic affine space* of the component, denoted $\text{aff}(C)$ for a component $C$, and the dimension of this space defines the *dimension* of the component, denoted $\dim(C)$. Intuitively, the dimension of a component characterizes the number of degrees of freedom among its points. Components of dimension 0, 1, 2 thus correspond to the classical notions of *vertices*, *edges* and *facets* of polyhedra. An illustration is given in Figure 1.

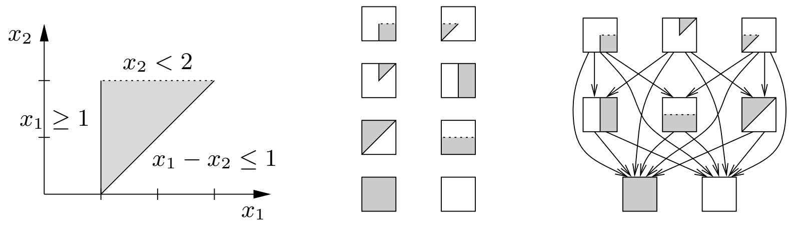


**Fig. 1.** Example of (a) polyhedron, (b) components, and (c) incidence relation

## 3.2 Incidence Relation

The components of a polyhedron are connected by an *incidence relation*. We have the following definition and theorem.

**Definition 5.** *A component $C_2$ of a polyhedron $\Pi$ is* incident *to a component $C_1$, which is denoted $C_1 \preceq C_2$, if for every point $\boldsymbol{v} \in \mathbb{R}^n$ that belongs to $C_1$, there exist points that are arbitrarily close to $\boldsymbol{v}$ and that belong to $C_2$.*

**Theorem 6.** *For every polyhedron, the incidence relation $\preceq$ is a partial order over its components. This relation is such that $C_1 \preceq C_2$ implies $\mathrm{aff}(C_1) \subseteq \mathrm{aff}(C_2)$ (and thus $\dim(C_1) \leq \dim(C_2)$).*

## 3.3 Extension to Polyhedral Partitions

The notions of polyhedral component and incidence relation can be extended to more general structures than polyhedra. We define a *polyhedral partition* as a partition $\Pi = \{\Pi_1, \Pi_2, \ldots, \Pi_m\}$ of $\mathbb{R}^n$, with $m > 0$, such that each $\Pi_i$ is a polyhedron. Such a partition can alternatively be specified as a *color function* $\Pi : \mathbb{R}^n \to \{1, 2, \ldots, m\}$ that maps every point $\boldsymbol{v} \in \mathbb{R}^n$ onto its index $\Pi(\boldsymbol{v})$ in the partition, which can be seen as a color assigned by the polyhedral partition out of $m$ distinct possibilities. Polyhedral partitions are especially useful in the framework of the point classification problem, where the core issue is to describe such a partition by a data structure from which one can easily compute the index, or color, of arbitrary points. A polyhedron can be seen as a particular instance of a polyhedral partition, limited to two colors corresponding to the points that respectively belong and do not belong to the polyhedron. In the rest of this paper, we will thus indifferently use polyhedra and polyhedral partitions.

The definition of pyramids readily adapts to polyhedral partitions: A polyhedral partition $\Pi$ over $\mathbb{R}^n$ is pyramidal with respect to the apex $\boldsymbol{v} \in \mathbb{R}^n$ if it is such that for every $\boldsymbol{x} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}_{>0}$, we have $\Pi(\boldsymbol{x}) = \Pi(\boldsymbol{v} + \lambda(\boldsymbol{x} - \boldsymbol{v}))$. A pyramidal partition of apex $\boldsymbol{v}$ is thus one that classifies the points of $\mathbb{R}^n$ on the sole basis of the direction in which they are seen from $\boldsymbol{v}$, which means that such a partition is not affected by scaling transformations centered on $\boldsymbol{v}$. From this extension of the concept of pyramid, one straightforwardly generalizes to polyhedral partitions the notions of local pyramid, components, and incidence relation.

## 4 Towards a Better Data Structure for Polyhedra

### 4.1 Real-Vector Automata

Selective Nef Complexes (SNC) are data structures that combine descriptions of the components of a polyhedron and of the incidence relation between them [10]. In this section, we study another class of representations, the Real-Vector Automata (RVA), and show that, even though RVA are based on different ideas from SNC representations, both data structures share some common principles of operation. Our aim will then be to define symbolic representations that are as concise as SNC, but inherit from RVA their canonicity property, as well as very efficient algorithms for solving the point location and point classification problems.

RVA are finite-state machines recognizing the coordinates of points, encoded into words [2,5]. They depend on the choice of a *numeration base* $r \in \mathbb{N}_{>1}$, which provides an alphabet of digits $\Sigma_r = \{0, 1, \ldots, r-1\}$, augmented with a distinguished symbol $\star$ used for separating the integer from the fractional part of encodings. In a given base $r$, a number $z \in \mathbb{R}_{\geq 0}$ is *encoded* by infinite words $a_{p-1}a_{p-2}\ldots a_0 \star a_{-1}a_{-2}\ldots$ such that $p > 0$, $a_i \in \Sigma_r$ for all $i < p$, and $z = \sum_{i<p} a_i r^i$. This scheme is extended to signed numbers by encoding negative numbers by their *r's complement*, which amounts to representing a number $z \in \mathbb{R}_{<0}$ by the encoding of $z + r^p$, where $p$ is the length of its integer part. The value of $p$ is not fixed, but has to satisfy the constraint $-r^{p-1} \leq z \leq r^{p-1}$. The integer part of an encoding can be increased at will, by repeating its leading digit (which is equal to 0 for positive and to $r-1$ for negative numbers), hence every number admits infinitely many encodings.

Points in $\mathbb{R}^n$ are encoded by combining encodings of their components, which can always be chosen such that their integer parts share the same length. By reading those component encodings synchronously, one symbol at a time, one obtains a point encoding that takes the form of an infinite word over the alphabet $\{0, 1, \ldots, r-1\}^n \cup \{\star\}$ (since the separator symbol is read at the same time in all component encodings, it can be denoted by a single symbol). The exponential size of the alphabet can be avoided by *serializing* the encodings, which amounts to replacing each symbol $(d_1, d_2, \ldots, d_n) \in \{0, 1, \ldots, r-1\}^n$ by the subword $d_1 d_2 \ldots d_n$ expressed over $\Sigma_r$.

Given a set $S \subseteq \mathbb{R}^n$ and a base $r \in \mathbb{N}_{>1}$, the base-$r$ encodings of the points in $S$ form a language. An infinite-word finite-state automaton that accepts this language is called a *Real-Vector Automaton (RVA)* representing $S$. It is known that RVA are expressive enough for representing all the sets that are definable in $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$ [2], i.e., the first-order additive theory of real and integer numbers, which covers our definition of polyhedra. Furthermore, a restricted form of infinite-word automata, *weak deterministic* ones, suffices for representing the sets definable in $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$, and the sets that are representable by these automata in every base exactly match those that are definable in that theory [5]. A weak automaton is a Büchi automaton such that each strongly connected component of its transition relation is entirely composed of accepting or non-accepting states. The advantages of using weak deterministic RVA is that they have efficient manipulation algorithms, in particular for applying to represented sets operations such as Boolean combinations, projections and Cartesian products, and that they admit an easily computable canonical form [14].

## 4.2 Point Decision in RVA

Consider a deterministic weak RVA $\mathcal{A}$ that represents a polyhedron $\Pi \subseteq \mathbb{R}^n$ in a base $r \in \mathbb{N}_{>1}$. Assessing whether a point $\boldsymbol{v} \in \mathbb{R}^n$ belongs or not to $\Pi$ reduces to encoding $\boldsymbol{v}$ in base $r$, which yields a word $w \in \Sigma_r^+ \star \Sigma_r^\omega$, and then checking whether this word is accepted by $\mathcal{A}$. Since $\mathcal{A}$ is deterministic, this can be done by following a single path in its transition graph. In a weak automaton, determining whether a path is accepting or not amounts to checking the accepting status of the last strongly connected component (SCC) that it visits. In other words, the point decision problem for $\boldsymbol{v}$ is solved by following transitions labeled by the successive symbols in $w$, which moves through the SCC of $\mathcal{A}$, until a final SCC is reached.

It has been shown that the SCC of $\mathcal{A}$ are related to the polyhedral components of $\Pi$ [4,3,7]. This can intuitively be explained as follows. Let $u \in \Sigma_r^+ \star \Sigma_r^*$ be a finite encoding prefix that contains $k$ digits in the fractional part of each vector component, and $s$ be the state of $\mathcal{A}$ reached after reading $u$. The points of $\mathbb{R}^n$ that admit an encoding prefixed by $u$ form a $n$-cube $C_u$ of size $r^{-k}$, and the shape of $\Pi$ inside this $n$-cube is uniquely determined by the state $s$. Assume that $s$ belongs to a non-trivial strongly connected component of $\mathcal{A}$, i.e., one containing a cycle from $s$ to itself, labeled by a word $u'$ that adds $k'$ additional digits to the encoding of each vector component. The encodings prefixed by $uu'$ determine a $n$-cube $C_{uu'}$ of size $r^{-(k+k')}$. In both cubes $C_u$ and $C_{uu'}$, the shape of $\Pi$ is identical. This shows that the linear scaling transformation that maps $C_u$ to $C_{uu'}$ leaves $\Pi$ invariant inside of this $n$-cube. It is established in [4] that this invariance property actually holds for arbitrary scaling factors, which implies that $\Pi$ is pyramidal inside $C_u$. A correspondence between the local pyramidal structures induced by the strongly connected components of $\mathcal{A}$ and the components of $\Pi$ has been discovered in [4], and is investigated in [7]. This correspondence is not exactly one-to-one: some polyhedral components can be split between several SCC, and components that have local pyramids that only differ by a translation are sometimes described by a common SCC.

The reachability relation between the SCC of $\mathcal{A}$ also loosely corresponds to the incidence relation between the components of $\Pi$. The intuition is as follows. Consider a non-trivial SCC $\mathcal{S}_2$ reachable from another one $\mathcal{S}_1$. For every point $\boldsymbol{v} \in \mathbb{R}^n$ with an encoding that ends up in $\mathcal{S}_1$, there exists a point $\boldsymbol{v}'$ that is arbitrarily close to $\boldsymbol{v}$ with an encoding ending up in $\mathcal{S}_2$. Indeed, one may follow in $\mathcal{S}_1$ the path reading the encoding of $\boldsymbol{v}$ for arbitrarily many transitions before deciding to divert towards one ending in $\mathcal{S}_2$.

The procedure for deciding whether a point $\boldsymbol{v} \in \mathbb{R}^n$ belongs to a polyhedron represented by a RVA can thus be summarized as follows. One follows in the transition graph of the automaton the path given by an encoding $w$ of $\boldsymbol{v}$, until reaching a non-trivial strongly connected component $\mathcal{S}_1$. If the remaining suffix of $w$ can be read without leaving $\mathcal{S}_1$, which means that $\boldsymbol{v}$ belongs to a polyhedral component $C_1$ represented by $\mathcal{S}_1$, then the accepting status of this component provides the answer to the point decision problem. If this suffix leaves $\mathcal{S}_1$, then it eventually reaches another non-trivial SCC $\mathcal{S}_2$, which intuitively corresponds to moving from $C_1$ to another polyhedral component $C_2$ that is incident to $C_1$. The same procedure is repeated in $\mathcal{S}_2$, and so on until the path finally reaches a SCC that it does not leave anymore.

## 4.3 Principles of IRVA

The idea behind *Implicit Real-Vector Automata* is to define a data structure representing the components of a polyhedron and the incidence relation between them, in such a way that the point decision problem can be solved by following deterministically a single path in the structure, similarly to RVA. Compared to RVA, the main advantage of IRVA will be their substantially more efficient representation of the polyhedral components.

Informally, an IRVA is a graph composed of *implicit states*, which correspond to the components of its represented polyhedron, and *explicit states* and *transitions*, that provide an acyclic and deterministic decision structure linking the implicit states. In order to decide whether a point $\boldsymbol{v} \in \mathbb{R}^n$ belongs to a polyhedron represented by an IRVA, one will start from a first implicit state $s_1$. If $\boldsymbol{v}$ belongs to the corresponding polyhedral component $C_1$, then the search is over and the answer to the decision problem depends on whether $C_1$ is a subset of the polyhedron (which may be indicated by a flag attached to $s_1$). If $\boldsymbol{v}$ does not belong to $C_1$, then the procedure follows the decision structure that leaves $s_1$, until reaching another implicit state $s_2$, representing a polyhedral component $C_2$ that is incident to $C_1$. The same procedure is repeated in this implicit state, and so on until reaching the component that finally contains $\boldsymbol{v}$. Note that this idea straightforwardly generalizes to representations of polyhedral partitions, by associating a color to each implicit state instead of a binary flag.

The information that needs to be associated to an implicit state $s$ includes the characteristic affine space $\mathrm{aff}(C)$ of its corresponding polyhedral component $C$, and either the color associated to this component or a binary acceptance flag.

The purpose of the deterministic decision structure leaving $s$ is to represent the structure of the underlying local pyramid of $C$, by directing the vectors leaving $C$ to the appropriate incident components. The development of such a decision structure will be addressed in Section 4.4.

A problem that remains to be addressed is to locate efficiently the first implicit state $s_1$ to be visited during the search for a point $\boldsymbol{v} \in \mathbb{R}^n$. This could be achieved by building a deterministic decision structure for classifying the points of $\mathbb{R}^n$. We choose to follow a simpler approach, which consists in considering only polyhedra in which the choice of the initial implicit state is trivial. We have the following result.

**Theorem 7.** *If a polyhedron $\Pi$ is a pyramid, then it contains a unique component $C$ that is minimum with respect to the incidence relation $\preceq$, i.e., such that $C \preceq C'$ for every component $C'$ of $\Pi$. This minimum component corresponds to the set of apexes of $\Pi$.*

As a consequence, if a pyramid $\Pi$ is represented by an IRVA, then the first implicit state visited during the search for a point can systematically be chosen to be the representation of the minimum component of $\Pi$, which eliminates the need for a special form of decision structure. Intuitively, this is possible because a pyramid of apex $\boldsymbol{v}$ is not affected by scaling transformations centered on $\boldsymbol{v}$. This property can be exploited for conducting the search in any arbitrarily small neighborhood of $\boldsymbol{v}$.

It is important to point out that moving from polyhedra to pyramids does not incur a loss of expressive power, for every polyhedron can be transformed into a pyramid that represents it without ambiguity, and vice-versa. We have the following definition.

**Definition 8.** *Let $\Pi \subseteq \mathbb{R}^n$ be a polyhedron. The* representing pyramid *of $\Pi$ is the polyhedron $\overline{\Pi} \subseteq \mathbb{R}^{n+1} = \{\lambda(x_1, \ldots, x_n, 1) \mid \lambda \in \mathbb{R}_{>0} \wedge (x_1, \ldots, x_n) \in \Pi\}$.*

For every polyhedron $\Pi \subseteq \mathbb{R}^n$, the polyhedron $\overline{\Pi}$ is a pyramid of apex $\mathbf{0}$. The polyhedron $\Pi$ can be recovered from $\overline{\Pi}$ by the transformation $\Pi = \{(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n, 1) \in \overline{\Pi}\}$, which amounts to computing the section of $\overline{\Pi}$ by the planar constraint $x_{n+1} = 1$, and projecting the result over the $n$ first vector components.

Note that the elementary operations over polyhedra, such as computing Boolean combinations, testing equality or inclusion, and solving the point decision and point classification problems, readily translate into identical or similar operations over their representing pyramids. The notion of representing pyramids also straightforwardly generalizes to polyhedral partitions.

In the sequel, we will thus only address without loss of generality the problem of designing a data structure for pyramids, or pyramidal partitions, that admit the apex $\mathbf{0}$. This choice brings the additional benefit of simplifying some structures. In particular, the characteristic affine spaces $\mathrm{aff}(C)$ of components become, in the case of such partitions, vector spaces since they systematically include $\mathbf{0}$. It follows that the implicit states of IRVA can actually be annotated by vector spaces instead of affine ones.

## 4.4 Decision Structure for Directions

Before defining formally IRVA, we discuss the details of the decision structures linking implicit states. This section is adapted from [3]. The problem that is tackled can be stated as follows. Let $\mathcal{A}$ be an IRVA representing a pyramidal partition $\Pi$ of $\mathbb{R}^n$ , with apex $\mathbf{0}$. Let $s$ be an implicit state of $\mathcal{A}$, representing a polyhedral component $C$ of $\Pi$. As explained in Section 4.3, a vector space $VS(s)$ is associated to $s$, and represents the set of apexes of the local pyramid $P_\Pi(C)$ of $C$. The problem consists in building a deterministic decision structure that classifies the points $\boldsymbol{v} \in \mathbb{R}^n$ such that $\boldsymbol{v} \not\in VS(s)$ according to their polyhedral component in the pyramid $P_\Pi(C)$.

For every $\lambda \in \mathbb{R}_{>0}$, the decision taken for the point $\lambda\boldsymbol{v}$ has to match exactly the one taken for $\boldsymbol{v}$, since $P_\Pi(C)$ is invariant by scaling transformations centered on $\mathbf{0}$. We ensure that this property is satisfied by first *normalizing* the point $\boldsymbol{v}$, which intuitively corresponds to keeping only the direction in which it can be reached from $VS(s)$, and then encode this direction over a finite alphabet. The decision structure leaving $s$ can then take the form of an acyclic graph, the edges of which are labeled by symbols of this alphabet.

We first describe the normalization operation. Let $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_m\}$, with $0 \leq m \leq n$, be a basis of the vector space $VS(s)$. If $m = n$, then the component $C$ is universal, meaning that $P_\Pi(C)$ has a uniform color over $\mathbb{R}^n$. In this case, there is no need for a decision structure leaving $s$, since one cannot have $\boldsymbol{v} \not\in VS(s)$. If $m < n$, then we introduce $n - m$ vectors $\boldsymbol{z}_1$, $\boldsymbol{z}_2$, $\ldots$, $\boldsymbol{z}_{n-m}$ such that $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_m, \boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_{n-m}\}$ is a basis of $\mathbb{R}^n$. The vectors $\boldsymbol{z}_1$, $\boldsymbol{z}_2$, $\ldots$, $\boldsymbol{z}_{n-m}$ can be chosen in a canonical way, by selecting among $(1, 0, \ldots, 0)$, $(0, 1, \ldots, 0)$, $\ldots$, $(0, 0, \ldots, 1)$, in that order, $n - m$ vectors linearly independent with $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_m\}$.

The next step for normalizing $\boldsymbol{v}$ is to express this point in the coordinate system $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n-m}\}$, obtaining a tuple $(y_1, \ldots, y_m, z_1, \ldots, z_{n-m})$. Clearly, $P_\Pi(C)$ is not affected by translations within $VS(s)$, hence only the coordinates $(z_1, z_2, \ldots, z_{n-m})$ are relevant for classifying $\boldsymbol{v}$.

Let $\boldsymbol{z} = (z_1, z_2, \ldots, z_{n-m})$. The next operation is to get rid of the magnitude of this vector, keeping only its direction. This can be done by computing the intersection of the half-line $\{\lambda\boldsymbol{z} \mid \lambda \in \mathbb{R}_{>0}\}$ with the faces of the *normalization cube* $[-\frac{1}{2}, \frac{1}{2}]^{n-m}$. The resulting point $\boldsymbol{v}' \in \mathbb{R}^{n-m}$ then provides the normalization of our original point $\boldsymbol{v}$.

It remains to define an encoding scheme for mapping normalized points onto words over a finite alphabet. Instead of using the technique discussed in Section 4.1, we use an encoding relation that exploits the fact that the points to be encoded belong to the normalization cube. Precisely, an encoding of a normalized point $\boldsymbol{v}' = (v'_1, v'_2, \ldots, v'_{n-m}) \in \mathbb{R}^{n-m}$ begins with a symbol $a \in \{-(1), +(1), -(2), +(2), \ldots, -(n-m), +(n-m)\}$ that identifies the face of the normalization cube to which $\boldsymbol{v}'$ belongs: If $a = -(i)$, with $1 \leq i \leq n - m$, then $v'_i = -\frac{1}{2}$; if $a = +(i)$, then $v'_i = +\frac{1}{2}$. The leading symbol $a$ is followed by a suffix $w \in \{0, 1\}^\omega$ that encodes the position of $\boldsymbol{v}'$ within the face of the normalization cube represented by $a$. The suffix $w$ is obtained as follows. If

$a \in \{-(i), +(i)\}$, with $1 \leq i \leq n - m$, then the $i$-th component $v'_i$ of $\boldsymbol{v}'$ is fixed, and it remains to encode the coordinates $\boldsymbol{v}'' = (v'_1, \ldots, v'_{i-1}, v'_{i+1}, \ldots, v'_{n-m})$. We then define $w \in \{0,1\}^\omega$ as the fractional part of a base-2 encoding of the point $\boldsymbol{v}'' + (\frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2})$, i.e., such that $0^{n-m-1} \star w$ is a binary encoding of this point.

This encoding scheme maps a normalized point $\boldsymbol{v}' \in \mathbb{R}^{n-m}$ onto words of the form $aw$, with $a \in \{-(1), +(1), -(2), +(2), \ldots, -(n-m), +(n-m)\}$ and $w \in \{0,1\}^\omega$. Note that some points have multiple encodings, because either they are located at the boundary between different faces of the normalization cube, or their position on a face of this cube admits more than one fractional binary encoding. This situation is not at all problematic, provided that the decision structure leaving the implicit state $s$ handles all these encodings in the same way.

In order to obtain a deterministic decision structure rooted at $s$, we consider the successive digits of encodings $w$. The leading digit $a$ of $w$ characterizes a specific face of the normalization cube. The normalized points $\boldsymbol{v}' \in \mathbb{R}^{n-m}$ that admit an encoding prefixed by $a$ form a convex pyramid, and it is easily shown that the points $\boldsymbol{v} \in \mathbb{R}^n$ that normalize into $\boldsymbol{v}'$ form a convex pyramid as well. This latter pyramid, which only depends on the vector space $V = VS(s)$ and the leading symbol $a$, will be denoted $R_{V,a}$. This pyramid corresponds to a region of $\mathbb{R}^n$ that has a non empty intersection with some subset $S$ of the components of $\Pi$. If this subset contains a unique minimum component $C'$ with respect to the incidence relation $\preceq$, then deciding whether a point $\boldsymbol{v} \in R_{V,a}$ belongs or not to $\Pi$ can be carried out in the neighborhood of $C'$, hence the decision branch labeled by $a$ can be directed to the implicit state representing $C'$.

If, on the other hand, the subset $S$ of components does not contain a minimum element with respect to $\preceq$, then the decision branch labeled by $a$ has to be developed further. By reading an additional prefix $u \in \{0,1\}^*$, one refines the region $R_{V,a}$ into the pyramid $R_{V,au}$ containing all points $\boldsymbol{v} \in \mathbb{R}^n$ whose normalization admits an encoding prefixed by $au$. Once again, if $R_{V,au}$ covers a subset of components of $\Pi$ with a unique minimum component $C'$, then the decision branch labeled by $au$ can be oriented to the implicit state representing $C'$. Otherwise, the refinement procedure has to be repeated until the prefix is long enough. Termination is ensured by the following result.

**Theorem 9.** *Let $\Pi$ be a pyramidal partition of $\mathbb{R}^n$, with apex $\mathbf{0}$, $C$ be one of its polyhedral components, and $V = \mathrm{aff}(C)$. There exists $k \in \mathbb{N}_{>0}$ such that for every encoding $w$ of length $k$, the subset of components of $\Pi$ that have a non-empty intersection with the region $R_{V,w}$ admits a unique minimum element with respect to the incidence relation $\preceq$.*

## 5 Implicit Real-Vector Automata

### 5.1 Syntax

We are now ready to define the syntax of IRVA. As discussed in Sections 3.3 and 4.3, the goal is to obtain a symbolic representation of a pyramidal partition, i.e., a generalized pyramid in which the components are labeled by a finite range of colors instead of a binary acceptance flag.

**Definition 10.** *An* Implicit Real-Vector Automaton (IRVA) *is a tuple* $(n, S_I, S_E, s_0, \delta, VS, col)$*, where:*

- $n \in \mathbb{N}$ *is a* dimension*,*
- $S_I$ *is a finite set of* implicit states*,*
- $S_E$ *is a finite set of* explicit states*,*
- $s_0 \in S_I$ *is an* initial state*,*
- $\delta : (S_I \times \pm(\mathbb{N}_{>0})) \cup (S_E \times \{0, 1\}) \to S_I \cup S_E$ *is a (partial)* transition function*,*
- $VS : S_I \to 2^{\mathbb{R}^n}$ *associates a* vector space *to each implicit state,*
- $col : S_I \to \mathbb{N}_{>0}$ *associates a* color *to each implicit state.*

In order to be valid, IRVA have to satisfy some syntactic constraints. First, the transition function $\delta$ must be acyclic as well as complete, in the sense that for every implicit state $s \in S_I$, $\delta(s, -(i))$ and $\delta(s, +(i))$ are defined iff $i \in \{1, 2, \ldots, n - \dim(VS(s))\}$, and for every explicit state $s \in S_E$, both $\delta(s, 0)$ and $\delta(s, 1)$ are defined. Let us denote by $s_1 \xrightarrow{w} s_2$, or more simply $s_1 \to s_2$ if $w$ is not of interest, the fact that the transition function $\delta$ leads from the implicit state $s_1 \in S_I$ to $s_2 \in S_I$, reading the word $w \in \pm(\mathbb{N}_{>0})\{0, 1\}^*$, and visiting only explicit states between $s_1$ and $s_2$. The reflexive and transitive closure of the relation $\to$ is denoted $\to^*$. We impose the following additional restrictions on IRVA: each implicit state $s \in S_I$ must be reachable, i.e., such that $s_0 \to^* s$, and for every pair $s_1, s_2 \in S_I$ such that $s_1 \xrightarrow{w} s_2$ for some word $w$, one must have $VS(s_1) \subset VS(s_2)$ (which implies $\dim(VS(s_1)) < \dim(VS(s_2))$), as well as $R_{VS(s_1),w} \cap VS(s_2) \neq \emptyset$. These restrictions intuitively express that the decision structures linking the implicit states are consistent with the properties of the incidence relation between polyhedral components.

### 5.2 Semantics

Let $\mathcal{A} = (n, S_I, S_E, s_0, \delta, VS, col)$ be an IRVA and $\boldsymbol{v} \in \mathbb{R}^n$ be a point. We have the following definition.

**Definition 11.** *A* run *of $\mathcal{A}$ over $\boldsymbol{v}$ is a finite sequence $s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \cdots \xrightarrow{w_m} s_m$, where $0 \leq m \leq n$, $s_0, s_1, \ldots, s_m \in S_I$, and $w_1, w_2, \ldots, w_m \in \pm(\mathbb{N}_{>0})\{0, 1\}^*$, such that $\boldsymbol{v} \in VS(s_m)$, and for every $i \in \{0, 1, \ldots, m - 1\}$, $\boldsymbol{v} \notin VS(s_i)$ and $\boldsymbol{v} \in R_{VS(s_i),w_{i+1}}$.*

In other words, a run over a point $\boldsymbol{v}$ is obtained by starting from the initial implicit state $s_0$, and repeatedly moving from an implicit state to another according to the deterministic decision structures induced by the transition function $\delta$, which amounts to following the relation $\to$. A run ends when it finally reaches an implicit state whose associated vector space contains $\boldsymbol{v}$. A point may admit multiple runs. In order to be able to solve the point decision and point classification problems by following a single run in an IRVA, we introduce the following semantical integrity constraint.

**Definition 12.** *An IRVA* $(n, S_I, S_E, s_0, \delta, \mathit{VS}, \mathit{col})$ *is* well-formed *if it is syntactically valid, and for every* $\boldsymbol{v} \in \mathbb{R}^n$*, each of its runs over* $\boldsymbol{v}$ *ends up in the same implicit state.*

From now on, we will only consider well-formed IRVA. Solving the point decision or classification problems on such IRVA thus reduce to following an arbitrary run over the point of interest. The answer is then provided by the color of the implicit state that is finally reached by this run.

We are now ready to define the semantics of an IRVA $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathit{VS}, \mathit{col})$, i.e., to describe the partitioning, or coloring, function $\Pi : \mathbb{R}^n \to \mathbb{N}_{>0}$ that it represents. Since the transition relation $\to$ between implicit states is acyclic, we can consider these implicit states in bottom-up order, associating a coloring function $\Pi_s$ to each $s \in S_I$. This procedure can be started from the states $s \in S_I$ such that $\dim(\mathit{VS}(s)) = n$, which do not have successors by $\to$, and will eventually end up in the initial state $s_0$. The coloring function $\Pi_{s_0}$ of $s_0$ will then provide the partition represented by the IRVA. The procedure relies on the following result.

**Theorem 13.** *Let* $s \in S_I$ *be an implicit state of* $\mathcal{A}$*. Its coloring function* $\Pi_s$ *is such that*

- $\Pi_s(\boldsymbol{v}) = \mathit{col}(s)$ *for all points* $\boldsymbol{v} \in \mathit{VS}(s)$*,*
- $\Pi_s(\boldsymbol{v}) = \Pi_{s'}(\boldsymbol{v})$ *for all states* $s' \in S_I$ *and points* $\boldsymbol{v} \in \mathbb{R}^n$ *such that* $s \xrightarrow{w} s'$ *and* $\boldsymbol{v} \in R_{\mathit{VS}(s),w}$*, for some word* $w \in \pm(\mathbb{N}_{>0})\{0,1\}^*$*.*

In summary, we have shown constructively how to build a coloring function that describes the semantics of a given IRVA. We thus have the following theorem.

**Theorem 14.** *Every well-formed IRVA* $(n, S_I, S_E, s_0, \delta, \mathit{VS}, \mathit{col})$ *represents a pyramidal partition of* $\mathbb{R}^n$*.*

## 6 Canonicity

### 6.1 Canonical IRVA Representations

We now show that, for every pyramidal partition $\Pi$ of $\mathbb{R}^n$ with apex $\mathbf{0}$, there exists an IRVA $\mathcal{A}$ that represents it, and that this IRVA can be defined canonically up to equality of vector spaces and isomorphism of transition graphs. We will then develop an algorithm for transforming any IRVA that represents $\Pi$ into this canonical form.

The first step consists in defining the set of implicit states $S_I$ of $\mathcal{A}$, by creating one implicit state $s_i$ for each polyhedral component $C_i$ of $\Pi$. From Theorem 7, we know that $\Pi$ admits a unique minimum component $C_0$ with respect to the incidence relation $\preceq$. The corresponding implicit state $s_0$ becomes the initial state of $\mathcal{A}$. For each $s_i \in S_I$, the vector space $\mathit{VS}(s_i)$ is then made equal to $\mathrm{aff}(C_i)$, and the color $\mathit{col}(s_i)$ takes the (common) value of $\Pi(\boldsymbol{v})$ for the points $\boldsymbol{v} \in C_i$.

It remains to define the decision structures that link the implicit states. Recall that, for an implicit state $s_i \in S_I$ and a word $w \in \pm(\{1, 2, \ldots, n - m\})\{0, 1\}^*$, where $m = \dim(\mathit{VS}(s_i))$, it is possible to have $s_i \overset{w}{\to} s_j$, with $s_j \in S_I$, iff the set of components $\{C_k \mid (C_i \preceq C_k) \wedge (R_{\mathit{VS}(s_i),w} \cap C_k \neq \emptyset)\}$ admits $C_j$ as unique minimum element (with respect to $\preceq$). Since our aim is to obtain a canonical decision structure, it is natural to only select the shortest words $w$ for which such a decision is possible, i.e., the transition $s_i \overset{w}{\to} s_j$ will be considered only if there does not exist a shorter prefix $u$ of $w$ for which $s_i \overset{u}{\to} s_j$ is possible. By applying this reasoning to all pairs of implicit states, one finally obtains a canonical form of the acyclic labeled transition relation $\to$ linking these states.

From the labeled relation $\to$, it is straightforward to define the explicit states of the canonical IRVA representing $\Pi$. This can be achieved by building a deterministic finite-state automaton with a transition relation corresponding to $\to$, considering that each implicit state has a unique distinguished accepting status. This automaton can then be minimized into a canonical form using classical techniques [12]. The implicit states are preserved by this operation, and the other states of the minimized automaton become the explicit states of the canonical IRVA. The transition function $\delta$ is then directly given by the transition relation of the minimized automaton.

## 6.2 Minimization Algorithm

We now sketch an algorithm for computing, from an IRVA $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathit{VS}, \mathit{col})$, a canonical IRVA that represents the same pyramidal partition. The idea is to exploit the acyclic structure of the transition graph, by inspecting the implicit and explicit states of $\mathcal{A}$ one by one in bottom-up order. Each step of the minimization procedure consists in examining one state $s \in S_I \cup S_E$, in order to determine whether it can be merged with another state $s'$ that has already been processed, or left otherwise unchanged. When a state $s$ is merged into a state $s'$, all its incoming transitions are redirected to $s'$, and its outgoing transitions are orphaned. This may leave unreachable states in the resulting IRVA, which are easily removed by a subsequent cleaning step. Thanks to the order in which the states are processed during minimization, the states $s$ considered at each step are such that their successors by the transition function have already undergone minimization. The minimization of IRVA thus proceeds quite similarly to the usual minimization algorithms for acyclic finite-state automata or for binary decision diagrams [8]. The precise rules for deciding whether states should be merged are however specific to IRVA.

A first situation occurs when the state $s$ under scrutiny happens to have identical successors as an already processed state $s'$, i.e., for every symbol $a$, one has $\delta(s, a) = s''$ iff $\delta(s', a) = s''$. In this case, if $s$ is an explicit state, then it can be merged into $s'$. If $s$ is an implicit state, one additionally has to check the conditions $\mathit{VS}(s) = \mathit{VS}(s')$ and $\mathit{col}(s) = \mathit{col}(s')$ before merging $s$ with $s'$.

The next rule is more complex. Consider an explicit state $s \in S_E$, with an outgoing decision structure leading to the set of implicit states $\{s_1, s_2, \ldots, s_k\} \subset S_I$. If this set contains a unique minimum element $s'$ with respect to the transition relation $\rightarrow^*$, i.e., if $s \rightarrow^* s_i$ for all $i \in \{1, 2, \ldots, k\}$, then the decision structure leaving $s$ is redundant, since all its paths can be redirected towards $s'$ without affecting the semantics of $\mathcal{A}$. In such a case, the state $s$ itself is redundant and can be merged with $s'$. It can be shown that this situation only occurs when the implicit state $s'$ is a direct successor of $s$; this property may be exploited for speeding up the search for the states $s_i$.

Finally, a similar rule applies to implicit states $s \in S_I$. Let $\{s_1, s_2, \ldots, s_k\} \subset S_I$ be the set of implicit states that are directly reachable from $s$ by the transition relation $\rightarrow$. If this set admits a unique minimum element $s'$ with respect to $\rightarrow^*$, then $s$ can be merged with $s'$ provided that two conditions are satisfied. First, the color of both states must match: $col(s) = col(s')$. Second, in the particular case where one has $\dim(VS(s')) = \dim(VS(s)) + 1$, it is essential to ensure that the state $s'$ does not represent a boundary of the polyhedral component associated to $s$. This is done by checking that, among the words $w$ such that $s \overset{w}{\rightarrow} s'$, at least two of them have leading symbols $-(i)$ and $+(i)$ with an identical face number and opposite polarities. This tricky particular case was overlooked in [3].

## 7 Combination Operation

Our goal is now to develop an algorithm for combining two IRVA $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively representing pyramidal partitions $\Pi_1$ and $\Pi_2$ of $\mathbb{R}^n$. Recall that these partitions can be seen as functions $\Pi_1 : \mathbb{R}^n \rightarrow \{1, 2, \ldots, m_1\}$ and $\Pi_2 : \mathbb{R}^n \rightarrow \{1, 2, \ldots, m_2\}$ that assign colors to the points in $\mathbb{R}^n$. In order to combine $\Pi_1$ and $\Pi_2$, we need a *combination function* $c : \{1, 2, \ldots, m_1\} \times \{1, 2, \ldots, m_2\} \rightarrow \{1, 2, \ldots, m\}$, where $m \in \mathbb{N}_{>0}$, that maps every pair of colors from $\Pi_1$ and $\Pi_2$ onto a single one. We have the following definition.

**Definition 15.** *The* combination *of $\Pi_1$ and $\Pi_2$ induced by $c$ is the pyramidal partition $\Pi = \Pi_1 \bowtie_c \Pi_2$ over $\mathbb{R}^n$ such that $\Pi(\boldsymbol{v}) = c(\Pi_1(\boldsymbol{v}), \Pi_2(\boldsymbol{v}))$ for every $\boldsymbol{v} \in \mathbb{R}^n$.*

Note that, in the particular case of binary partitions, this definition covers the computation of intersections, unions, and differences of sets, by choosing a combination function that corresponds to the appropriate Boolean operator.

The construction that we are about to describe shares some similarities with the computation of the product of two finite-state automata. The idea is to construct incrementally an IRVA $\mathcal{A}$ representing $\Pi = \Pi_1 \bowtie_c \Pi_2$, starting from its initial state and developing its transition function step by step. Each implicit state $s$ of $\mathcal{A}$ corresponds to a pair $(s_1, s_2)$, where $s_1$ (resp. $s_2$) is an implicit state of $\mathcal{A}_1$ (resp. $\mathcal{A}_2$). The polyhedral component of $\Pi$ represented by $s$ corresponds to the points $\boldsymbol{v} \in \mathbb{R}^n$ that simultaneously belong to the component $C_1$ of $\Pi_1$ represented by $s_1$, and to the component $C_2$ of $\Pi_2$ represented by $s_2$. As a consequence, one has $VS(s) = VS(s_1) \cap VS(s_2)$ and $col(s) = c(col(s_1), col(s_2))$. The initial state of $\mathcal{A}$ is the first to be created, by pairing the initial states of $\mathcal{A}_1$ and $\mathcal{A}_2$.

During the construction of $\mathcal{A}$, the creation of a new implicit state $s$ is followed by the development of its outgoing decision structure. This is done by exploring the prefixes that can be read from $s$ in breadth-first order. For such a prefix $w$, one checks whether $\mathcal{A}$ admits an implicit state $s'$ such that $s \xrightarrow{w} s'$. This check is carried out by first computing the convex region $R_{VS(s),w}$, and then determining whether this region covers a unique minimum component incident to $C_1$ in $\Pi_1$, as well as one incident to $C_2$ in $\Pi_2$ (with respect to the incidence relations $\preceq_1$ and $\preceq_2$ of these pyramids). If a suitable implicit state $s'$ exists, then it either corresponds to a previously computed state of $\mathcal{A}$, or to a new state that needs to be created. Otherwise, the decision labeled by $w$ has to be developed further.

A key operation in the previous procedure is thus, given the IRVA $\mathcal{A}_i$ representing the pyramid $\Pi_i$, with $i \in \{1, 2\}$, a convex region $R \subseteq \mathbb{R}^n$, and an implicit state $s_i$ of $\mathcal{A}_i$ representing a component $C_i$ of $\Pi_i$, to determine whether the set of components of $\Pi_i$ that are incident to $C_i$ and have a non-empty intersection with $R$ admits a unique minimum element $C'_i$. We encapsulate this operation in a function $minel(\mathcal{A}_i, R, s_i)$ that returns the implicit state of $\mathcal{A}_i$ representing $C'_i$ if it exists, and $\perp$ otherwise. The value of $minel(\mathcal{A}_i, R, s_i)$ can be computed as follows. First, one explores the implicit states $s'_i$ of $\mathcal{A}_i$ that are reachable from $s_i$, i.e., for which there exist words $w_1$, $w_2$, $\ldots w_k$ and implicit states $q_1$, $q_2, \ldots q_{k-1}$, with $k > 0$, such that $s_i \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \xrightarrow{w_3} \cdots \xrightarrow{w_{k-1}} q_{k-1} \xrightarrow{w_k} s'_i$. All such states $s'_i$ for which the intersection $R \cap R_{VS(s_i),w_1} \cap R_{VS(q_1),w_2} \cap \ldots \cap R_{VS(q_{k-1}),w_k} \cap VS(s'_i)$ is non-empty are collected in a set $U$. If $U$ contains a minimum state $s'_i$ with respect to $\to^*$, i.e., such that $s'_i \to^* s''_i$ for every $s''_i \in U$, then one has $minel(\mathcal{A}_i, R, s_i) = s'_i$. Otherwise, the function returns $minel(\mathcal{A}_i, R, s_i) = \perp$.

The procedure for combining two IRVA is formalized in Algorithm 1. In addition to $minel$, this procedure relies on a function $succ(s)$ that returns the alphabet of symbols that can potentially be read from a state $s$, a function $new()$ that instantiates new explicit states, and the usual *push*, *pop* and *empty*? operations on stacks. The algorithm relies on a stack for storing the states of $\mathcal{A}$ whose outgoing decision structures still need to be developed. This stack contains tuples $(s_1, s_2, s, s_I, w)$, where $s$ is such a state, $s_1$ and $s_2$ are the current states reached in respectively $\mathcal{A}_1$ and $\mathcal{A}_2$, $s_I$ is the last visited implicit state in $\mathcal{A}$, and $w$ is the word read from $s_I$ to $s$.

## 8 Conclusions and Perspectives

We have studied a data structure, the Implicit Real Vector Automaton (IRVA), that can be used for representing symbolically polyhedra or polyhedral partitions in $\mathbb{R}^n$. IRVA are not limited to convex or regularized polyhedra, admit an easily computable minimal form, and are closed under Boolean operators.

IRVA imitate the principles of operation of Real-Vector Automata (RVA), another automata-based data structure suited for polyhedra, but can be considerably more concise. IRVA inherit from RVA very efficient algorithms for solving the point decision and classification problems, which is a substantial advantage compared to other symbolic representations of polyhedra, in particular Selective Nef Complexes [10].

```
input  : Two IRVA $\mathcal{A}_1 = (n_1, S_{I1}, S_{E1}, s_{01}, \delta_1, VS_1, col_1)$ and
         $\mathcal{A}_2 = (n_2, S_{I2}, S_{E2}, s_{02}, \delta_2, VS_2, col_2)$, a combination function $c$
output: An IRVA $\mathcal{A} = (n, S_I, S_E, s_0, \delta, VS, col)$
$s_0 := (s_{01}, s_{02})$;
$S_I := \{s_0\}$;
$S_E := \emptyset$;
$VS(s_0) := VS_1(s_{01}) \cap VS_2(s_{02})$;
$col(s_0) := c(col_1(s_{01}), col_2(s_{02}))$;
$stack := \emptyset$;
push $(stack, (s_{01}, s_{02}, s_0, s_0, \varepsilon))$;
while not(empty?$(stack)$) do
    $(s_1, s_2, s, s_I, w) :=$ pop$(stack)$;
    foreach $a \in$ succ$(s)$ do
        $m_1 :=$ minel$(\mathcal{A}_1, R_{VS(s_I),wa}, s_1)$;
        $m_2 :=$ minel$(\mathcal{A}_2, R_{VS(s_I),wa}, s_2)$;
        if $m_1 \neq \perp$ and $m_2 \neq \perp$ then
            $R := VS_1(m_1) \cap VS_2(m_2)$;
            if $R \cap R_{VS(s_I),wa} = \emptyset$ or $\dim(VS(s_I)) \geq \dim(R)$ then
                $s_N :=$ new();
                $S_E := S_E \cup \{s_N\}$;
                $\delta(s, a) := s_N$;
                push$(stack, (m_1, m_2, s_N, s_I, wa))$;
            else
                if $(m_1, m_2) \notin S_I$ then
                    $S_I := S_I \cup \{(m_1, m_2)\}$;
                    $VS((m_1, m_2)) := R$;
                    $col((m_1, m_2)) := c(col_1(m_1), col_2(m_2))$;
                    push$(stack, (m_1, m_2, (m_1, m_2), (m_1, m_2), \varepsilon))$;
                end
                $\delta(s, a) := (m_1, m_2)$;
            end
        else
            if $m_1 = \perp$ then $m_1 := s_1$;
            if $m_2 = \perp$ then $m_2 := s_2$;
            $s_N :=$ new();
            $S_E := S_E \cup \{s_N\}$;
            $\delta(s, a) := s_N$;
            push$(stack, (m_1, m_2, s_N, s_I, wa))$;
        end
    end
end
```

**Algorithm 1.** Computation of $\mathcal{A}_1 \bowtie_c \mathcal{A}_2$

Future work will address the implementation of a package for building and manipulating IRVA, the assessment of their performances in actual applications, and the computation of additional operations such as projecting polyhedra and converting IRVA to and from other representations.

## References

1. Bieri, H., Nef, W.: Elementary Set Operations with $d$-Dimensional Polyhedra. In: Noltemeier, H. (ed.) CG-WS 1988. LNCS, vol. 333, pp. 97–112. Springer, Heidelberg (1988)
2. Boigelot, B., Bronne, L., Rassart, S.: An improved reachability analysis method for strongly linear hybrid systems. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 167–177. Springer, Heidelberg (1997)
3. Boigelot, B., Brusten, J., Degbomont, J.-F.: Implicit real vector automata. In: Proc. INFINITY. Electronic Proceedings in Theoretical Computer Science, vol. 39, pp. 63–76 (2010)
4. Boigelot, B., Brusten, J., Leroux, J.: A Generalization of Semenov's Theorem to Automata Over Real Numbers. In: Schmidt, R.A. (ed.) CADE-22. LNCS, vol. 5663, pp. 469–484. Springer, Heidelberg (2009)
5. Boigelot, B., Jodogne, S., Wolper, P.: An effective decision procedure for linear arithmetic over the integers and reals. ACM Transactions on Computational Logic 6(3), 614–633 (2005)
6. Bournez, O., Maler, O., Pnueli, A.: Orthogonal Polyhedra: Representation and Computation. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) HSCC 1999. LNCS, vol. 1569, pp. 46–60. Springer, Heidelberg (1999)
7. Brusten, J.: On the sets of real vectors recognized by finite automata in multiple bases. Ph.D. thesis, University of Liège (2011)
8. Bryant, R.E.: Symbolic Boolean manipulation with ordered binary decision diagrams. ACM Computing Surveys 24(3), 293–318 (1992)
9. Dutertre, B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
10. Hachenberger, P., Kettner, L., Mehlhorn, K.: Boolean operations on 3D selective Nef complexes: Data structure, algorithms, optimized implementation and experiments. Comput. Geom. 38(1–2), 64–99 (2007)
11. Halbwachs, N., Raymond, P., Proy, Y.E.: Verification of Linear Hybrid Systems by Means of Convex Approximations. In: LeCharlier, B. (ed.) SAS 1994. LNCS, vol. 864, pp. 223–237. Springer, Heidelberg (1994)
12. Hopcroft, J.: An n logn algorithm for minimizing states in a finite automaton. Tech. rep., Stanford (1971)
13. Le Verge, H.: A note on Chernikova's algorithm. Tech. rep., IRISA, Rennes (1992)
14. Löding, C.: Efficient minimization of deterministic weak $\omega$-automata. Information Processing Letters 79(3), 105–109 (2001)
15. Requicha, A.A.G.: Representations for rigid solids: Theory, methods, and systems. ACM Comput. Surv. 12(4), 437–464 (1980)
16. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of constraint programming. Elsevier (2006)

# Around the Physical Church-Turing Thesis: Cellular Automata, Formal Languages, and the Principles of Quantum Theory

Gilles Dowek

INRIA, 23 avenue d'Italie, CS 81321, 75214 Paris Cedex 13, France
gilles.dowek@inria.fr
http://www-roc.inria.fr/who/Gilles.Dowek/

**Abstract.** The physical Church-Turing thesis explains the Galileo thesis, but also suggests an evolution of the language used to describe nature. It can be proved from more basic principle of physics, but it also questions these principles, putting the emphasis on the principle of a bounded density of information. This principle itself questions our formulation of quantum theory, in particular the choice of a field for the scalars and the origin of the infinite dimension of the vector spaces used as state spaces[1].

## 1 The Church-Turing Thesis and Its Various Forms

### 1.1 Why a Thesis?

It is a quite common situation in mathematics, that a notion, first understood intuitively, receives a formal definition at some point. For instance, the notion of a real number has been understood intuitively in geometry, for instance as the length of a segment, before it has been formally defined in the 19th century, by Cauchy and Dedekind. Another example is the notion of an algorithm, that has been understood intuitively for long, before a formal definition of the notion of a *computable function* has been given in the 30s, by Gödel and Herbrand, Church, Turing, and others.

There is however a difference between these two notions. When the notion of a real number was defined, there seems to have been no real discussion of the fact that the formal definition of Cauchy or Dedekind was indeed a formalization of our intuitive notion of a real number. In the same way, there is no discussion of the fact that the formal definitions of the notions of a triangle, a square, or a circle are indeed formalizations of our intuitive notions of a triangle, a square, or a circle. Even for the formal definitions of the notions of a distance and of orthogonality, based on the notion of an inner product, we finally agree that they correspond to our intuitive notion of a distance and of orthogonality.

[1] These tutorial notes owe a lot to Pablo Arrighi with whom most of this research has been done and to Maël Pégny with whom I gave a course on the Church-Turing thesis in the PUC Rio in 2011, on the invitation of Luiz Carlos Pereira. I also want to thank Olivier Bournez, José Félix Costa, Nachum Dershowitz, Jean-Baptiste Joinet, Giuseppe Longo, and Thierry Paul for many discussion on this thesis.

The situation is different with the notion of a computable function, as the formal definition of this notion came with a thesis that this formal definition indeed captures our intuitive notion of computability.

This peculiarity of the notion of a computable function can be explained by its history. Before the modern definition, another had been given: the notion of a primitive recursive function, and it was then noticed that there existed functions, such as the Ackermann function, that were intuitively computable, but not primitive recursive. This has raised a reasonable doubt that history may repeat itself and that another counterexample may be given to this new definition.

## 1.2 The Physical Church-Turing Thesis

Stated as such, the epistemological status of the Church-Turing thesis is unclear. It cannot be proved, as it uses a notion that is not formally defined: that of a function computable in the intuitive sense. It cannot be experimentally tested as it is not a statement about nature. But it could be falsified, if anyone came with a counterexample, *i.e.* a function that is not computable in the formal sense, but such that everyone agrees that it is computable in the intuitive sense, as it happened with the Ackermann function. Yet, this requirement of a consensus on the computability, in the intuitive sense, of this function puts it at the border of falsifiability. Thus, stated as such, the Church-Turing thesis is hardly a thesis, and several attempts have been made to replace this notion of computability in the intuitive sense by a more precise notion. These attempts go at least in three directions.

- The first is to axiomatize a computability predicate and to prove the equivalence of this implicitly defined notion with that defined by Gödel and Herbrand, Church, Turing, *etc.* This is the way taken by Dershowitz and Gurevich [19], and others.
- The second is to define a notion of computability based on the analysis of the operations executed by a human being performing a computation and to prove the equivalence of this notion with that defined by Gödel and Herbrand, Church, Turing, *etc.* This is the way taken by Turing himself, and others.
- The third is to define a notion of computability based on the analysis of the operations executed by a machine performing a computation and to prove the equivalence of this notion with that defined by Gödel and Herbrand, Church, Turing, *etc.* This is the way taken by Gandy [26], and others.

The thesis expressing that any function computable by a machine is computable in the sense of Gödel and Herbrand, Church, Turing, *etc.* is called the *physical Church-Turing thesis.*

Giving a meaning to this thesis requires to give a definition of the notion of a machine. A *machine* is any physical system, equipped with a interaction protocol defining a way to communicate some information to the system by choosing some parameters $a = \langle a_1, ..., a_n \rangle$, when preparing the system, and to extract some information from the system, by measuring some values $b = \langle b_1, ..., b_p \rangle$.

Such a machine defines a relation: the relation $R$ such that $a\ R\ b$ if $b$ is a possible result for the measurements, when the chosen parameters are $a$. We say that this relation $R$ is *realized* by this machine. For instance, if one applies an electrical tension $U$ to a conductor of resistance $R$ and measures the current $I$ passing through this conductor, then $a = \langle U, R \rangle$ and $b = I$ and the realized relation is that relating $\langle U, R \rangle$ and $I$ when $U = RI$. In the same way, if one lets a body freely falling in vacuum for some time $t$ and measures the position $x$ of this body, the realized relation is that relating $t$ and $x$ when $x = \frac{1}{2}gt^2$.

The physical Church-Turing thesis expresses that relations realized by a machine are computable.

The physical Church-Turing thesis is obviously a thesis about nature. Thus its correctness depends on the world we are in. It is easy to imagine worlds where this thesis is valid and worlds where it is not. In particular this thesis is not valid in worlds where accelerating machines or infinitely parallel machines can be built. The theory of hypercomputability studies such worlds.

Yet, not everyone agrees on the epistemological status of this thesis. According to some, such as Deutsch [20], this thesis — or its negation — is a principle of physics, like the homogeneity of space or the homogeneity of time. According to others, such as Gandy, this thesis — or its negation — must be established as a consequence of other principles of physics. There is no real contradiction between these two points of view, as we know that the choice of axioms in a theory is somewhat arbitrary. For instance, in set theory, we may chose a particular form of the axiom of choice and prove the others. Yet, some propositions are simple enough to be taken as axioms or principles, and others are not.

## 2 Beyond Natural Numbers

Computability is primarily defined for the functions from the set of natural numbers to itself, but many functions we use when discussing the physical Church-Turing thesis are defined on other domains. Thus, the notion of computability must be defined beyond the set of natural numbers.

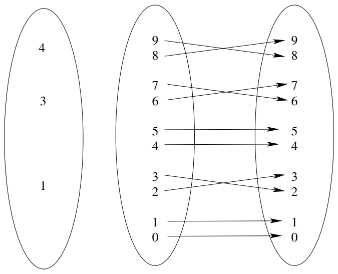
### 2.1 Indexings

The first way to define a notion of computability on a set $S$ different from the natural numbers is to transport the notion of computability from the set of real numbers to the set $S$ with the use of an *indexing*, *i.e.* a bijection $\ulcorner . \urcorner$ between $S$ and a subset of $\mathbb{N}$.

Then, a function $f$ from $S$ to $S$ is said to be computable, if there exists a computable function $\hat{f}$ from $\mathbb{N}$ to $\mathbb{N}$ such that for all $x$ in $S$, $\ulcorner f(x) \urcorner = \hat{f}(\ulcorner x \urcorner)$.

An objection to this method, due to Montague [28], is that the defined notion of computability depends on the choice of the indexing. To see why, let us consider an undecidable set $U$, that does not contain 0 and the function $D$ mapping

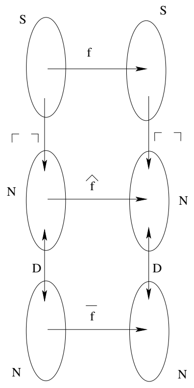
- $2n$ to $2n + 1$ and $2n + 1$ to $2n$ if $n \in U$,
- $2n$ to $2n$ and $2n + 1$ to $2n + 1$ otherwise.



This function is a non computable bijection from $\mathbb{N}$ to $\mathbb{N}$. Moreover it is involutive, *i.e.* $D^{-1} = D$, $D(0) = 0$, and $D(1) = 1$.

Then, consider a set $S$, an indexing $\ulcorner . \urcorner$ of this set, elements $u$ and $v$ in $S$ such that $\ulcorner u \urcorner = 0$ and $\ulcorner v \urcorner = 1$, and the function $f$ from $S$ to $S$ defined by $f(x) = u$ if $\ulcorner x \urcorner$ is is even and $f(x) = v$ otherwise. As the function $\hat{f}$ from $\mathbb{N}$ to $\mathbb{N}$ defined by $\hat{f}(n) = 0$ if $n$ is even and $\hat{f}(n) = 1$ otherwise is computable, the function $f$ is computable *with respect to the indexing* $\ulcorner . \urcorner$.

But, the function $D \circ \ulcorner . \urcorner$ is also an indexing of $S$



and the function $\overline{f} = D \circ \hat{f} \circ D$ is not computable. Indeed $\overline{f}(2n) = 1$ if and only if $D(\hat{f}(D(2n))) = 1$ if and only if $D(2n)$ is odd if and only if $n \in U$. Thus, the function $f$ is not computable *with respect to the indexing* $D \circ \ulcorner . \urcorner$. The computability of $f$ therefore depends on the chosen indexing.

Thus, unless we have a canonical way to represent the parameters $a$ and the measured values $b$ by natural numbers, the formulation of the physical Church-Turing thesis is relative to the choice of an indexing.

Three answers have been given to this objection.

**Articulation.** A list $a_1, a_2, ..., a_k$ of natural numbers has a *canonical index*

$$\ulcorner a_1, a_2, ..., a_k \urcorner = (a_1; (a_2; (...; (a_k; 0))))$$

where ; is the bijection between $\mathbb{N}^2$ and $\mathbb{N} \setminus \{0\}$ defined by $(n;p) = (n+p)(n+p+1)/2+p+1$. And a tree indexed by natural numbers $p(t_1,...,t_k)$ has a *canonical index*

$$\ulcorner p(t_1,...,t_k)\urcorner = (p;\ulcorner t_1\urcorner;...;\ulcorner t_k\urcorner;0)$$

This permits to define indexings for all articulated sets of trees, where a set of tree is *articulated* if it is $n$-articulated for some $n$, $0$-*articulated* if all its elements are labeled in a finite set and $(n+1)$-*articulated* if all its elements are labeled in a $n$-articulated set of trees. For instance, proofs are trees labeled by sequents, that are trees labeled by formulae that are trees labeled by function symbols, predicate symbols, and variables, that are trees labeled in a finite set. Thus the set of proofs is articulated. In a similar way, all the sets of syntactic objects, formulae, proofs, programs, *etc.*, are articulated.

Let $S$ be an articulated set of trees, ultimately labeled in the finite set $E$. Using the canonical indexing of trees above, we can transform any indexing $\ulcorner.\urcorner^0$ of the set $E$ into an indexing $\ulcorner.\urcorner^n$ of $S$ by

$$\ulcorner f(t_1,...,t_k)\urcorner^{r+1} = (\ulcorner f\urcorner^r;\ulcorner t_1\urcorner^{r+1};...;\ulcorner t_k\urcorner^{r+1};0)$$

As all finite functions are computable, the choice of the indexing $\ulcorner.\urcorner^0$ is immaterial and all the indexings based on this canonical tree indexing define the same set of computable functions.

**Finite Generation.** This argument has been generalized by Rabin [30] to all finitely generated algebraic structures.

In mathematics, we are not interested in sets *per se*, but in *structures*, *i.e.* sets equipped with operations: $\langle\mathbb{N}, S\rangle$, $\langle\mathbb{N}, +, \times\rangle$, $\langle L, ::\rangle$, $\langle L, @\rangle$, *etc.* As their name suggests, these operations must be computable. Thus, when indexing a structure, we must restrict to the *admissible* indexings: those that make these operations computable.

A structure is said to be *finitely generated* when all its elements can be built from a finite number of elements with the operations of the structure, for instance the set of natural numbers equipped with addition, the set of lists of elements of a finite set equipped with concatenation, the set of rational numbers equipped with addition, multiplication and division, *etc.* are finitely generated structures.

When a structure is finitely generated, if $i$ and $j$ are two admissible indexings, then there exists a computable function $f$ such that $j = f \circ i$. Thus, all admissible indexings define the same notion of computability: computability is *stable*.

In particular computability is stable on all the finite extensions of the field of rational numbers, all the finite-dimensional vector spaces over such fields, *etc.* [3].

**Computability of Sets of Functions.** Programs in a given programming language are lists of letters in a finite alphabet. We have seen that, as the set of lists of elements of a finite set equipped with concatenation @ is finitely generated, all its admissible indexings define the same set of computable functions, and the characteristic function $\chi$ of the set of terminating programs is non computable, relatively to any admissible indexing. In other words, for any indexing, if the function @ is computable, then the function $\chi$ is not.

Boker and Dershowitz [11] have suggested that this result could be stated without focusing on the concatenation operation @, as the fact that the functions @ and $\chi$ cannot be both computable, with respect to the same indexing. Calling a set of functions *computable*, if there exists an indexing with respect to which all the elements of the set are computable, the set $\{@, \chi\}$ is non computable, in an absolute sense.

The set $\{\chi\}$ is computable as it is possible to cheat on the indexing and associate an even number to terminating programs and an odd number to non terminating ones, but the price to pay is to make concatenation non computable.

Another way to formulate this result is that the set $C$ of functions computable with respect to an admissible indexing is a maximal computable set of functions. Indeed, let $\phi$ be a function that is not in $C$ and $i$ an indexing such that $\phi$ is computable relatively to $i$. Then, $i$ is not admissible and the concatenation operation is not computable with respect to $i$. Thus, there exists a function of $C$ that is not computable with respect to $i$ and the set of functions computable with respect to $i$ is not a proper superset of $C$.

In other words, changing the indexing can change the set of computable function, but it cannot extend it.

As a consequence, if we formulate the physical Church-Turing thesis as the fact that the set of functions realized by a machine is the set of computable functions, we obtain a thesis that depends on the way we interpret the physical parameters and the measured values. And this thesis can be invalidated by building a machine that computes the parity of a natural number and interpreting it as a machine deciding the halting problem, by cheating on the indexing.

But if we formulate this thesis as the fact that the set of functions realized by a machine is a strict superset of the set of computable functions for no indexing, then we have a thesis that is independent on the choice of an indexing.

## 2.2 Real Numbers

The use of an indexing permits to transport the notion of computability to countable sets only. Computability over non countable sets is an unexplored world, except for one set: that of real numbers. As real numbers might be useful in physics, we have to say a few words about the computability of real functions.

To know a real number $x$ is not to be able to answer the query "give me all the digits of $x$", but to be able to answer the query "give me the $n^{th}$ first digits of $x$" for all $n$, or equivalently to the query "give me a rational approximation of $x$ with an error less than $\varepsilon$" for all $\varepsilon$ [29,33].

To know a real function $f$ is to be able to answer all the queries of the form "give me a rational approximation of $f(x)$ with an error less than $\varepsilon$" provided we are answered all queries of the form "give me a rational approximation of $x$ with an error less than $\eta$".

Thus a function $f$ from $\mathbb{R}$ to $\mathbb{R}$ is computable if there exists a computable function $F$ from $\mathbb{Q} \times (\mathbb{Q}^+ \setminus \{0\})$ to $\mathbb{Q} \times (\mathbb{Q}^+ \setminus \{0\} \cup \{+\infty\})$ such that for all $x$ in $\mathbb{R}$, $q$ and $r$ in $\mathbb{Q}$, $\eta$ and $\varepsilon$ in $\mathbb{Q}^+ \setminus \{0\}$

$$(F(q,\eta) = (r,\varepsilon) \text{ and } |x-q| \leq \eta) \Rightarrow |f(x) - r| \leq \varepsilon$$

and if $(q_n)_n$, $(\eta_n)_n$, are sequences such that for all $n$, $|x - q_n| \leq \eta_n$ and the sequence $\eta$ goes to 0 at infinity, then, calling $(r_n, \varepsilon_n) = F(q_n, \eta_n)$, the sequence $\varepsilon$ also goes to 0 at infinity.

### 2.3 Non-determinism

Computability theory is a theory of computable functions. However, when describing a machine we spoke, not about functions, but about relations. This is because several sequences of measured values $b$ may correspond to the same sequence of parameters $a$. This non-determinism may have different sources, it may either come from the absolute non-determinism of nature, or because some variables are kept hidden: for instance, in the experiment above, if we choose the resistance $R$ and measure the intensity $I$, then $I$ does not depend functionally on $R$.

We know that a non deterministic algorithm from a set $A$ to a set $B$ may always be considered as a deterministic algorithm from the set $A$ to the powerset $\mathcal{P}(B)$: a relation $R$ may always be considered as a function mapping $a$ to the set $R_a = \{b \in B \mid a \; R \; b\}$. If we represent the set $R_a$ by an algorithm computing its partial characteristic function, then a relation $R$ may be called computable when there exists an algorithm mapping $a$ to an algorithm mapping $b$ to 1 if and only if $a \; R \; b$. Using the fact that a function taking values in a functional space is equivalent to a function of several arguments, the relation $R$ is represented by its own partial characteristic function. Thus, a relation is computable if and only if it is semi-decidable.

We could also represent the set $R_a$ by an enumeration function and this would lead to define computable relations as effectively enumerable relations. On natural numbers, and on any set on which equality is decidable, these two notions coincide. This is not the case on real numbers and only the second notion makes sense in this case [21].

## 3 Gandy's Proof

As previously noticed, some statements are too complex to be taken as axioms or principles and must rather be derived from more basic ones. This is probably the case with the physical Church-Turing thesis. And indeed Gandy [26] has shown how to derive this thesis from three more basic hypotheses about nature.

### 3.1 Gandy's Hypotheses

Gandy's hypotheses are

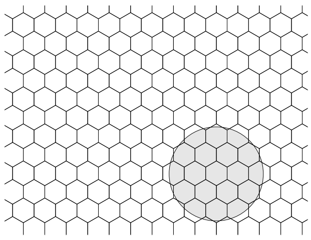
– the homogeneity of space and time,
– the boundedness of the velocity of propagation of information,
– the boundedness of the density of information.

The boundedness of the velocity of propagation of information can be expressed as the fact that the state of a system in one place can only affect the state of a system in another only after a delay, proportional to their distance. This principle has been part of the principles of physics, at least since special relativity.

The boundedness of the density of information can be expressed as the fact that a physical system of finite diameter has a finite state space. This principle is a more original idea. We may see it as an abstract formulation of the quantization idea: the fact that a variable cannot take any real number as a value, but only one of a discrete set. A similar principle has been stated more explicitly by Bekenstein [8].

### 3.2 Gandy's Proof

Then, to prove the physical Church-Turing thesis from these hypotheses, we just need to partition the space into an infinite number of identical cells of finite diameter. Because information has a bounded density, the state space of each cell is finite. Because space is homogeneous, this state space is the same for all cells. At the origin of time all the cells except a finite number are in a particular *quiescent* state. Like space, time can be discretized. Because the velocity of propagation of information is bounded and space and time are homogeneous, the state of a cell at a given time step is function of the state of a finite number of neighbors cells at the previous time step.



This function of a finite number of variables varying in a finite set is computable. Hence the state of each cell at each time step can be computed from the initial state.

In other words, using Gandy's hypotheses, a tiling of space and time allows to describe the evolution of any system as a cellular automata [35]. Of course, the choice of the tiling should not affect the evolution of the system, as the tiling is not a property of the system itself, but of our description of the system.

### 3.3 Criticizing Gandy's Hypotheses

Gandy's hypotheses can be, and have been [18], criticized. For instance, it is well-known that, in Newtonian mechanics, gravity is instantaneous and thus information can be propagated with an infinite velocity. Also, the position of a body between two points $A$ and $B$, the distance $AB$ taken as a unit, can be any real number between 0 and 1 and thus can contain an infinite quantity of information: any infinite sequence in a finite alphabet can be encoded as the digits of such a number, in an appropriate base. Yet, these properties of Newtonian mechanics seem to be rather weaknesses of this theory than properties that would allow to communicate instantaneously or to store an infinite amount of information in a finite space: Gandy's hypotheses have not be refuted experimentally, for instance by the construction of an instantaneous computer network or by the construction of a hard drive with an unbounded capacity.

More importantly, even if Gandy's hypotheses must be refined, Gandy's proof shows that the physical Church-Turing thesis is a consequence of some hypotheses about nature, that do not refer to notions such that those of language or computability.

## 4 The Physical Church-Turing Thesis and the Galileo Thesis

### 4.1 The Galileo Thesis

**The Effectiveness of Mathematics in Natural Sciences.** The thesis that mathematics are effective in the natural sciences has been formulated by Galileo in 1623: "Philosophy is written in this vast book, which continuously lies open before our eyes (I mean the Universe). But it cannot be understood unless you have first learned the language and recognize the characters in which it is written. It is written in the language of mathematics" [25].

Galileo formulated this thesis, but did not give any explanation why it held. And long after Galileo, the lack of such an explanation was noticed by Einstein according to whom "The eternal mystery of the world is its comprehensibility" [24] or Wigner according to whom "The enormous usefulness of mathematics in the natural sciences is something bordering on the mysterious and that there is no rational explanation for it" [34].

**Insufficient Explanations.** Several explanations of this unreasonable effectiveness of mathematics in the natural sciences have been attempted:

1. God is a mathematician and He wrote the vast book in the language of mathematics.
2. The mathematical concepts are built by abstracting from empirical objects.
3. Scientists select only those phenomena that can be mathematically described.
4. Scientists approximate the phenomena they study, until they can be mathematically described.
5. Our brain is part of nature, hence our concepts are natural objects, thus they are of the same kind as the objects they describe.

Each of these explanation is insufficient. The first reduces the problem to that of why God is a mathematician, which seems even harder to explain. The second is partial: if some mathematical concepts are built by abstracting from natural objects, the concept of ellipse, for instance, has not been built by abstracting from the trajectory of the planets, as it has been introduced some two thousands years before. The third leaves intact the problem of why so many — if not all — phenomena can be described in the language of mathematics. The fourth leaves intact the problem of why phenomena can be approximately — if not accurately — described in the language of mathematics. The fifth presupposes that we understand better a phenomenon from the inside than from the outside, which is not the case in general.

**Perhaps Several Kinds of Effectiveness.** The effectiveness of mathematics in the natural sciences may be of different kinds. And instead of looking for a global explanation of all kinds of effectiveness, we should perhaps look for more local explanations.

For instance, the atomic masses of the chemical elements have a regular structure, as they are the integer multiples of some unit. When this regularity was discovered, there were three exceptions to this rule, because no elements of atomic mass 45, 68, and 70 were known. Yet, as some chemists trusted the structure of the natural numbers better that their observations, they predicted the existence of these three elements, that were later discovered. This is a striking example of the effectiveness of the structure of natural numbers in chemistry.

But, this striking regularity is easily explained by the fact that the mass of the atoms is mostly due to the mass of protons and neutrons that constitute their nucleus and that each nucleus contains a whole number of such particles.

Yet, this explanation sheds light on the effectiveness of the structure of natural numbers to describe the atomic masses of the chemical elements, but it does not shed light on the effectiveness of mathematics in the natural sciences in general, for instance, it does not shed light on the effectiveness of the quadratic functions to describe the free fall in vacuum.

Thus, we shall focus on a particular instance of the general thesis that mathematics are effective in the natural sciences: the fact that physically realized relations can be expressed by a proposition of the language of mathematics.

**Physically Realized Relations.** Let us imagine an experiment where one prepares a physical system by choosing some parameters $a = \langle a_1, ..., a_n \rangle$ and measures some values $b = \langle b_1, ..., b_p \rangle$. Such an experiment defines a relation: the relation $R$ such that $a\ R\ b$ if $b$ is a possible result for the measurements when the chosen parameters are $a$. We say that this relation $R$ is *realized* by this experiment. A relation that is realized by an experiment is said to be *physically realized.*

For instance, if one applies an electrical tension $U$ to a conductor of resistance $R$ and measures the current $I$ passing through this conductor, then $a = \langle U, R \rangle$ and $b = I$ and the realized relation is that relating $\langle U, R \rangle$ and $I$ when $U = RI$.

In the same way, if one lets a body freely falling in vacuum for some time $t$ and measures the distance crossed by this body, the realized relation is that relating $t$ and $x$ when $x = \frac{1}{2}gt^2$.

The relation between $\langle U, R\rangle$ and $I$ can be expressed by a proposition in the language of mathematics $U = RI$. In the same way, the relation between the time and the position of a body freely falling in vacuum can be described by a proposition in the language of mathematics $x = \frac{1}{2}gt^2$. Among the uncountable number of relations between numbers, only a countable number can be defined by a proposition in the language of mathematics, such as $U = RI$ or $x = \frac{1}{2}gt^2$ and all the physically realized relations seem to be in this small set.

As Galileo stressed the *rôle* of the *language* of mathematics, we can call *the Galileo thesis* the thesis that all physically realized relation can be expressed by a proposition in the language of mathematics.

Instead of attempting to explain the general thesis that mathematics are effective in the natural sciences, we shall restrict our investigation to attempt to explain this unreasonable effectiveness of the propositions of the language of mathematics to express physically realized relations.

## 4.2 The Physical Church-Turing Thesis Implies the Galileo Thesis

Our definition of the notions of a machine and of an experiment are identical, both a machine and an experiment is a physical system prepared by choosing some parameters $a = \langle a_1, ..., a_n\rangle$ and on which some values $b = \langle b_1, ..., b_p\rangle$ are measured. Thus the notion of a relation realized by a machine and of a physically realized relation are identical.

The physical Church-Turing thesis states that physically realized relations are computable and the Galileo thesis states that they can be expressed by a proposition in the language of mathematics.

Once we have identified that the Galileo thesis and the physical Church-Turing thesis are theses about the same set of relations, we may remark that the physical Church-Turing thesis implies the Galileo thesis [22]. Indeed, as any program expressing a computable relation is a mathematical expression, all computable relations can be defined by a proposition in the language of mathematics. In fact, computable relations can even be expressed by a proposition in a very small fragment of mathematics: the language of Peano arithmetic. Thus, if the physical Church-Turing thesis holds, then all physically realized relations are computable, hence they can be expressed by a proposition of the language of mathematics, *i.e.* the Galileo thesis holds.

We have seen that

– Gandy's hypotheses imply the physical Church-Turing thesis,
– and the physical Church-Turing thesis implies the Galileo thesis.

Thus, we can deduce that Gandy's hypotheses imply the Galileo thesis and attempt to prove this directly.

Yet, from a historical point of view, it is important to notice the *rôle* of computability theory and of the physical Church-Turing thesis in connecting Gandy's hypotheses to the Galileo thesis.

## 4.3 An Algorithmic Description of the Laws of Nature

A side effect of this explanation of the Galileo thesis is that the laws of nature can be described not only in the language of mathematics, but also in an algorithmic language.

Instead of expressing the law of free fall in vacuum by the proposition $x = \frac{1}{2}gt^2$, we could express it by the algorithm `fun t -> g * t * t / 2`, leading to a second Galilean revolution in the language of natural sciences. In particular, as long as differential equations have computable solutions [29,12,15,16] the language of differential equations can be seen as an algorithmic language.

Yet, this algorithmic description of the laws of nature may have a broader scope than the description of the laws of nature with differential equations. For instance, the transformation of a messenger RNA string to a protein is easily expressed by an algorithm, while it cannot be expressed by a differential equation.

## 4.4 A Property of Nature or of the Theory?

An objection to Galileo's formulation of the Galileo thesis "The Universe [...] is [...] a book [...] written in the language of mathematics" is that it confuses nature with our description of nature: nature itself is neither a book written in the language of mathematics, nor a book written in another language: it is not a book at all. Only our description of nature is a book. Thus, we could imagine that our description of nature is written in the language of mathematics because we have chosen to write it this way, nature having nothing to do with our decision [36].

Yet, nature and our description of nature are not independent: our description must have an experimental adequation of some sort with nature. And as we shall see, when constructing a theory, the scientists have very little freedom when "choosing" the set of physically realized relations.

Let us consider first a particular case where all the realized relations are functional. Then, we show that if two theories differ on the set of realized relations, one of the theories can be, at least in principle, experimentally refuted. Indeed, if the set of the realized relations differ, then there exists a relation $R$ that is realized according to one theory $\mathcal{T}$ but not according to the other theory $\mathcal{T}'$. Let $E$ be the experiment realizing $R$ according to the theory $\mathcal{T}$ and $R'$ be the relation realized by this experiment according to the theory $\mathcal{T}'$. As $R$ is not realized according to $\mathcal{T}'$, the relations $R$ and $R'$ are different. Thus, there exists $a$, $b$, and $b'$, such that $b \neq b'$, $a\ R\ b$, and $a\ R'\ b'$. Then, if we perform the experiment $E$ with the parameters $a$, the measures will either give the result $b$ and refute $\mathcal{T}'$, or $b'$ and refute $\mathcal{T}$, or an other value and refute both theories.

When the realized relations need not be functional, we have a weaker result: either a theory can be refuted, or it predicts, among others, a result that never occurs, whatever the number of times the experiment is repeated. Again, if the set of the realized relations differ, then there exists an experiment that realizes a relation $R$ according to one theory and a relation $R'$, $R' \neq R$, according to the other. Thus, there exists an $a$, such that the set $R_a$ of the $b$ such that $a\ R\ b$ and the set $R'_a$ of the $b$ such of $b$ such that $a\ R'\ b$ differ. As these sets are different, they are not both equal to $R_a \cap R'_a$. Then, if we repeat the experiment with the parameters $a$, either the measures give one result that is not in $R_a \cap R'_a$ and at least one of the theories is refuted, or the measures always give results in $R_a \cap R'_a$ and at least one theory predicts a result that never occurs.

## 5 The Physical Church-Turing Thesis and the Quantum Theory

### 5.1 The Bounded Density of Information in the Quantum Case

The hypothesis of a bounded density of information seems to be inspired by the idea of quantization of the state space, but is in contradiction with quantum theory. Indeed in quantum theory even a system with two distinct states $\mathbf{u}$ and $\mathbf{v}$ has an infinite number of states $\alpha\mathbf{u}+\beta\mathbf{v}$, for $|\alpha|^2+|\beta|^2 = 1$. Thus, in quantum theory, this hypothesis cannot be formulated as the fact that the state space of a cell is finite.

Yet, this does not mean that the amount of possible outcomes of a measurement of some value associated to this system, is itself infinite and the bounded density of information principle can be formulated as the fact that each such measurement may only yield a finite number of possible outcomes.

This requirement amounts to the fact that the state space of a region of space of finite diameter is a finite-dimensional vector space. This constitutes a good alternative to the fact that the amount of possible outcomes, when measuring the system, is finite, as this formulation does not involve the notion of a measurement.

In the classical case, the finiteness of the state space $S$ is used to deduce that any function from $S$ to $S$ is computable. When $S$ is a finite-dimensional vector space, it is not the case that all functions from $S$ to $S$ are computable, but it is the case that all linear maps are computable, as the application of a linear map to a vector boils down to matrix multiplication, *i.e.* to addition and multiplication of scalars. Fortunately, quantum evolutions are always linear maps, hence they are computable. In short, linearity tames infinities, when infinite sets are restricted to finite-dimensional vector spaces.

To define the state space of an non necessarily bounded region of space, we first decompose this region into a finite or infinite number of cells of finite diameter. The state space of each cell is a finite-dimensional vector space. Let $\mathbf{e}_1, \ldots, \mathbf{e}_n$ be a base of this set. The state space of the whole system is the tensor product of all these finite-dimensional spaces. Its vectors are linear combinations of tensor products of elements of $\mathbf{e}_1, \ldots, \mathbf{e}_n$. As all the cells but a finite number of them are in a quiescent state, we can even restrict to finite tensor products.

We get this way a Fock-like space. This space is infinite-dimensional, hence it is not finitely generated with the operations + and ., but as it has a base formed with vectors that are tensor products of elements of $\mathbf{e}_1, \ldots, \mathbf{e}_n$: it is finitely generated with the operations +, . and $\otimes$. In particular computability is stable on such a space [3].

Thus, in the quantum case, the hypothesis of a finite density of information does not rule out infinite-dimensional vector spaces, but it explains where this infinity comes from: from the fact, that in an unbounded physical space, we can assemble an unbounded number of cells.

A difficult question is that of the impact of this hypothesis of a boundedness of information on the choice of the field of scalars we use. If we use the field $\mathbb{C}$, then a scalar contains an infinite quantity of information, for instance any infinite sequence of 0 and 1 may be encoded as the digits of a scalar. Thus, the linear combination $\alpha\mathbf{u} + \beta\mathbf{v}$, also contains an infinite quantity of information. Of course, this information cannot be retrieved in a single measurement, but it can, in principle, be probabilistically retrieved, by repeated measurements on similarly prepared systems.

On the opposite, we might consider that when superposing two base states $\alpha\mathbf{u} + \beta\mathbf{v}$, only a finite number of linear combinations are possible.

A intermediate solution is to assume that the state space of each cell is a finite-dimensional vector space over a finite-degree extension of the field of rationals. Since we are in discrete-time discrete-space quantum theory, such a restriction as little consequences: we have all the scalars that can be generated by a universal set of quantum gates for instance [13], see also [2] for a more in-depth discussion. Nevertheless, in the continuous picture, this kind of assumptions are not without consequences, and these are currently being investigated [17,10].

## 5.2 The Bounded Velocity of Information in the Quantum Case

Like the hypothesis of a finite density of information, that of a finite velocity of propagation of information has to be defined with caution in the quantum case. For instance, according to some interpretations of the Einstein-Podolsky-Rosen paradox, it could be said to contradict quantum theory.

In the Einstein-Podolsky-Rosen paradox, however, no *accessible* information can be communicated faster than the speed of light [9]. Similarly, it can be proved that not more than one bit of accessible information can be stored within a single qubit [27]. Drawing this distinction between the description of the infinite, non-local quantum states and the information that can actually be accessed about them, hints towards the quantum version of these hypotheses.

In the classical case we could assume that the state of a compound system was simply given by the state of each component. In the quantum setting this no longer holds, some correlation information needs to be added. In other words, the state space of two regions is not the Cartesian product of the state space of each region, but its tensor product. Actually, if we stick to state vectors, knowing the state vector, e.g. $(\mathbf{u} \otimes \mathbf{u}) + (\mathbf{v} \otimes \mathbf{v})$, of the compound system, we cannot even assign a state vector to the first system. In order to do so, we need to switch to the density matrix formalism. Each state vector $\mathbf{u}$ is then replaced by the pure density matrix $|\mathbf{u}\rangle\mathbf{u}$ and if $\rho$ is the density matrix of a compound system, then we can assign a density matrix to each subsystem, defined as a partial trace of $\rho$. The partial trace is defined by mapping $A \times B$ to $A$ and extending linearly to $A \otimes B \to A$. Still, knowing the density matrix of each subsystem is again not sufficient to reconstruct the state of the compound system.

In this setting, the most natural way to formalize the *bounded velocity of propagation of information* hypothesis is that given in [6] where it was referred to as *Causality.* Causality says that there exists a constant $T$ such that for any region $A$, any point in time $t$, the density matrix associated to the region $A$ at time $t+T$, $\rho(A, t+T)$ depends only on $\rho(A', t)$, with $A'$ the region of radius 1 around $A$. Actually, this definition is a rephrase of the $C^*$-algebra formulation found in [31], which itself stems from quantum field theoretical approaches to enforcing causality [14]. The difficulty of this axiomatic formalization of the bounded velocity of propagation of information in the quantum case, is that it is quite non-constructive: it is no longer the case that because we know that $\rho(A, t+T)$ is a local function $f_A$ of $\rho(A', t)$, and $\rho(B, t+T)$ is a local function $f_B$ of $\rho(B', t)$, that $\rho(A \cup B, t+T)$ can be reconstructed from $\rho(A' \cup B', t)$ by means of these two functions.

A more constructive approach to formalizing the bounded velocity of propagation of information in the quantum case would be to, instead, state that the global evolution is *localizable* [7,23,32,4], meaning that the global evolution is implementable by local mechanisms, each of them physically acceptable. Here this would say that the global evolution $G$ is in fact quantum circuit of local gates with infinite width but finite depth. But the disadvantage of this approach is that this is a strong supposition to make. Fortunately, in [4,5], the two approaches where shown to be equivalent. Hence we only need to assume causality and we can deduce locality.

With these two hypotheses formalized as explained, it is possible to extend Gandy's theorem to quantum theory [1].

## 6 Conclusion

The physical Church-Turing thesis explains the Galileo thesis, but also suggests an evolution of the language used to describe nature.

It can be proved from more basic principle of physics, but it also questions these principles, putting the emphasis on the principle of a bounded density of information. This principle itself questions our formulation of quantum theory, in particular the choice of a field for the scalars and the origin of the infinite dimension of the vector spaces used as state spaces.

## References

1. Arrighi, P., Dowek, G.: The physical Church-Turing thesis and the principles of quantum theory (to appear)
2. Arrighi, P., Dowek, G.: Operational semantics for formal tensorial calculus. In: Proceedings of QPL, vol. 33, pp. 21–38. Turku Centre for Computer Science General Publication (2004); arXiv pre-print quant-ph/0501150
3. Arrighi, P., Dowek, G.: On the Completeness of Quantum Computation Models. In: Ferreira, F., Löwe, B., Mayordomo, E., Mendes Gomes, L. (eds.) CiE 2010. LNCS, vol. 6158, pp. 21–30. Springer, Heidelberg (2010)
4. Arrighi, P., Nesme, V., Werner, R.: Unitarity plus causality implies localizability. In: QIP 2010, ArXiv preprint: arXiv:0711.3975 (2007)
5. Arrighi, P., Nesme, V., Werner, R.: Unitarity plus causality implies localizability (Full version). Journal of Computer and System Sciences (2010)
6. Arrighi, P., Nesme, V., Werner, R.F.: Quantum Cellular Automata Over Finite, Unbounded Configurations. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 64–75. Springer, Heidelberg (2008)
7. Beckman, D., Gottesman, D., Nielsen, M.A., Preskill, J.: Causal and localizable quantum operations. Phys. Rev. A 64(052309) (2001)
8. Bekenstein, J.D.: Universal upper bound to entropy-to-energy ratio for bounded systems. Phys. Rev. D 23, 287–298 (1981)
9. Bell, J.: On the Einstein Podolsky Rosen paradox. Physics 1, 195 (1964)
10. Benioff, P.: New gauge fields from extension of space time parallel transport of vector spaces to the underlying number systems. Arxiv preprint arXiv:1008.3134 (2010)
11. Boker, U., Dershowitz, N.: The Church-Turing Thesis Over Arbitrary Domains. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science. LNCS, vol. 4800, pp. 199–229. Springer, Heidelberg (2008)
12. Bournez, O., Campagnolo, M.: A survey on continuous time computations. New computational paradigms. Changing Conceptions of What is Computable, 383–423 (2008)
13. Boykin, P.O., Mor, T., Pulver, M., Roychowdhury, V., Vatan, F.: On universal and fault-tolerant quantum computing: A novel basis and a new constructive proof of universality for Shor's basis. In: FOCS 1999: Proceedings of the 40th Annual Symposium on Foundations of Computer Science, p. 486. IEEE Computer Society, Washington, DC (1999)
14. Buchholz, D.: Current trends in axiomatic quantum field theory. Lect. Notes Phys, vol. 558, p. 4364 (2000)
15. Collins, P., Graça, D.S.: Effective computability of solutions of ordinary differential equations — The thousand monkeys approach. In: Brattka, V., Dillhage, R., Grubba, T., Klutsch, A. (eds.) 5th International Conference on Computability and Complexity in Analysis. Electronic Notes Theorerical Computer Science, vol. 221, pp. 103–114 (2008)
16. Collins, P., Graça, D.S.: Effective computability of solutions of differential inclusions — The ten thousand monkeys approach. Journal of Universal Computer Science 15(6), 1162–1185 (2009)
17. Connes, A.: The Witt construction in characteristic one and quantization. Arxiv preprint arXiv:1009.1769 (2010)
18. Copeland, B., Shagrir, O.: Physical Computation: How General are Gandys Principles for Mechanisms? Minds and Machines 17(2), 217–231 (2007)

19. Dershowitz, N., Gurevich, Y.: A natural axiomatization of the computability and proof of Church's thesis. The Bulletin of Symbolic Logic 14(3) (2008)
20. Deutsch, D.: Quantum theory, the Church-Turing principle and the universal quantum computer. In: Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, vol. 400(1985), pp. 97–117 (1818)
21. Dowek, G.: Non determinisitic computation over the real numbers (submited to publication)
22. Dowek, G.: The physical Church thesis as an explanation of the Galileo thesis (submited to publication)
23. Eggeling, T., Schlingemann, D., Werner, R.: Semicausal operations are semilocalizable. EPL (Europhysics Letters) 57, 782 (2002)
24. Einstein, A.: Physics and Reality. Journal of the Franklin Institute 221(3), 349–382 (1936)
25. Galilei, G.: Il Saggiatore (1623)
26. Gandy, R.: Church's thesis and principles for mechanisms. In: The Kleene Symposium. North-Holland (1980)
27. Holevo, A.: Information-theoretical aspects of quantum measurement. Problemy Peredachi Informatsii 9(2), 31–42 (1973)
28. Montague, R.: Towards a general theory of computability. Synthese 12(4), 429–438 (1960)
29. Pour-El, M., Richards, J.: Computability in Analysis and Physics. Springer, Heidelberg (1989)
30. Rabin, M.: Computable algebra, general theory and theory of computable fields. Transactions of the American Mathematical Society 95(2), 341–360 (1960)
31. Schumacher, B., Werner, R.: Reversible quantum cellular automata. ArXiv preprint quant-ph/0405174 (2004)
32. Schumacher, B., Westmoreland, M.D.: Locality and information transfer in quantum operations. Quantum Information Processing 4(1), 13–34 (2005)
33. Weihrauch, K.: Computable analysis: an introduction. Springer, Heidelberg (2000)
34. Wigner, E.: The unreasonable effectiveness of mathematics in the natural sciences. Communications in Pure and Applied Mathematics 13(1) (1960)
35. Wolfram, S.: A new Kind of Science. Wolfram Media (2002)
36. Ziegler, M.: Physically-relativized Church-Turing hypotheses: Physical foundations of computing and complexity theory of computational physics. Applied Mathematics and Computation 215(4), 1431–1447 (2009)

# A Parameterized Complexity Tutorial

Rod Downey*

School of Mathematics, Statistics and Operations Research
Victoria University
P.O. Box 600, Wellington, New Zealand
rod.downey@vuw.ac.nz

**Abstract.** The article was prepared for the LATA 2012 conference where I will be presenting two one and half hour lectures for a short tutorial on parameterized complexity. Much fuller accounts can be found in the books Downey-Fellows [33, 34], Niedermeier [72], Flum-Grohe [49], the two issues of the *Computer Journal* [36] and the recent survey Downey-Thilikos [39].

## 1 Introduction

### 1.1 Preamble

Following early ideas of Fellows and Langston (such as [42–44]) in the early 1990's Mike Fellows and the author initiated a new direction in complexity theory with a series of papers [27–29], and then a number of other papers with talented co-authors, the state of the art (at the time) being given in the monograph [33].

The idea was that to try to devise a complexity theory more attuned to the considerations of practical computation than other kinds of complexity such as the classical theory of NP-completeness.

Now over 20 years have passed. Moreover, the 14 years since the publication of [33] has seen remarkable development in the area. There have appeared new books, many surveys, and more importantly, a whole host of new techniques, applications, and structural complexity. A remarkable number of young talented people have entered the field.

It is also remains true that in many computer science departments, and departments applying complexity the theory only penetrates in a kind of random fashion. Whilst I am sure that in the audience there will be people who now know more about parameterized complexity than I do, my plan is to give a basic tutorial for the others. I hope that they will see some applications for themselves, and perceive the two sided dialog that the theory allows between the hardness theory and the methods of establishing parameterized tractability.

### 1.2 The Idea

The pioneers of complexity theory (such as Edmonds [40]) identified *polynomial time* as a mathematical idealization of what might be considered feasible for the

* Research supported by the Marsden Fund of New Zealand.

running times of algorithms. As the familiar story of complexity theory runs, we soon discover that for many problems, the only algorithm we can find is to try all possibilities. To do so takes $\Omega(2^n)$ for instances of size $n$. In spite of the combined efforts of many algorithm designers, we are unable to show that there is no polynomial time algorithm for a wide class of such problems.

The celebrated work of Cook, Levin and crucially Karp [61] that many of these problems could be shown to be polynomial-time reducible to each other, and to the problem of acceptance for a polynomial time nondeterministic Turing machine. That is, they are NP-complete. This means that we have a *practical* "proof" of hardness. If any of the problems were in polynomial time, all would be; and secondly showing them to be in polynomial time would show that acceptance for a polynomial time nondeterministic Turing machine would be also. The belief is that a nondeterministic Turing machine is such an opaque object, without any obvious algebraic structure, that it seems impossible to see if it has an accepting path without trying all of them.

Parameterized complexity views the story above as a *first foray* into *feasible* computation. However, for practical computation, it seems that we ought to refine the analysis to make it more *fine grained*. Firstly, when we show that something is NP-complete or worse, what we are focusing on is the worst case behaviour. Second, the analysis takes the input as being measured by its *size* alone. You can ask yourself the question: when in real life do we know nothing else about a problem than its *size*? The answer is *never*. For instance, the problem is planar, tree-like, has many parameters bounded, etc. The idea behind parameterized complexity is to try to exploit the *structure* of the input to get some practical tractability. That is, we try to understand what aspect of the problem is to blame for the combinatorial explosion which occurs. If this parameter can be controlled then we would have achieved practical tractability.

Anyone working where they design actual algorithms for real-life problems knows that you should fine tune the algorithm for the situation at hand. Moreover, in applications such as computational biology, structure of things like DNA is *far* from random. The *main idea* of parameterized complexity is to design a paradigm that will address complexity issues in the situation where we know in advance that certain parameters will be likely bounded and this might significantly affect the complexity. Thus in the database example, an algorithm that works very efficiently for small formulas with low logical depth might well be perfectly acceptable in practice.

The idea is not to replace polynomial time as the *underlying* paradigm of feasibility, but to provide a set of tools that refine this concept, allowing some exponential aspect in the running times by allowing us either to use the given structure of the input to arrive at feasibility, or develop some relevant hardness theory to show that the kind of structure is not useful for this approach.

There will be two parts to this theory. First, there have evolved distinct techniques which seem generally applicable to parameterized problems. In this tutorial, I will look at the main players which include *bounded search trees, kernelization,* *other reduction techniques such a crown reductions [3] and Cai's technique [15], iterative compression,* and somewhat less practical but still useful *bounded width metrics*, as well as things like *colour coding methods from logic*, and impractical things like WQO theory.

Hand in hand with this is a hardness theory. I will look at the basic hardness classes such as the W-, A-, M- and other hierarchies. I will also look at how these hierarchies can be exploited to establish practical limitations of the techniques.

## 1.3 Some Definitions

I will now discuss the standard examples which we use for the theory. Mike Fellows and my early work had the three problems Vertex Cover, Dominating Set,Independent Set motivating the theory.

For a graph $G$ a vertex cover is where vertices cover edges: that is $C = \{v_1, \ldots, v_k\}$ is a vertex cover iff for each $e \in E(G)$, there is a $v_i \in C$ such that $v_i \in e$. They should recall that a dominating set is where vertices cover vertices: $D = \{v_1, \ldots, v_k\}$ is a dominating set iff for all $v \in V(G)$, either $v \in D$ or there is an $e \in E(G)$ such that $e = \langle v_i, v \rangle$ for some $v_i \in D$. Finally an independent set is a collection of vertices no pair of which are connected. Of course, these are some of the basic $NP$-complete problems identified by Karp [61].

As in [26], and earlier [33], I will motivate the definitions by looking at a problem in computational biology. Computational biology has been interacting with parameterized complexity from the beginning, and this interaction has continued with throughout, with the work Langston and his group (who have contracts throughout the world to analyze biological data, and use Vertex Cover and other FPT techniques routinely), of Niedermeier and his group, and others. The paper [64] describes a classic application to computational biology.

Suppose we had a conflict graph of some data from this area. Because of the nature of the data we know that it is likely the conflicts are at most about 50 or so, but the data set is large, maybe $10^{12}$ points. We wish to eliminate the conflicts, by identifying those 50 or fewer points. Let's examine the problem depending on whether the identification turns out to be a dominating set problem or a vertex cover problem.

Dominating Set. Essentially the only known algorithm for this problem is to try all possibilities. Since we are looking at subsets of size 50 or less then we will need to examine all $(10^{12})^{50}$ many possibilities. Of course this is completely impossible.

Vertex Cover. There is now an algorithm running in time $O(1.2738^k + kn)$ ([22]) for determining if an $G$ has a vertex cover of size $k$. This and and structurally similar algorithms has been implemented and is practical for $n$ of unlimited practical size and $k$ large. The relevant $k$ has been increasing all the time, evolving from about 400 in [20], to Langston's team [64, 77] who now routinely solve instances on graphs with millions of nodes and vertex covers in the thousands. Moreover, this last work is on actual biological data.

As well as using bounded branching (and parallelization [4]), the method used for this algorithm for VERTEX COVER is called *kernelization* and is based on reduction rules[1], which tend to be easy to implement and perform often much better than anticipated in practice. We will discuss this method in detail soon. The following table from Downey-Fellows [33] exhibits the difference between the parameter $k$ being part of the exponent like DOMINATING SET or as part of the constant like VERTEX COVER. This table compares of a running time of $\Omega(n^k)$ vs $2^k n$.

**Table 1.** The Ratio $\frac{n^{k+1}}{2^k n}$ for Various Values of $n$ and $k$

| | $n = 50$ | $n = 100$ | $n = 150$ |
|---|---|---|---|
| $k = 2$ | 625 | 2,500 | 5,625 |
| $k = 3$ | 15,625 | 125,000 | 421,875 |
| $k = 5$ | 390,625 | 6,250,000 | 31,640,625 |
| $k = 10$ | $1.9 \times 10^{12}$ | $9.8 \times 10^{14}$ | $3.7 \times 10^{16}$ |
| $k = 20$ | $1.8 \times 10^{26}$ | $9.5 \times 10^{31}$ | $2.1 \times 10^{35}$ |

In classical complexity a decision problem is specified by two items of information:

(1) The input to the problem.
(2) The question to be answered.

In parameterized complexity there are three parts of a problem specification:

(1) The input to the problem.
(2) The aspects of the input that constitute the parameter.
(3) The question.

Thus *one* parameterized version of VERTEX COVER is the following:
VERTEX COVER
*Instance:* A graph $G = (V, E)$.
*Parameter:* A positive integer $k$.
*Question:* Does $G$ have a vertex cover of size $\leq k$?

We could, for instance, parameterize the problem in other ways. For example, we could parameterize by some width metric, some other shape of the graph, planarity etc. Any of these would enable us to seek hidden tractability in the problem at hand.

For a formal definition, for simplicity I will stick to the *strongly uniform* definition of being fixed-parameter tractable. There are other definitions of less importance in practice, and I refer the reader to [33] or [49] for more details.

A *parameterized language* is $L \subseteq \Sigma^* \times \Sigma^*$ where we refer to the second coordinate as the *parameter*. It does no harm to think of $L \subseteq \Sigma^* \times \mathbb{N}$. Flum and

[1] There are other FPT methods based around reduction rules such as Leizhen Cai [15] and Khot and Raman [63], which work on certain hereditary properties.

Grohe have an alternative formulation where the second coordinate is a function $\kappa : \Sigma^* \rightarrow \Sigma^*$, but I prefer to keep the second parameter as a string or number.

**Definition 1.** *A parameterized language $L$ is* (strongly) fixed parameter tractable *($FPT$), iff there is a computable function $f$, a constant $c$, and a (deterministic) algorithm $M$ such that for all $x, k$,*

$$\langle x, k\rangle \in L \textit{ iff } M(x, k) \textit{ accepts},$$

*and the running time of $M(x, k)$ is $\leq f(k)|x|^c$.*

It is not difficult to show that the multiplicative constant in the definition can be replaced by an additive one, so that $L \in FPT$ iff $L$ can be accepted by a machine in time $O(|x|^c) + f(k)$ for some computable $f$. In the case of Vertex Cover we have $f(k) = 1.2738^k$, and the $O$ is 2. One nice notation useful here is the $O^*$ notation which ignores the $f(k)$ be it additive or multiplicative and is only concerned with the exponential part. The algorithm would be said to be $O^*(n^1)$. The table on the web site

`http://fpt.wikidot.com/fpt-races`

lists 35 (at the time of writing) basic problems which are fixed parameter tractable with (mostly) practical algorithms, and for which there are current "races" for algorithms with the best run times.

In the following I will briefly mention some of the techniques I will discuss in the tutorial.

## 2 Positive Techniques

### 2.1 Bounded Search Trees

A fundamental source of high running times is *branching* in algorithms. A very crude idea to limit the running time is to keep this branching small and a function of the parameter. For instance, for Vertex Cover, we can do this as follows. Take any edge $e = vw$, and begin a search tree, by having leaves labeled by $v$ and $w$, and for each leaf recursively do this for the graphs gotten by deleting any edge covered by $v$ and $w$ respectively. The depth of this process for a $k$-vertex cover is $k$. Thus, using this idea, we can decide of $G$ has a vertex cover in time $O(2^k|G|)$ using this method.

The running time of this method can often be exploited by by making the tree smaller, or by using some kind of asymptotic combinatorics to constrain the search. For example, if $G$ has no vertex of degree three or more, then $G$ consists of a collection of cycles, and this is pretty trivial to check. Thus we can assume we have vertices of higher degree than 2. For vertex cover of $G$ we must have either $v$ or *all of its neighbours*, so we create children of the root node corresponding to these two possibilities. The first child is labeled with $\{v\}$ and $G - v$, the second with $\{w_1, w_2, \ldots, w_p\}$, the neighbours of $v$, and $G - \{w_1, w_2, \ldots, w_p\}$. In the case of the first child, we are still looking for a size $k-1$ vertex cover, but in the case of the second child we need only look for a vertex cover of size $k-p$, where $p$ is at least 3. Thus, the bound on the size of the search tree is now somewhat smaller than $2^k$. It can be shown that this algorithm runs in time $O(5^{k\backslash 4} \cdot n)$, and in typical graphs, there are lots of vertices of higher degree than 3, and hence this works even faster.

More involved rules exploring the *local structure* of neighbourhoods in graphs, result in the algorithm of Chen et. al. [22]) with running time $O^*(1.2738^k)$ for the branching.

There are a number of problems for which this technique is the only method, or at least the best method, for parameterized algorithms. The method has been particularly successful in computational biology with problems like the Closest String problem [55] and Maximum Agreement Forest problem [58].

In passing I remark that this method is inherently parallelizable and as we see is often used in conjunction with other techniques. The method for Vertex Cover can be found discussed in [4].

## 2.2 Kernelization

The idea is that if we make the problem *smaller* then the search will be quicker. This is a *data reduction* or *pre-processing* idea, and is the heart of many heuristics.

Whilst there are variations of the idea below, the simplest version of kernelization is the following.

**Definition 2 (Kernelization)**
*Let $L \subseteq \Sigma^* \times \Sigma^*$ be a parameterized language. A reduction to a problem kernel, or kernelization, comprises replacing an instance $(I,k)$ by a reduced instance $(I',k')$, called a problem kernel, such that*

*(i) $k' \leq k$,*
*(ii) $|I'| \leq g(k)$, for some function $g$ depending only on $k$, and*
*(iii) $(I,k) \in L$ if and only if $(I',k') \in L$.*

*The reduction from $(I,k)$ to $(I',k')$ must be computable in time polynomial in $|I| + |k|$.*

There are other notions, where the kernel may be another problem (often "annotated") or the parameter might increase, but, crucially, the size of the kernel depends *only on $k$*.

Here are some natural reduction rules for a kernel for Vertex Cover.
Reduction Rule VC1:
Remove all isolated vertices.

Reduction Rule VC2:
For any degree one vertex $v$, add its single neighbour $u$ to the solution set and remove $u$ and all of its incident edges from the graph.

These rules are obvious. Sam Buss (see [33]) originally observed that, for a simple graph $G$, any vertex of degree greater than $k$ must belong to every $k$-element vertex cover of $G$ (otherwise all the neighbours of the vertex must be included, and there are more than $k$ of these).
This leads to our last reduction rule.
Reduction Rule VC3:
If there is a vertex $v$ of degree at least $k+1$, add $v$ to the solution set and remove $v$ and all of its neighbours.

After exhaustively applying these rules, we get to a graph $(G', k')$, where no vertex in the reduced graph has degree greater than $k' \leq k$, or less than two. Then simple combinatorics shows that if such a reduced graph has a size $k$ vertex cover, its must have size $\leq k^2$. This is the size $k^2$ kernelization.

Now we can apply the bounded depth search tree rule to this reduced graph, and get an algorithm for vertex cover running in time $O(1.2738^k)k^2$. As observed by Langston and his team in problems in sequence analysis, and articulated by Niedermeier and Rossmanith [73] better running times can be obtained by *interleaving* depth-bounded search trees and kernelization. That is, first kernelize, begin a bounded search tree, and the *re-kernelize* the children, and repeat. This really does make a difference. In [71] the 3-Hitting Set problem is given as an example. An instance $(I, k)$ of this problem can be reduced to a kernel of size $k^3$ in time $O(|I|)$, and the problem can be solved by employing a search tree of size $2.27^k$. Compare a running time of $O(2.27^k \cdot k^3 + |I|)$ (without interleaving) with a running time of $O(2.27^k + |I|)$ (with interleaving).

In actual implementations there are other considerations such as load sharing amongst processors and the like. We refer to the articles in the Computer Journal special issue concerning practical FPT.

The best kernelization for Vertex Cover is due to Nemhauser and Trotter [70] and has size $2k$. It is based on matching and we will look at this technique in the tutorial. The technique is quite powerful.

There are many other reduction techniques based on locan graph structure like *crown reductions* ([3]), and then the generalization to the less practical *protrusions.* See Downey and Thilikos [39].

## 2.3 Iterative Compression

This technique was first introduced in a paper by Reed, Smith and Vetta in 2004 [75] and more or less re-discovered by Karp [62]. Although currently only a small number of results are known, it seems to be applicable to a range of parameterized minimization problems, where the parameter is the size of the solution set. Most of the currently known iterative compression algorithms solve *feedback set problems* in graphs, problems where the task is to destroy certain cycles in the graph by deleting at most $k$ vertices or edges. In particular, the k-Graph Bipartisation problem, where the task is to find a set of at most $k$ vertices whose deletion destroys all odd-length cycles, has been shown to be FPT by means of iterative compression [75]. This had been a long-standing open problem in parameterized complexity theory.

**Definition 3 (Compression Routine)**
*A* compression routine *is an algorithm that, given a problem instance $I$ and a solution of size $k$, either calculates a smaller solution or proves that the given solution is of minimum size.*

Here is a compression routine for Vertex Cover. Begin with $(G = (V, E), k)$, we build the graph $G$ vertex by vertex. We start with an initial set of vertices $V' = \emptyset$ and an initial solution $C = \emptyset$. At each step, we add a new vertex $v$ to both $V'$ and $C$, $V' \leftarrow V' \cup \{v\}$, $C \leftarrow C \cup \{v\}$. We then call the compression routine on the pair $(G[V'], C)$, where $G[V']$ is the subgraph induced by $V'$ in $G$, to obtain a new solution $C'$. If $|C'| > k$ then we output NO, otherwise we set $C \leftarrow C'$.

If we successfully complete the $n$th step where $V' = V$, we output $C$ with $|C| \leq k$. Note that $C$ will be an optimal solution for $G$.

The compression routine takes a graph $G$ and a vertex cover $C$ for $G$ and returns a smaller vertex cover for $G$ if there is one, otherwise, it returns $C$ unchanged. Each time the compression routine is used it is provided with an intermediate solution of size at most $k + 1$.

The implementation of the compression routine proceeds as follows. We consider a smaller vertex cover $C'$ as a *modification* of the larger vertex cover $C$. This modification retains some vertices $Y \subseteq C$ while the other vertices $S = C \setminus Y$ are replaced with $|S| - 1$ new vertices from $V \setminus C$. The idea is to try by brute force all $2^{|C|}$ partitions of $C$ into such sets $Y$ and $S$. For each such partition, the vertices from $Y$ along with all of their adjacent edges are deleted. In the resulting instance $G' = G[V \setminus Y]$, it remains to find an optimal vertex cover that is disjoint from $S$. Since we have decided to take no vertex from $S$ into the vertex cover, we have to take that endpoint of each edge that is not in $S$. At least one endpoint of each edge in $G'$ is in $S$, since $S$ is a vertex cover for $G'$. If both endpoints of some edge in $G'$ are in $S$, then this choice of $S$ cannot lead to a vertex cover $C'$ with $S \cap C' = \emptyset$. We can quickly find an optimal vertex cover for $G'$ that is disjoint from $S$ by taking every vertex that is not in $S$ and has degree greater than zero. Together with $Y$, this gives a new vertex cover $C'$ for $G$. For each choice of $Y$ and $S$, this can be done in time $O(m)$, leading to $O(2^{|C|}m) = O(2^k m)$ time overall for one call of the compression routine. With at most $n$ iterations of the compression algorithm, we get an algorithm for k-Vertex Cover running in time $O(2^k mn)$.

The parametric tractability of the method stems from the fact that each intermediate solution considered has size bounded by some $k' = f(k)$, where $k$ is the parameter value for the original problem.

## 2.4 Other Techniques

Other methods we will discuss will include *colour coding* (Alon, Yuster and Zwick [7]), *treewidth* (e.g. [13]), and more exotic methods. Given time, we will look at constructivization of the methods via what is called *bidimensionality theory*.

For colour-coding, it is most easily understood with a simple example. The method remains not-quite-practical as the numbers involved are large, but not astronomical. We will apply the problem to $k$-Path which seeks to find a (simple) path of $k$ vertices in $G$. What we do is to *randomly* colour the whole graph with $k$ colors, and look for a *colourful* solution, namely one with $k$ vertices of one of each color.

The two keys to this idea are

(i) we can check for colourful paths quickly.
(ii) if there is a simple path then the probability that it will have $k$ colors for a random coloring is $\frac{k!}{k^k}$ which is bounded by $e^{-k}$.

Then, given (i) and (ii), we only need repeat process enough to fast probabilistic algorithm. We prove (i) by using dynamic programming: simply add a vertex $v_0$ with color 0, connect to those of color 1, then generate the colorful paths of length $i$ starting from $v_0$ inductively, rather like Dijkstra's algorithm, the running time being $O(k2^k|E|)$.

**Theorem 4 (Alon, Yuster and Zwick [7]).** $k$-Path *can be solved in expected time* $2^{O(k)}|E|$.

Alon, Yuster and Zwick demonstrated that this technique could be applied to a number of problems of the form asking "is $G'$ a subgraph of $G$?" The desired FPT algorithm can now be obtained by a process of derandomization. The desired FPT algorithm can now be obtained by a process of derandomization. A *k-perfect family of hash functions* is a family $\mathcal{F}$ of functions (colorings) taking $[n] = \{1, \dots n\}$ onto $[k]$, such that for all $S \subseteq [n]$ of size $k$ there is a $f \in \mathcal{F}$ whose restriction to is bijective (colourful). It is known that $k$-perfect families of $2^{O(k)} \log n$ linear time hash functions. This gives a deterministic $2^{O(k)}|E| \log |V|$ algorithm for $k$-Path. More such applications can be found in Downey and Fellows [33], and Niedermeier [71, 72]. The $O(k)$ in the exponent is where the problem lies, and the derandomization method at present seems far from practical. It should be noted that the method does not show that $k$-Clique is in randomized FPT because (i) above *fails*. We also remaark that recent work ([11]) has shown, assuming a reasonable complexity assumption (namely that the polynomial time hierarchy does not collapse to two or fewer levels), there is no polynomial size kernel for $k$-Path. Thus, this is a classic problem which is in $O^*(2^{O(k)})$-FTP yet has no polynomial kernel under widely held complexity assumptions.

Another use of dynamic programming concernes optimization problem on graphs with inductively defined internal structure. One example of this is treewidth, but there are other width metrics like *cliquewidth, branchwidth,* and others.

**Definition 5 (Robertson and Seymour [76])**

*(a) A* tree-decomposition *of a graph $G = (V, E)$ is a tree $\mathcal{T}$ together with a collection of subsets $T_x$ (called* bags*) of $V$ labeled by the vertices $x$ of $\mathcal{T}$ such that $\cup_{x \in \mathcal{T}} T_x = V$ and (i) and (ii) below hold:*
*(i) For every edge $uv$ of $G$ there is some $x$ such that $\{u, v\} \subseteq T_x$.*
*(ii) (Interpolation Property) If $y$ is a vertex on the unique path in $\mathcal{T}$ from $x$ to $z$ then $T_x \cap T_z \subseteq T_y$.*

*(b) The* width *of a tree decomposition is the maximum value of* $|T_x| - 1$ *taken over all the vertices* $x$ *of the tree* $\mathcal{T}$ *of the decomposition.*
*(c) The treewidth of a graph* $G$ *is the minimum treewidth of all tree decompositions of* $G$.

The point of the notion is that it is a measure of how treelike the graph is. One can similarly define *path decomposition* where $\mathcal{T}$ must be a path. A tree decomposition is a road map as to how to build the graph. Knowing a tree or path decomposition of a graph allows for dynamic programming since what is important is the "information flow" across a relatively narrow cut in the graph.

Figure 1 gives an example of a tree decomposition of width 2.

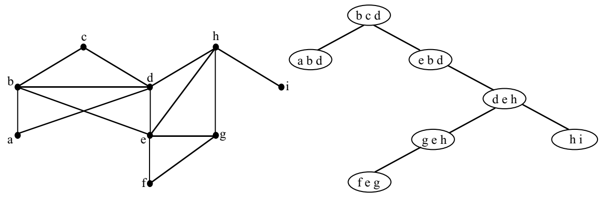


**Fig. 1.** Example of Tree Decomposition of Width 2

The point is that suppose you were seeking a dominating set in the graph of the example. Starting at the bag $\{f, e, h\}$, there are $2^3 = 8$ possible beginnings of a dominating set: $\emptyset, \{f\}, \{e\}, \{g\}, \{e, f\}, \{f, g\}, \{e, g\}, \{e, f, g\}$, that is, the subsets of $\{e, f, g\}$. These correspond to dominating sets of sizes, respectively 0,1,1,1,2,2,2,3. Now, if we move to the next bag, $\{g, e, h\}$, up the path we lose the vertex $f$ and it will never appear again (because of the interpolation property). So that to figure out the possible dominating sets, we only need to remember to point at what we have done before, and the information flow across the boundary of the cut. Thus if we choose $\emptyset$ from the first bag, we must choose one of $\{e\}$, $\{g\}$, or $\{e, g\}$ from bag $\{g, e, h\}$, but choosing $\{e\}$ from bag $\{f, e, h\}$, would allow for $\emptyset$, for example, from $\{g, e, h\}$. Join bags are handled in a similar manner. Then one has to simply climb up the decomposition to the top and read off the minimum dominating set. The crucial thing is that the exponential part of running time of this process depends on the bag size, and the complexity of the definition of the property of the problem being analysed.

This vague statement can be made more formal using Courcelle's Theorem ([24]) which shows that any problem in "monadic second order counting logic" is linear time FPT for graphs on a fixed treewidth, and later work shows a similar statement for graphs of bounded "local treewidth" for first order properties. The key problem with these methods is that it can be hard to actually find the tree decomposition, though the problem is FPT by Bodlaender [9]. Unfortunately, this algorithm is $O(2^{35t^3}|G|)$ for a fixed treewidth $t$, which is far from practical. It also seems unlikely that the problem of treewidth $k$ has a polynomial kernel as argued in [11]. Moreover, assuming reasonable assumptions, it has been shown that the big towers of 2's in the constants for the running times obtained from Courcelle's Theorem cannot be gotten rid of. (Frick and Grohe [52], also Flum and Grohe [48])

## 3 Parametric Intractability

The two key ingredients of a hardness theory are (i) a notion of hardness and (ii) a notion of "problem $A$ could be solved efficiently if we could solve problem $B$"; that is a notion of red reducibility.

In the classic theory of NP completeness (i) is achieved by the following:
Nondeterministic Turing Machine Acceptance
*Input:* A nondeterministic Turing Machine $M$ and a number $e$.
*Question:* Does $M$ have an accepting computation in $\leq |M|^e$ steps?

The Cook-Levin argument is that a Turing machine is such an opaque object that it seems that there would be no way to decide if $M$ accepts, without essentially trying the paths. If we accept this thesis, then we probably should accept that the following problem is not $O(|M|^c)$ for any fixed $c$ and is probably $\Omega(|M|^k)$ since again our intuition would be that all paths would need to be tried:
Short Nondeterministic Turing Machine Acceptance
*Input:* A nondeterministic Turing Machine $M$
*Parameter:* A number $k$.
*Question:* Does $M$ have an accepting computation in $\leq k$ steps?

Assuming the rhetoric of the Cook-Levin Theorem, it seems hard to believe that Short Nondeterministic Turing Machine Acceptance could be in FPT, for example, solved in $O(|M|^3)$ for any path length $k$. Indeed, Short Nondeterministic Turing Machine Acceptance not in FPT is closely related to the statement $n$-variable 3Sat not being solvable in subexponential time.

Thus to show Dominating Set is likely not FPT could be achieved by showing that *if we could solve it in time $O(n^c)$ by for each fixed $k$, then we could have a $O(n^c)$ for* Short Nondeterministic Turing Machine Acceptance. Our principal working definition for parameterized reductions is the following.

**Definition 6.** *Let $L, L'$ be two parameterized languages. We say that $L \leq_{fpt} L'$ iff there is an algorithm $M$, a computable function $f$ and a constant $c$, such that*

$$M : \langle G, k\rangle \mapsto \langle G', k'\rangle,$$

*so that*
*(i) $M(\langle G, k\rangle)$ runs in time $\leq g(k)|G|^c$.*
*(ii) $k' \leq f(k)$.*
*(iii) $\langle G, k\rangle \in L$ iff $\langle G', k'\rangle \in L'$.*

We give now a simple example of a parametric reduction from $k$-Clique to $k$-Independent Set, where the standard reduction is parametric (a situation not common). The following is a consequence of Cai, Chen, Downey and Fellows [16], and Downey and Fellows [31].

**Theorem 7.** *The following are hard for* Short Nondeterministic Turing Machine Acceptance*:* Independent Set*,* Dominating Set*.*

In the tutorial, I plan to discuss the proof of Theorem 7.

A classic non-example of a parameterized reduction is the classical reduction of Sat to 3Sat. Early on Downey and Fellows conjectured that there is *no* parametric reduction of $k$-Dominating Set to $k$-Independent Set, and we see that the refined reduction of parameterized complexity theory give rise to several hierarchies based on logical depth. In particular, the $W$-hierarchy is based on weighted versions of satsifiabillity (i.e., the number of literals set to true) for a formula in $PoSoP...$ form or depth $t$. I will discuss the $W$-hierarchy below.

$$W[1] \subseteq W[2] \subseteq W[3] \ldots W[SAT] \subseteq W[P] \subseteq XP.$$

Here the other classes $W[P]$, the weighted circuit satisfiability class, and $XP$ which has as its defining problem the class whose $k$-th slice is complete for $DTIME(n^k)$, this being provably distinct from FPT and akin to exponential time. There are many, many problems hard for $W[1]$ and complete at many levels of this hierarchy. There are also other hierarchies based on other ideas of logical depth. One important hierarchy of this kind was found by Flum and Grohe is the $\mathsf{A}$-hierarchy which is also based on a logical alternation. For a class $\Phi$ of formulae, we can define the following parameterized problem.

$p$-MC($\Phi$)
*Instance:* A structure $\mathcal{A}$ and a formula $\varphi \in \Phi$.
*Parameter:* $|\varphi|$.
*Question:* Decide if $\phi(\mathcal{A}) \neq \emptyset$, where this denotes the evaluation of $\phi$ in $\mathcal{A}$.

Flum and Grohe define

$$\mathrm{A}[t] = [p\text{-MC}(\Sigma_t)]^{\mathrm{FPT}}.$$

For instance, for $k \geq 1$, $k$-Clique can be defined by

$$\mathrm{clique}_k = \exists x_1, \ldots x_k ( \bigwedge_{1 \leq i < j \leq k} x_i \neq x_j \wedge \bigwedge_{1 \leq i < j \leq k} E x_i x_j)$$

in the language of graphs, and the interpretation of the formula in a graph $G$ would be that $G$ has a clique of size $k$. Thus the mapping $(G, k) \mapsto (G, \mathrm{clique}_k)$ is a fixed parameter reduction showing that parameterized Clique is in $\mathsf{A}[1]$. Flum and Grohe populate various levels of the $A$-hierarchy and show the following.

**Theorem 8 (Flum and Grohe [46, 48]).** *The following hold:*

*(i)* $\mathsf{A}[1] = \mathsf{W}[1]$.
*(ii)* $\mathsf{A}[t] \subseteq \mathsf{W}[t]$.

Clearly $\mathsf{A}[t] \subseteq \mathsf{XP}$, but no other containment with respect to other classes of the $\mathsf{W}$-hierarchy is known. There are also hierarchies like the $M$-hierarchy based around parameterizing the problem size.

As discussed in [26], this hierarchy is related to PTAS's; the connection of parameterized complexity with PTAS's going back to Bazgan [8]. The reader may note that parameterized complexity is addressing intractability *within polynomial time*. In this vein, the parameterized framework can be used to demonstrate that many classical problems that admit a PTAS do not, in fact, admit any PTAS with a practical running time, unless W[1] =FPT. The idea here is that if a PTAS has a running time such as $O(n^{\frac{1}{\epsilon}})$, where $\epsilon$ is the error ratio, then the PTAS is unlikely to be useful. For example if $\epsilon = 0.1$ then the running time is already $n$ to the 10th power for an error of 10%. The idea is then to use $\frac{1}{\epsilon}$ as a parameter and show $W[1]$-hardness, say, for that parameterization. Then the problem cannot have a PTAS without $\frac{1}{\epsilon}$ (or worse) in the exponent. Thus it will be unlikely that any feasible PTAS can be found. More precise measurements can be obtained using the $M$-hierarchy, as we now see.

It was an insight of Cai and Juedes that tight lower bounds for approximation and parameterized complexity are intimately related; and indeed, are also related to classical questions about $NP$ and subexponential time. In particular, Cai et. al. [17] who showed that the method of using planar formulae tends to give PTAS's that are never practical. The exact calibration of PTAS's and parameterized complexity comes through yet another hierarchy called the M-hierarchy.

The base level of the hierarchy is the problem M[1] defined by the core problem below.

*Instance:* A CNF circuit $C$ (or, equivalently, a CNF formula) of size $k \log n$.
*Parameter:* A positive integer $k$.
*Question:* Is $C$ satisfiable?

That is, we are parameterizing the *size* of the problem rather than some aspect of the problem. The idea naturally extends to higher levels for that, for example, $M[2]$ would be a product of sums of product formula of size $k \log n$ and we are asking whether it is satisfiable. The basic result is thatFPT⊆ M[1] *subseteq*W[1]. The hypothesis FPT≠M[1] is *equivalent to* a classical conjecture called the *exponential time hypothesis*, ETH. This hypothesis is due to Impagliazzo, Paturi and Zane [60] and asserts that $n$-variable 3Sat cannot be solved in "subexponential time", DTime $(2^{o(n)})$. This conjecture accords with the intuition that not only does $P \neq NP$ but actually $NP$ is really at exponential level.

One example of a lower bound was the original paper of Cai and Juedes [18, 19] who proved the following definitive result.

**Theorem 9 (Cai and Juedes [18, 19])**

$k$-Planar Vertex Cover,
$k$-Planar Independent Set,
$k$-Planar Dominating Set, *and*
$k$-Planar Red/Blue Dominating Set

*cannot be in* $O^*(2^{o(\sqrt{k})})$*-FPT unless* FPT=M[1] *(or, equivalently, unless ETH fails).*

We remark that Theorem 9 is optimal as all the problems above have been classified as $O^*(2^{O(\sqrt{k})})$ (see e.g. Downey and Thilikos [39])

## 3.1 XP-Optimality

There is a new programme akin to the above establishing tight lower bounds on parameterized problems, assuming various non-collapses of the parameterized hierarchies. A powerful example of this is what is called XP optimality. This new programme regards the classes like W[1] as artifacts of the basic problem of proving hardness under reasonable assumptions, and strikes at membership ofXP. We illustrate this via Independent Set and Dominating Set which certainly are in XP. But what's the best exponent we can hope for for slice $k$?

**Theorem 10 (Chen et. al [21]).** *The following hold:*

*(i)* Independent Set *cannot be solved in time* $n^{o(k)}$ *unless* FPT=M[1].
*(ii)* Dominating Set *cannot be solved in time* $n^{o(k)}$ *unless* FPT=M[2].

## 4 Other Things

The other things I plan to discuss is how to use parameterized complexity to show that various methods such as logical metatheorems (such as Courcelle's Theorem and the Frick Grohe [51] theorem on local treewidth) which make very large constants, cannot be improved assuming the $W$-hierarchy does nor collapse. Also I will look at recent work on lower bounds for kernelization such as [11, 14, 23, 50, 59]. This is quite exciting work which allows us to demonstrate that the kernelization technique cannot be used to give polynomial sized kernels.

I won't have time to discuss the use of parameterized complexity to rule out various techniques in other areas such as Aleknovich and Razborov [6].

## 5 Left Out

Clearly in 2 lectures I won't have time to deal with things in a lot of depth. I have alose left out a lot. For example, there is work on parameterized counting (McCartin [67] and Flum and Grohe [47]) where we count the number of paths of length $k$ to define, for instance, $\#W[1]$. One nice theorem here is the following.

**Theorem 11 (Flum and Grohe [47]).** *Counting the number of cycles of size $k$ in a bipartite graph is* #W[1]*-complete.*

This result can be viewed as a parameterized analog of Valiant's theorem on the permanent. Another area is parameterized randomization, such as Downey, Fellows and Regan [38], and Müller [68, 69], but here problems remain. Parameterized approximation looks at questions like: Is it possible to have an FPT algorithm which, on parameter $k$, either outputs a size $2k$ dominating set for $G$, or says no dominating set of size $k$? Such algorithms famously exist for Bin

Packing and don't exist for most natural $W[P]$ complete problems. Here we refer to Downey, Fellows, McCartin and Rosamond [37] and Eickmeyer, Grohe and Grüber [41] for more details. We have left out discussions of parameterizing above guaranteed values such as Mahajan and Raman [65], plus discussions of the breadth of applications. For the last, we can only point at the relevant books, and the *Computer Journal* issues [36].

There are many other important topics such as implementations, connections with exact algorithms, connections with classical complexity and the like. Time and space limitations preclude this material being included.

## References

1. Abrahamson, K., Downey, R., Fellows, M.: Fixed-Parameter Intractability II. In: Enjalbert, P., Wagner, K.W., Finkel, A. (eds.) STACS 1993. LNCS, vol. 665, pp. 374–385. Springer, Heidelberg (1993)
2. Abrahamson, K., Downey, R., Fellows, M.: Fixed-parameter tractability and completeness IV: On completeness for $W[P]$ and PSPACE analogs. Annals of Pure and Applied Logic 73, 235–276 (1995), `http://www.mrfellows.net/papers/J31-fpt4-95.pdf`
3. Abu-Khzam, F.N., Fellows, M., Langston, M.A., Suters, W.H.: Crown structures for vertex cover kernelization. Theory Comput. Syst. 41(3), 411–430 (2007)
4. Abu-Khzam, F.N., Langston, M.A., Shanbhag, P., Symons, C.T.: Scalable parallel algorithms for FPT problems. Algorithmica 45(3), 269–284 (2006)
5. Adler, I., Grohe, M., Kreutzer, S.: Computing excluded minors. In: Proceedings of the of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, pp. 641–650. SIAM (2008)
6. Alekhnovich, M., Razborov, A.A.: Resolution is not automatizable unless W[P] is tractable. In: FOCS, pp. 210–219. IEEE Computer Society (2001)
7. Alon, N., Yuster, R., Zwick, U.: Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs. In: Proc. Symp. Theory of Computing (STOC), pp. 326–335. ACM (1994)
8. Bazgan, C.: Schémas d'approximation et complexité paramétrée, Rapport de stage de DEA d'Informatique à Orsay. Tech. rep., Informatique d'Orsay (1995)
9. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25, 1305–1317 (1996)
10. Bodlaender, H.L., Downey, R., Fellows, M., Hallett, M.T., Wareham, H.T.: Parameterized complexity analysis in computational biology. Computer Applications in the Biosciences 11, 49–57 (1995)
11. Bodlaender, H.L., Downey, R., Fellows, M., Hermelin, D.: On problems without polynomial kernels. Journal of Computing and System Sciences 75(8), 423–434 (2009)
12. Bodlaender, H.L., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M.: (Meta) kernelization. In: FOCS, pp. 629–638. IEEE Computer Society (2009)
13. Bodlaender, H.L., Koster, A.M.C.A.: Combinatorial optimization on graphs of bounded treewidth. The Computer Journal 51, 255–269 (2008)
14. Bodlaender, H.L., Thomasse, S., Yeo, A.: Analysis of data reduction, transformations give evidence for non-existence of polynomial kernels. Tech. Rep. UU-CS-2008-030, Utrecht University (2008)

15. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Information Processing Letters 58(4), 171–176 (1996)
16. Cai, L., Chen, J., Downey, R., Fellows, M.: On the parameterized complexity of short computation and factorization. Arch. for Math. Logic 36, 321–337 (1997)
17. Cai, L., Fellows, M., Juedes, D.W., Rosamond, F.A.: The complexity of polynomial-time approximation. Theoretical Computer Science 41, 459–477 (2007)
18. Cai, L., Juedes, D.W.: Subexponential Parameterized Algorithms Collapse the W-Hierarchy. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 273–284. Springer, Heidelberg (2001)
19. Cai, L., Juedes, D.W.: On the existence of subexponential parameterized algorithms. Journal of Computing and Systems Science 67, 789–807 (2003)
20. Cheetham, J., Dehne, F.K.H.A., Rau-Chaplin, A., Stege, U., Taillon, P.J.: Solving large FPT problems on coarse-grained parallel machines. Journal of Computer and Systems Sciences 67(4), 691–706 (2003)
21. Chen, J., Chor, B., Fellows, M., Huang, X., Juedes, D.W., Kanj, I.A., Xia, G.: Tight lower bounds for certain parameterized NP-hard problems. Information and Computation 201, 216–231 (2005)
22. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theor. Comput. Sci. 411, 3736–3756 (2010)
23. Chen, Y., Flum, J., Müller, M.: Lower bounds for kernelizations and other preprocessing procedures. Theory Comput. Syst. 48(4), 803–839 (2011)
24. Courcelle, B.: Recognizability and second-order definability for sets of finite graphs. Tech. Rep. I-8634, Universite de Bordeaux (1987)
25. Demaine, E.D., Hajiaghayi, M.: The bidimensionality theory and its algorithmic applications. Comput. J. 51(3), 292–302 (2008)
26. Downey, R.: Parameterized complexity for the skeptic. In: Proc. 18th IEEE Annual Conference on Computational Complexity, pp. 147–169 (2003)
27. Downey, R., Fellows, M.: Fixed parameter tractability and completeness. In: Ambos-Spies, K., Homer, S., Schöning, U. (eds.) Complexity Theory: Current Research, pp. 161–187. Cambridge University Press (1992); congressus Numerantium, 87
28. Downey, R., Fellows, M.: Fixed-parameter intractability. In: Structure in Complexity Theory Conference, pp. 36–49 (1992)
29. Downey, R., Fellows, M.: Fixed-parameter tractability and completeness III: Some structural aspects of the $W$-hierarchy. In: Ambos-Spies, K., Homer, S., Schöning, U. (eds.) Complexity Theory: Current Research - Proceedings of the 1992 Dagstuhl Workshop on Structural Complexity, pp. 166–191. Cambridge University Press, Cambridge (1993)
30. Downey, R., Fellows, M.: Fixed-parameter tractability and completeness I: Basic theory. SIAM Journal of Computing 24, 873–921 (1995), `http://mrfellows.net/papers/J29-completeness1-95.ps`
31. Downey, R., Fellows, M.: Fixed-parameter tractability and completeness II: Completeness for W[1]. Theoretical Computer Science A 141, 109–131 (1995), `http://mrfellows.net/papers/J30-CompletenessII-95.ps`
32. Downey, R., Fellows, M.: Parameterized computational feasibility. In: Clote, P., Remmel, J. (eds.) Proceedings of the Second Cornell Workshop on Feasible Mathematics. Feasible Mathematics II, pp. 219–244. Birkhäuser, Boston (1995), `http://mrfellows.net/papers/C17-feasibility95.ps`
33. Downey, R., Fellows, M.: Parameterized Complexity, 530 pages. Springer, Heidelberg (1998)

34. Downey, R., Fellows, M.: Fundamentals of Parameterized Complexity. Springer, Heidelberg (in preparation, 2012)
35. Downey, R., Fellows, M., Kapron, B., Hallett, M., Wareham, H.T.: The Parameterized Complexity of Some Problems in Logic and Linguistics. In: Matiyasevich, Y.V., Nerode, A. (eds.) LFCS 1994. LNCS, vol. 813, pp. 89–100. Springer, Heidelberg (1994)
36. Downey, R., Fellows, M., Langston, M.: Two Special Issue of The Computer Journal Special Issue on Parameterized Complexity. The Computer Journal 51(1&3) (2008), `http://mrfellows.netpapers/J71-CJforward08.pdf`
37. Downey, R., Fellows, M., McCartin, C., Rosamond, F.: Parameterized approximation of dominating set problems. Information Processing Letters 109(1), 68–70 (2008)
38. Downey, R., Fellows, M., Regan, K.W.: Parameterized circuit complexity and the $W$ hierarchy. Theoretical Computer Science A 191(1-2), 97–115 (1998), `http://mrfellows.net/papers/DFR98_CircuitComplexity.pdf`
39. Downey, R., Thilikos, D.M.: Confronting intractability via parameters. Computer Science Review 5, 279–317 (2011)
40. Edmonds, J.: Paths, trees, and flowers. Canad. J. Math. 17, 449–467 (1965), `http://www.cs.berkeley.edu/~christos/classics/edmonds.ps`
41. Eickmeyer, K., Grohe, M., Grüber, M.: Approximation of natural W[P]-Complete minimisation problems is hard. In: Proceedings of the 2008 IEEE 23rd Annual Conference on Computational Complexity, CCC 2008, pp. 8–18. IEEE Computer Society, Washington, DC (2008), `http://dx.doi.org/10.1109/CCC.2008.24`
42. Fellows, M., Langston, M.A.: Nonconstructive advances in polynomial-time complexity. Information Processing Letters 26(3), 155–162 (1987)
43. Fellows, M., Langston, M.A.: Nonconstructive tools for proving polynomial-time decidability. Journal of the Association for Computing Machinery 35(3), 727–739 (1988)
44. Fellows, M., Langston, M.A.: An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations. In: Proceedings of the IEEE Symposium on the Foundations of Computer Science (FOCS), pp. 520–525. IEEE Computer Society (1989)
45. de Fluiter, B.: Algorithms for graphs of small treewidth. Ph.D. thesis, Utrecht University (1970)
46. Flum, J., Grohe, M.: Describing Parameterized Complexity Classes. In: Alt, H., Ferreira, A. (eds.) STACS 2002. LNCS, vol. 2285, pp. 359–371. Springer, Heidelberg (2002)
47. Flum, J., Grohe, M.: The parameterized complexity of counting problems. In: Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS 2002), pp. 538–547. IEEE Computer Society (2002)
48. Flum, J., Grohe, M.: Parametrized complexity and subexponential time. Bulletin of the EATCS 84, 71–100 (2004)
49. Flum, J., Grohe, M.: Parameterized complexity theory. Texts in theoretical computer science. Springer, Heidelberg (2006), `http://books.google.com/books?id=VfJz6hvFAjoC`
50. Fortnow, L., Santhanam, R.: Infeasibility of instance compression and succinct PCPs for NP. Journal of Computing and System Sciences 77(1), 91–106 (2011)
51. Frick, M., Grohe, M.: Deciding first-order properties of locally tree-decomposable structures. J. ACM 48(6), 1184–1206 (2001), `http://doi.acm.org/10.1145/504794.504798`

52. Frick, M., Grohe, M.: The complexity of first-order and monadic second-order logic revisited. In: Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science, LICS 2002, pp. 215–224. IEEE Computer Society, Washington, DC (2002)
53. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
54. Goldberg, P.W., Golumbic, M.C., Kaplan, H., Shamir, R.: Four strikes against physical mapping of DNA. Journal of Computational Biology 2(1), 139–152 (1993)
55. Gramm, J., Niedermeier, R., Rossmanith, P.: Exact Solutions for Closest String and Related Problems. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 441–453. Springer, Heidelberg (2001)
56. Grohe, M.: Generalized Model-Checking Problems for First-Order Logic. In: Ferreira, A., Reichel, H. (eds.) STACS 2001. LNCS, vol. 2010, pp. 12–26. Springer, Heidelberg (2001)
57. Grohe, M., Kreutzer, S.: Methods for algorithmic meta theorems. In: Grohe, M., Makowsky, J. (eds.) Model Theoretic Methods in Finite Combinatorics. Contemporary Mathematics, vol. 558. American Mathematical Society (2011)
58. Hallett, M., Mccartin, C.: A faster FPT algorithm for the maximum agreement forest problem. Theory of Computing Systems 41(3), 539–550 (2007)
59. Hermelin, D., Kratsch, S., Soltys, K., Whalstrom, M., Wu, X.: Hierarchies of inefficient kernelization (to appear)
60. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? Journal of Computer and System Sciences 63(4), 512–530 (2001)
61. Karp, R.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
62. Karp, R.: Heuristic algorithms in computational molecular biology. Journal of Computer and System Sciences 77(1), 122–128 (2011), http://dx.doi.org/10.1016/j.jcss.2010.06.009
63. Khot, S., Raman, V.: Parameterized complexity of finding subgraphs with hereditary properties. Theoretical Computer Science 289(2), 997–1008 (2002)
64. Langston, M.A., Perkins, A.D., Saxton, A.M., Scharff, J.A., Voy, B.H.: Innovative computational methods for transcriptomic data analysis: A case study in the use of FPT for practical algorithm design and implementation. The Computer Journal 51(1), 26–38 (2008)
65. Mahajan, M., Raman, V.: Parameterizing above guaranteed values: MaxSat and MaxCut. Journal Algorithms 31(2), 335–354 (1999)
66. Marx, D.: Future directions in parameterized complexity (tentative title) (to appear, 2012)
67. McCartin, C.: Parameterized counting problems. Annals of Pure and Applied Logic 138(1-3), 147–182 (2006)
68. Müller, M.: Randomized Approximations of Parameterized Counting Problems. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 50–59. Springer, Heidelberg (2006)
69. Müller, M.: Valiant-Vazirani Lemmata for various logics. Electronic Colloquium on Computational Complexity (ECCC) 15(063) (2008)
70. Nemhauser, G., Trotter Jr., L.: Vertex packings: Structural properties and algorithms. Mathematical Programming 8, 232–248 (1975)
71. Niedermeier, R.: Invitation to fixed-parameter algorithms. Habilitationschrift, University of Tübingen (2002)

72. Niedermeier, R.: Invitation to fixed-parameter algorithms. Oxford lecture series in mathematics and its applications. Oxford University Press (2006), http://books.google.com/books?id=mAA_OkeJxhsC
73. Niedermeier, R., Rossmanith, P.: A general method to speed up fixed-parameter-tractable algorithms. Information Processing Letters 73, 125–129 (2000), http://theinf1.informatik.uni-jena.de/publications/ipl00.pdf
74. Papadimitriou, C.H., Yannakakis, M.: On the complexity of database queries. In: PODS, pp. 12–19. ACM Press (1997); Journal Version in Journal of Computer System Sciences 58, 407–427 (1999)
75. Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. Operations Research Letters 32(4), 299–301 (2004), http://dx.doi.org/10.1016/j.orl.2003.10.009
76. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. Journal of Algorithms 7(3), 309–322 (1986)
77. Weerapurage, D.P., Eblen, J.D., Rogers, G., Langston, M.A.: Parallel Vertex Cover: A Case Study in Dynamic Load Balancing. In: Chen, J., Ranjan, R. (eds.) Australasian Symposium on Parallel and Distributed Computing (AusPDC 2011). CRPIT, vol. 118, pp. 25–32. ACS, Perth (2011), http://crpit.com/confpapers/CRPITV118Weerapurage.pdf

# The Computer Science of DNA Nanotechnology

Jack H. Lutz [*]

Department of Computer Science
Iowa State University
Ames, IA 50011 USA
lutz@cs.iastate.edu

**Abstract.** DNA nanotechnology, pioneered by Seeman, Winfree, and Rothemund, exploits the information processing capabilities of nucleic acids to *program matter* to do our bidding at atomic and molecular scales. This field is now a rapidly growing interdisciplinary research adventure involving chemists, molecular biologists, computer scientists, materials scientists, electrical and computer engineers, and others. DNA tile assembly, DNA origami, and DNA strand displacement have enabled the programmed self-assembly of complex nanoscale structures, dynamic nanoscale machines, and nanoscale Boolean circuits. Applications on the horizon include patterning of smaller, faster computer chips; nanoscale detectors and instruments for measurement; and in-cell computers that diagnose and treat disease. This talk will survey the role of computer science in making DNA nanotechnology more productive, predictable, and safe. Topics will include the specification and verification of nanoscale systems, the intrinsic universality (a strong version of Turing universality) of self-assembly, the role of randomness in molecular programming, and the essential role of software in the *design* of wet-lab experiments in DNA nanotechnology.

[*] This research was supported by National Science Foundation Grants 0652569 and 1143830.

# The Minimal Cost Reachability Problem in Priced Timed Pushdown Systems

Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman

Uppsala University, Sweden

**Abstract.** This paper introduces the model of *priced timed pushdown systems* as an extension of discrete-timed pushdown systems with a cost model that assigns multidimensional costs to both transitions and stack symbols. For this model, we consider the minimal cost reachability problem: i.e., given a priced timed pushdown system and a target set of configurations, determine the minimal possible cost of any run from the initial to a target configuration. We solve the problem by reducing it to the reachability problem in standard pushdown systems.

## 1 Introduction

Pushdown systems are one of the most widely used models for the study and analysis of recursive systems [11]. Furthermore, several models have been introduced in [5,6,9,10,14] which extend the model by introducing timed behaviors.

We consider a new model for *Timed Pushdown Systems* consisting of pushdown systems augmented with a finite set of (integer valued) clocks. Moreover, the symbols that are pushed into the stack have integer ages measuring the time that has elapsed since they were pushed. A clock can be set to zero simultaneously with any transition. At any moment, the value of a clock is the time elapsed since the last time it was reset. With each transition we associate time-intervals (whose bounds are natural numbers or $\omega$) that restrict the clock values and ages of the sequence of symbols (on the top of the stack) that can be popped.

In parallel, there have been several works on extending the model of timed automata [3] with prices (weights) (see e.g., [4,8]). Priced timed automata are suitable models for embedded systems, where we have to take into consideration the fact that the system behavior may be constrained by the consumption of different types of resources. More precisely, priced timed automata extend classical timed automata with a cost function *Cost* that maps every location and every transition to a nonnegative integer. For a transition, *Cost* gives the cost of performing the transition. For a location, *Cost* gives the cost per time unit for staying in the location. In this manner, we can define, for each run of the system, the accumulated cost of staying in locations and performing transitions.

We study a natural extension of timed pushdown systems, namely *Priced Timed Pushdown Systems* (PTPS). We allow the cost function to map transitions and stack symbols of the pushdown system into vectors of integers (of some given length $k$). Again, for a transition, *Cost* gives the cost of performing the transition;

while for a stack symbol, *Cost* gives the cost per time unit for the symbol to stay in the stack. We consider the *minimal cost reachability problem* for PTPS where, given an *intial configuration* $c_0$ and a set $F$ of *final configurations*, the task is to compute the minimal accumulated cost of a run that reaches $F$ from $c_0$. Here, we assume that $F$ is *regular* (i.e., $F$ can be described using a finite state automaton). Since the set of costs within which we can reach $F$ from $c_0$ is upward closed (regardless of the form of $F$), the minimal cost reachability problem can be reduced, using the construction of Valk and Jantzen [15], to the *cost-threshold problem.* In the latter, we are given a cost vector $\mathbf{v}$, and we want to check if it is possible to reach $F$ from $c_0$ with a cost that does not exceed $\mathbf{v}$.

In this paper, we prove that the cost-threshold problem for PTPS can be reduced to the reachability problem for (unpriced) timed pushdown systems. The idea consists of encoding the cost of a computation in the state of the timed pushdown systems. Moreover, we show that the reachability problem for timed pushdown systems can be reduced to the reachability for (unpriced and untimed) pushdown systems which is decidable. Hence, we get the decidability of the cost-threshold problem for PTPS (and consequently also solving the minimal cost reachability problem for PTPS).

*Related work.* The works in [6,9,10] consider timed pushdown automata. However, the models in these works consider only global clocks which means that the stack symbols are not equipped with clocks. In contrast, we associate with each pushed stack symbol one clock (reflecting its age). In fact, our model can be easily extended such that each stack symbol has several clocks.

In [14], the authors introduce *recursive timed automata*, a model where clocks are considered as variables. A recursive timed automaton allows passing the values of clocks using either *pass-by-value* or *pass-by-reference* mechanisms. This feature is not supported in our model since we do not allow pass-by-value communication between procedures. Moreover, in the recursive timed automaton model, the local clocks of the caller procedure are stopped until the called procedure returns. This makes the semantics of the models incomparable with ours, since all the clocks in our model evolve synchronously.

In [5], the authors define the class of *extended pushdown timed automata* which is a pushdown automaton enriched with a set of clocks, with an additional stack used to store/restore clock valuations (which leads to the undecidability of the reachability problem). In our model, clocks are associated with stack symbols and store/restore operations are not allowed.

None of the above works considers prices in their models.

The minimal cost reachability problem has been addressed for several models: priced timed automata (e.g., [4,8]), and priced timed Petri nets ([1,2]). To the best of our knowledge, this is the first work that addresses the problem for PTPS.

## 2 Preliminaries

Let $\mathbb{N}$ denote the non-negative integers, and let $\mathbb{N}^k$ and $\mathbb{N}^k_\omega$ denote the set of vectors of dimension $k$ over $\mathbb{N}$ and $\mathbb{N} \cup \{\omega\}$, respectively ($\omega$ represents the first limit ordinal). We use $\mathbf{0}^k$ (or simply $\mathbf{0}$, depending on the context) to denote the vector of dimension $k$ whose elements have all the value 0, and *Intrv* to denote the set of intervals in $\mathbb{N} \times \mathbb{N}_\omega$. In the context of vectors, *less than* means the standard componentwise ordering $\leq$.

For sets $A$ and $B$, we use $f : A \to B$ to denote that $f$ is a total function that maps $A$ to $B$. We use $[A \to B]$ to denote the set of all total functions from $A$ to $B$. Given a set $A$ with an ordering $\preceq$ and a subset $B \subseteq A$, $B$ is said to be *upward closed* in $A$ if $b \in B$, $a \in A$ and $b \preceq a$ implies $a \in B$. Given a set $B \subseteq A$, we define the *upward closure* $B \uparrow$ (resp. *downward closure* $B \downarrow$) to be the set $\{a \in A \,|\, \exists b \in B : \, b \preceq a \, (resp. \, a \preceq b)\}$.

Let $\Sigma$ be an alphabet. We denote by $\Sigma^*$ (resp. $\Sigma^+$) the set of all *words* (resp. non-empty words) over $\Sigma$, and by $\epsilon$ the empty word. A language is a (possibly infinite) set of words. The length of a word $w \in \Sigma^*$ is denoted by $|w|$. (We assume that $|\epsilon| = 0$). For every $i : 1 \leq i \leq |w|$, let $w(i)$ be the symbol at position $i$ in $w$. For $a \in \Sigma$, we write $a \in w$ if $a$ appears in $w$, i.e., $a = w(i)$ for some $i : 1 \leq i \leq |w|$. We use $|w|_a$ to denote the number of occurrences of $a$ in $w$.

## 3 Priced Timed Pushdown Systems

The *Timed Pushdown System* (TPS) model is an extension of pushdown systems augmented with a finite set of (integer valued) clocks. Moreover, the symbols that are pushed into the stack have integer ages measuring the time that has elapsed since they were pushed. A clock can be set to zero simultaneously with any transition. At any moment, the value of a clock is the time elapsed since the last time it was reset. With each transition we associate time-intervals (whose bounds are natural numbers or $\omega$) which restrict the clock values and ages of the sequence of symbols (on the top of the stack) that can be popped. Then, we extend this model to priced timed pushdown systems (PTPS) by assigning multidimensional costs to both transitions (action costs) and stack symbols (storage costs). Each firing of a discrete transition costs the assigned cost vector. The cost of a timed transition depends on the stack content. For each stack symbol $\gamma$, if the stack contains $k_1$ occurrences of $\gamma$ and the cost of storing $\gamma$ per time unit is $\mathbf{v}$, then firing a timed transition will add $k_1\mathbf{v}$ to the current accumulated cost.

*Syntax.* A Priced Timed Pushdown System (PTPS) is a tuple $N = (X, Q, \Gamma, \Delta, cost)$ where $X$ is a finite set of clocks, $Q$ is a finite set of states, $\Gamma$ is the stack alphabet, $cost : (\Gamma \cup \Delta) \to \mathbb{N}^k$ is a function assigning (multidimensional) firing costs to transitions and storage costs to stack symbols, and $\Delta$ is a finite set of transition rules of the form:

$$(q, (\alpha_1, J_1) \cdots (\alpha_m, J_m)) \xrightarrow{(\phi\,,\,R)} (q', (\gamma_1, I_1) \cdots (\gamma_n, I_n))$$

where (1) $q, q' \in Q$ are two states, (2) $\alpha_1 \cdots \alpha_m \in \Gamma^*$ is the word to be popped such that the age of each symbol $\alpha_i \in \Gamma$ should be in the interval $J_i$ for all $i : 1 \leq i \leq m$, (3) $\phi : X \to Intrv$ is a clock constraint over $X$ (i.e., the valuation of a clock $x \in X$ should be in $\phi(x)$), (4) $R \subseteq X$ is the set of clocks to be reset, and (5) $\gamma_1 \cdots \gamma_n \in \Gamma^*$ is the word to be pushed into the stack such that the initial age of each symbol $\gamma_j \in \Gamma$ should be in the interval $I_j$ for all $j : 1 \leq j \leq n$. Observe that the set $\Delta$ of transition rules can be defined as a finite subset of $\big(Q \times (\Gamma \times \mathit{Intrv})^*\big) \times \big([X \to \mathit{Intrv}] \times 2^X\big) \times \big(Q \times (\Gamma \times \mathit{Intrv})^*\big)$.

If for every $\gamma \in \Gamma$ and $\delta \in \Delta$, $\mathit{cost}(\gamma) = \mathit{cost}(\delta) = \mathbf{0}$ (i.e., the cost of any transition or stack symbol is $\mathbf{0}$), then $N$ is called a *(unpriced) Timed Pushdown System* (TPS), which can be described by the tuple $(X, Q, \Gamma, \Delta)$. Moreover, if $\Delta \subseteq \big(Q \times (\Gamma \times I_0)^*\big) \times ([X \to I_0] \times 2^X) \times (Q \times (\Gamma \times I_0)^*)$ with $I_0 = \{[0..\omega]\}$, then the TPS $N$ is called an *unpriced and untimed pushdown system*, or simply Pushdown System (PS), which can be described by the tuple $(Q, \Gamma, \Delta)$.

*Configurations.* A configuration $c$ of $N$ is a triple $(q, \nu, w)$ where $q \in Q$ is a state, $\nu : X \to \mathbb{N}$ is a clock valuation, and $w \in (\Gamma \times \mathbb{N})^*$ is the stack content. Observe that the stack contains a sequence of pairs representing the pushed symbols and their ages. Let $\mathit{Conf}(N)$ denote the set of all configurations of $N$.

A set of configurations $C \subseteq \mathit{Conf}(N)$ is said to be *regular* if there are a set $Q' \subseteq Q$ of states, a **finite** set $\Psi \subseteq [X \to \mathbb{N}]$ of clock valuations, and a **regular** language $L$ over $(\Gamma \times \mathit{Intrv})$ such that $C = \{(q, \nu, w) \,|\, q \in Q', \nu \in \Psi, w \text{ satisfies some } l \in L\}$. A language $L$ over $(\Gamma \times \mathit{Intrv})$ is regular if and only if there is a finite state automaton $\mathcal{A}$ over the alphabet $(\Gamma \times \mathit{Intrv})$ such that the language accepted by $\mathcal{A}$ is precisely $L$.

Let $c = (q, \nu, (\gamma_1, a_1) \cdots (\gamma_n, a_n))$ be a configuration of $N$ with $q \in Q$, $\nu \in [X \to \mathbb{N}]$, and $(\gamma_i, a_i) \in \Gamma \times \mathbb{N}$ for all $i : 1 \leq i \leq n$. Then, we use $c^{+1}$ to denote the configuration $(q, \nu', w')$ defined as follows: (1) $\nu'(x) = \nu(x) + 1$ for all $x \in X$ (i.e., the value of each clock is increased by one time unit), and (2) $w' = (\gamma_1, a_1 + 1) \cdots (\gamma_n, a_n + 1)$ (i.e., the age of each symbol $\gamma_i$ in the stack is also increased by one time unit). Note that $(\gamma_1, a_1)$ is at the top and $(\gamma_n, a_n)$ is at the bottom of the stack.

Let $\gamma_1, \ldots, \gamma_n \in \Gamma$, $a_1, \ldots, a_n \in \mathbb{N}$ and $I_1, \ldots, I_n \in \mathit{Intrv}$. We say that the stack content $w = (\gamma_1, a_1) \cdots (\gamma_n, a_n)$ satisfies the stack constraint $r = (\gamma_1, I_1) \cdots (\gamma_n, I_n)$ (denoted by $w \in r$) if and only if $a_i \in I_i$ for all $i : 1 \leq i \leq n$ (i.e., the age of each symbol $\gamma_i$ belongs to the interval $I_i$).

*Transition relation.* We define two transition relations on the set of configurations of $N$: timed and discrete. The timed transition relation increases the value of each clock and the age of each pushed stack symbol by one. Formally, $c \to_{\mathit{time}} c'$ if and only if $c' = c^{+1}$.

We define the discrete transition relation $\to_D$ as $\bigcup_{\delta \in \Delta} \to_\delta$ where $\to_\delta$ represents the effect of *performing* the discrete transition $\delta$. More precisely, let us assume that the transition $\delta$ is of the form $(q, r) \xrightarrow{(\phi, R)} (q', r')$ where $q, q' \in Q$ are states, $r, r' \in (\Gamma \times \mathit{Intrv})^*$ are stack constraints, $R \subseteq C$ is the set of clocks to be reset, and $\phi : X \to \mathit{Intrv}$ is a clock constraint over $X$. For configurations $c = (q, \nu, w)$ and $c' = (q', \nu', w')$, we have $c \to_\delta c'$ if and only if

– There are $u, u'' \in (\Gamma \times \mathbb{N})^*$ such that $w = u \cdot u''$ and $u \in r$. The transition $t$ can be performed only if there is a word $u$, at the top of the stack, satisfying the constraint $r$ (i.e., $u \in r$), and if this is the case $u$ can be popped.

– There is $u' \in (\Gamma \times \mathbb{N})^*$ such that $w' = u' \cdot u''$ and $u' \in r'$. The newly pushed word $u'$ into the stack should satisfy the stack constraint given by $r'$.
– $\nu(x) \in \phi(x)$ for all $x \in X$. The current clock value of $x$ should satisfy the time constraint imposed by $\phi$.
– $\nu'(x) = 0$ for all $x \in R$ and $\nu'(x') = \nu(x')$ for all $x' \notin R$. The clocks in $R$ are reset to 0, and thus start counting time with respect to the time of occurrence of this transition.

We write $\rightarrow_N$ (or simply $\rightarrow$ when it is clear from the context) to denote the transition relation given by $\rightarrow_{time} \cup \rightarrow_D$. We use $\rightarrow^*$ to denote the reflexive-transitive closure of $\rightarrow$. It is easy to extend $\rightarrow^*$ to sets of configurations.

Let $c, c' \in \mathit{Conf}(N)$. A computation $\pi$ of $N$ from $c$ to $c'$ is of the form $c_0 \rightarrow c_1 \rightarrow \cdots \rightarrow c_n$ where $c_0 = c$, $c_n = c'$, and $c_i \rightarrow c_{i+1}$ for all $i : 0 \leq i < n$. We write $c \xrightarrow{\pi} c'$ to denote that there is a computation $\pi$ of $N$ from $c$ to $c'$. We define $\mathit{Reach}(c) := \{c'' \mid c \rightarrow^* c''\}$ as the set of configurations reachable from $c$.

**The cost of computations.** The cost of a stack content $w \in (\Gamma \times \mathbb{N})^*$ is defined as $\mathit{Cost}(w) = \sum_{(\gamma,a) \in \Gamma \times \mathbb{N}} |w|_{(\gamma,a)}\ \mathit{cost}(\gamma)$. Intuitively, the cost of $w$ corresponds to the sum, over the stack symbols $\gamma \in \Gamma$, of the number of occurrences of $\gamma$ in $w$ multiplied by its individual cost $\mathit{cost}(\gamma)$. In the same way, we can define the cost of stack constraints: Given a stack constraint $r \in (\Gamma \times \mathit{Intrv})^*$, $\mathit{Cost}(r) = \mathit{Cost}(u)$ for some $u \in r$. Notice that the function $\mathit{Cost}$ over stack constraints is well-defined since $\mathit{Cost}(u) = \mathit{Cost}(u')$ for all $u, u' \in r$.

The cost of a discrete transition $\delta$ is defined as $\mathit{Cost}(c \rightarrow_\delta c') = \mathit{cost}(\delta)$ and the cost of a timed transition is defined as $\mathit{Cost}(c \rightarrow_{time} c^{+1}) = \mathit{Cost}(w)$ where $c$ is of the form $(q, \nu, w)$. The cost of a computation $\pi = c_0 \rightarrow c_1 \rightarrow \cdots \rightarrow c_n$ is the sum of all transition costs, i.e., $\mathit{Cost}(\pi) = \sum_{i=0}^{n-1} \mathit{Cost}(c_i \rightarrow c_{i+1})$.

*The Cost-Threshold Problem.* We study the problem of computing the minimal cost for reaching a configuration in a given regular target set.

<u>Cost-Threshold Problem</u>

**Instance:** A PTPS $N = (X, Q, \Gamma, \Delta, \mathit{cost})$ with an initial configuration $c_0 \in \mathit{Conf}(N)$, a regular set $F$ of final configurations and a vector $\mathbf{v} \in \mathbb{N}_\omega^k$.

**Question:** Does there exist $c \in F$ and $c_0 \xrightarrow{\pi} c$ such that $\mathit{Cost}(\pi) \leq \mathbf{v}$?

The cost-threshold problem is called the reachability problem when the PTPS $N$ is an unpriced (un)timed pushdown system. In fact, in the case of unpriced (un)timed pushdown systems, the cost-threshold problem boils down to checking whether there exist a configuration $c \in F$ and a computation $\pi$ such that $c_0 \xrightarrow{\pi} c$. Moreover, in the case of pushdown systems, the regular set $F$ of final configurations can be defined using the tuple $Q' \times L$ where $Q' \subseteq Q$ and $L$ is a regular language over $\Gamma$.

Since all costs are non-negative (in the set $\mathbb{N}^k$), the standard componentwise ordering $\leq$ on costs is a well-quasi order and thus every upward closed set of costs has finitely many minimal elements [12]. Moreover, if we have a positive instance of the cost-threshold problem with some allowed cost $\mathbf{v}$ then any modified instance with some allowed cost $\mathbf{v}' \geq \mathbf{v}$ will also be positive. Thus the set of possible costs in the cost-threshold problem is upward-closed. In this case, the Valk-Jantzen theorem [15] implies that the set of minimal possible costs can be computed if the Cost-Threshold problem is decidable.

**Theorem 1 (Valk and Jantzen [15]).** *Given an upward-closed set $\mathbf{V} \subseteq \mathbb{N}^k$, the finite set $\mathbf{V}_{min}$ of minimal elements of $\mathbf{V}$ is computable if for any vector $\mathbf{v} \in \mathbb{N}_\omega^k$ the predicate $\mathbf{v}{\downarrow} \cap \mathbf{V} \neq \emptyset$ is decidable.*

<u>Computing Minimal Possible Costs</u>

**Instance:** A PTPS $N$ with an initial configuration $c_0$ and a regular set $F$ of final configurations.

**Question:** Compute the minimal possible costs of reaching $F$, i.e., the finitely many minimal elements of $\mathbf{V}_F = \{\mathbf{v} \in \mathbb{N}^k \mid \exists c \in F, \pi.\, c_0 \xrightarrow{\pi} c \wedge \mathit{Cost}(\pi) \leq \mathbf{v}\}$.

Since $\mathbf{V}_F$ is upward-closed and for any vector $\mathbf{v}$, if $F$ is reachable with a cost less than $\mathbf{v}$, then $\mathbf{v} \in \mathbf{V}_F$, we have by Theorem 1:

**Theorem 2.** *Let $N$ be a PTPS with an initial configuration $c_0$ and a regular set of final configurations $F$. Then, computing the minimal possible costs of reaching $F$ can be reduced to the cost-threshold problem of reaching $F$ from $c_0$.*

## 4 From Priced to Unpriced Timed Pushdown Systems

In this section, we show that it is possible to reduce the cost-threshold reachability problem for PTPS to the reachability problem for TPS.

**Theorem 3.** *The cost-threshold reachability problem for priced timed pushdown systems can be reduced to the reachability problem for timed pushdown systems.*

The rest of this section is devoted to the proof of this theorem. Consider an instance of the cost-threshold reachability problem for priced timed pushdown systems. Let $N = (X, Q, \Gamma, \Delta, \mathit{cost})$ be a PTPS with $\mathit{cost}(\Gamma) \cup \mathit{cost}(\Delta) \subseteq \mathbb{N}^k$, $c_0 \in \mathit{Conf}(N)$ be the initial configuration, $F$ be the regular set of final configurations, and $\mathbf{v} = (v_1, \ldots, v_k) \in \mathbb{N}_\omega^k$.

First, for every $i : 1 \leq i \leq k$, if $v_i = \omega$ then we replace $v_i$ by 0 and set the $i$-th component of the cost function $\mathit{cost}$ of each stack symbol $\gamma \in \Gamma$ and transition $\delta \in \Delta$ to 0. Hence, we can assume that $\mathbf{v} = (v_1, \ldots, v_k) \in \mathbb{N}^k$.

In the following, we construct a TPS $N' = (X', Q', \Gamma', \Delta')$ such that the cost-threshold reachability for $N$ is reducible to the reachability problem for $N'$. The idea is to simulate any computation of $N$ by a computation of $N'$. During the simulation, $N'$ keeps track in its state of the current total cost of the computation (performed so far) and the current cost of the stack content. So, when a discrete transition $\delta$ is performed, $N'$ adds the cost of $\delta$ to the current total cost. (Observe that the total cost should always be less than the vector $\mathbf{v}$ since we are only interested in computations whose total cost is less than $\mathbf{v}$.) Now, when a timed transition is performed, the cost of the stack content (if it is less than $\mathbf{v}$) will be added to the total cost.

The first main difficulty is that, during one time unit, the cost of the stack content can be strictly greater than (or incomparable to) $\mathbf{v}$. To overcome this difficulty, $N'$ keeps track of the cost of the stack content up to $\mathbf{v}$, and uses the special symbol $\top$ when the current cost of the stack content is not less than $\mathbf{v}$. Moreover, $N'$ adds the cost of the current stack (stored in the state of $N'$) to each newly pushed stack symbol $\gamma \in \Gamma$. Hence, a stack symbol of $N'$ is of one of the following two forms: either $(\gamma, \mathbf{v}')$ or $(\gamma, \top)$ where $\gamma \in \Gamma$ and $\mathbf{v}' \leq \mathbf{v}$. Then, every transition performed by $N'$ preserves the invariant between the stack cost stored in the current state of $N'$ and the topmost stack symbol in the stack. This means that if the current cost of the stack content stored in the state is $s$ and the topmost stack symbol is $(\gamma, s')$ then the following condition should be satisfied: If $s' = \top$ or $\mathit{cost}(\gamma) + s' \not\leq \mathbf{v}$ then $s = \top$; otherwise $s = \mathit{cost}(\gamma) + s'$.

The second difficulty is that timed transitions of $N'$ are performed in a non-deterministic manner. However, $N'$ needs to know when a timed transition has been performed in order to add the current stack content cost to the total cost stored in its state. For this we add a new clock $x_{new}$ which is used to detect if one unit of time has elapsed (i.e., a timed transition has been performed) or not. Then, discrete transitions of $N$ can only be simulated by $N'$ when the value of the clock $x_{new}$ is equal to 0. If the current value of $x_{new}$ is 1, then $N'$ will add the current stack content cost (if it is less than $\mathbf{v}$) to the total cost stored in its state and reset $x_{new}$. Formally, $N'$ is defined as follows:

- Let $\top$ be a symbol. The stack alphabet $\Gamma'$ of $N'$ is defined by the set $\Gamma \times (\mathbf{v}\!\downarrow \cup \{\top\})$. Intuitively, a stack symbol of the form $(\gamma, \mathbf{v}')$ (resp. $(\gamma, \top)$) corresponds to the fact that the cost of the stack content before pushing the stack symbol $(\gamma, \mathbf{v}')$ (resp. $(\gamma, \top)$) into the stack is $\mathbf{v}'$ (resp. not less than $\mathbf{v}$).
- A state of $N'$ is of the form $(q, t, s)$ where $q \in Q$ is a state of $N$, $t \in \mathbf{v}\!\downarrow$ is the current accumulated total cost (which should be less than $\mathbf{v}$), and $s \in (\mathbf{v}\!\downarrow \cup \{\top\})$ reflects the current cost of the stack content (if the current cost $\mathbf{v}'$ of the stack content is less than $\mathbf{v}$ then $s = \mathbf{v}'$ otherwise $s = \top$).
- The set of clocks of $N'$ contains all the clocks of $N$ and a new clock $x_{new}$ such that $x_{new} \notin X$ (i.e., $X' = X \cup \{x_{new}\}$). The clock $x_{new}$ is used to detect when a timed transition is performed in order to add the cost of the current stack content (if it is less than $\mathbf{v}$) to the total cost.
- The set $\Delta'$ is the smallest set of rules satisfying the following conditions:

  **1. Simulating a discrete transition of** $N$**:** Let $t, t' \in (\mathbf{v}\!\downarrow)$ be two vectors less than $\mathbf{v}$ and $s, s' \in (\mathbf{v}\!\downarrow \cup \{\top\})$. For every discrete transition $\delta \in \Delta$ of the form $(q, (\alpha_1, J_1) \cdots (\alpha_m, J_m)) \xrightarrow{(\phi,R)} (q', (\gamma_1, I_1) \cdots (\gamma_n, I_n))$, the TPS $N'$ has a transition of the form $((q, t, s), r) \xrightarrow{(\phi',R)} ((q', t', s'), r')$, with $r = ((\alpha_1, a_1), J_1) \cdots ((\alpha_m, a_m), J_m)$ and $r' = ((\gamma_1, b_1), I_1) \cdots ((\gamma_n, b_n), I_n)$, if the following conditions are satisfied:

  - $t' := t + \mathit{cost}(\delta)$. The cost of the transition $\delta$ is add to the total cost $t$.
  - $\phi'(x) = \phi(x)$ if $x \in X$, and $\phi'(x_{new}) = [0..0]$. The discrete transition can be performed by $N'$ only if the clock value of $x_{new}$ is equal to 0.

- Let $d = s$ if $r = \epsilon$ (i.e., $m = 0$) otherwise $d = a_m$. Intuitively, $d$ represents the stack cost after popping a word satisfying the stack constraint $r$.
  - If $d = \top$ then $s' = \top$ and $b_i = \top$ for all $i : 1 \leq i \leq n$. In fact, if the current cost of the stack after popping a word satisfying the constraint $r$ is not less than $\mathbf{v}$ (since $d = \top$), then it is the case after pushing any stack symbol (so, $b_i = \top$ and $s' = \top$).
  - If $d \leq \mathbf{v}$ then for every $i : 1 \leq i \leq n$, let $\mathbf{v_i} = d + \sum_{j=i}^{n} cost(\gamma_j)$ be the sum of $d$ (i.e, the current stack cost after popping a word satisfying the constraint $r$) and the cost of the sequence of stack symbols $\gamma_i \gamma_{i+1} \cdots \gamma_n$. Then, if $\mathbf{v_1} \leq \mathbf{v}$ then $s' = \mathbf{v_1}$, otherwise $s' = \top$. Moreover, $b_n = d$ and for every $i : 1 \leq i < n$, if $\mathbf{v_{i+1}} \leq \mathbf{v}$ then $b_i = \mathbf{v_{i+1}}$, otherwise $b_i = \top$.

**2. Simulating a timed transition of** $N$**:** For every state $q \in Q$ and vectors $t, t', s \in (\mathbf{v}{\downarrow})$ less than $\mathbf{v}$, the TPS $N'$ has a transition $\delta_{time}$ of the form $((q, t, s), \epsilon) \xrightarrow{(\phi, R)} ((q, t', s), \epsilon)$ if the following conditions hold:

- $\phi(x) = [0..\omega]$ for all $x \in X$ and $\phi(x_{new}) = [1..1]$. This means that one time unit has passed, and hence, a timed transition has been performed.
- $t' := t + s$. The current cost of the stack content is added to the current total cost since a timed transition has been performed.
- $R = \{x_{new}\}$. Only the clock $x_{new}$ is reset to 0.

*Relation between* $N$ *and* $N'$*.* Let $w = ((\gamma_1, a_1), y_1)((\gamma_2, a_2), y_2) \cdots ((\gamma_n, a_n), y_n)$ be a possible stack content of $N'$ where $\gamma_i \in \Gamma$, $a_i \in (\mathbf{v}\downarrow) \cup \{\top\}$, and $y_i \in \mathbb{N}$ for all $i : 1 \leq i \leq n$. Recall that the symbol $((\gamma_n, a_n), y_n)$ is in the bottom of the stack and $((\gamma_1, a_1), y_1)$ is the topmost stack symbol. For every $i : 1 \leq i \leq n$, let $\mathbf{v_i} = \sum_{j=i}^{n} cost(\gamma_j)$ be the cost of the sequence of stack symbols $\gamma_i \cdots \gamma_n$.

Then, $w$ is a *valid* stack content of $N'$ if and only if $a_n = \mathbf{0}$ and for every $i : 1 \leq i < n$, if $\mathbf{v_{i+1}} \leq \mathbf{v}$ then $a_i = \mathbf{v_{i+1}}$, otherwise $a_i = \top$. Observe that the transition relation of $N'$ preserves the validity of the stack content in any configuration reachable from a configuration whose stack content is initially valid.

Now, we define the mapping $T$ that associates, for every configuration $c = (q, \nu, (\gamma_1, y_1) \cdots (\gamma_n, y_n))$ of $N$ and every vector $t \leq \mathbf{v}$, a configuration $T(c, t, \mathbf{v}) = ((q, t, s), \nu', w')$ of $N'$ such that the following conditions are satisfied:

– $w'$ is the unique valid stack content of the form:

$$((\gamma_1, a_1), y_1)((\gamma_2, a_2), y_2) \cdots ((\gamma_n, a_n), y_n)$$

Notice that such a valid stack configuration exists by definition and unique.

– If $Cost(\gamma_1 \cdots \gamma_n) \leq \mathbf{v}$ then $s = Cost(\gamma_1 \cdots \gamma_n)$, otherwise $s = \top$.

– $\nu'(x) = \nu(x)$ if $x \in X$ and $\nu'(x_{new}) = 0$.

Observe that $T$ is a bijection. The definition of the mapping $T$ is extended in the straightforward manner to sets of configurations of $N$ and costs less than $\mathbf{v}$. Finally, Theorem 3 is an immediate consequence of the following lemma:

**Lemma 4.** *Let* $F' = T(F, \mathbf{v}\downarrow, \mathbf{v})$*. There exists a configuration* $c \in F$ *and a computation* $c_0 \xrightarrow{\pi} c$ *of* $N$ *s.t.* $Cost(\pi) \leq \mathbf{v}$ *iff there is a configuration* $c' \in F'$ *s.t.* $T(c_0, \mathbf{0}, \mathbf{v}) \rightarrow_{N'}^{*} c'$*. Moreover, the set* $F'$ *of configurations of* $N'$ *is regular.*

## 5 From Timed Pushdown Systems to Pushdown Systems

In this section, we show that it is possible to reduce the reachability problem for TPS to its corresponding problem for PS. Let us first show that the reachability problem for TPS can be reduced to the reachability problem for TPS between two configurations with empty stack and where the value of each clock is 0.

**Lemma 5.** *Let $N$ be a timed pushdown system, $c_0 \in Conf(N)$ be an initial configuration, and $F \subseteq Conf(N)$ be a regular set of final configurations. Then, it is possible to construct a timed pushdown system $N' = (X', Q', \mathit{\Gamma}', \mathit{\Delta}')$ and two configurations $c'_0 = (q_0, \nu, \epsilon)$ and $c'_f = (q_f, \nu, \epsilon)$ such that $q_0, q_f \in Q'$, $\nu(x) = 0$ for all $x \in X'$, and there is $c \in F$ such that $c_0 \rightarrow^*_N c$ iff $c'_0 \rightarrow^*_{N'} c'_f$.*

*Proof.* The proof of this lemma is similar to the case of standard pushdown systems. In fact, any computation of $N'$ will be divided in three phases. In the first phase, the TPS $N'$ performs some push and nop transitions in order to reach the configuration $c_0$. Then, $N'$ starts to mimic the behavior of $N$. Finally, $N'$ performs a sequence of pop and nop transitions to check, in a nondeterministic way, whether the current reached configuration is in $F$, and then resets all clocks to zero. This can be done since $F$ is a regular set of configurations. □

Let us now prove that it is possible to reduce the reachability problem for timed pushdown systems to the reachability problem for pushdown systems.

**Theorem 6.** *The reachability problem for timed pushdown system can be reduced to the same problem for pushdown systems.*

The rest of this section is devoted to the proof of Theorem 6. Consider an instance of the reachability problem for timed pushdown systems: Let $N = (X, Q, \mathit{\Gamma}, \mathit{\Delta})$ be a timed pushdown system, $c_0 \in Conf(N)$ be an initial configuration, $F$ be a regular set of final configurations. From Lemma 5, we can assume without loss of generality that $c_0 = (q_0, \nu, \epsilon)$ and $F = \{(q_f, \nu, \epsilon)\}$ where $q_0, q_f \in Q$ and $\nu(x) = 0$ for all $x \in X$.

Let *max* be the maximal natural number appearing in the time intervals of the transition relation $\mathit{\Delta}$. Observe that if the value of a clock or the age of a stack symbol is strictly greater than *max* then we can assume without loss of generality that it is $\omega$.

In the following, we construct a pushdown system $N' = (Q', \mathit{\Gamma}', \mathit{\Delta}')$ such that the reachability problem for $N$ is reducible to the reachability problem in $N'$. The main idea is to simulate a computation of $N$ by a computation of $N'$. During the simulation process, the pushdown system keeps track of the value of each clock of $N$ in its state up to the value *max*. In fact, if the value of a clock of $N$ is strictly greater than *max*, $N'$ can assume without loss of generality that the value of such clock is $\omega$. Observe that this simulation process of the clocks of $N$ cannot be extended to the ages of the stack symbols in the stack, since $N'$ has limited access to its stack (it can only access the top part). To overcome this problem, we add to each stack symbol $\gamma \in \mathit{\Gamma}$ of $N$, its initial age, and the time that has elapsed between the pushing of $\gamma$ and the last time it was the topmost symbol. As in the case of the clocks of $N$, we can assume that the initial age and the time elapsed associated with each stack symbol of $N'$ is in $[0 \mathinner{..} max] \cup \{\omega\}$. Now, every performed transition in $N'$ should preserve that the age of the topmost stack symbol is given by the sum of its initial age and the time elapsed so far. Hence, when a symbol is popped from the stack, the elapsed time of the new topmost stack symbol must be updated by adding to it the elapsed time of the popped symbol. Formally, the pushdown system $N'$ is defined as follows:

– The set $Q'$ of states of $N'$ is $Q \times [X \to ([0 \mathinner{..} max] \cup \{\omega\})]$. A state of $N'$ is then of the form $(q, \nu)$ where $q \in Q$ is the current state of $N$ and $\nu$ is a mapping from $X$ to $[0 \mathinner{..} max] \cup \{\omega\}$. If a clock $x \in X$ has a value strictly greater than $max$ in $N$ then $\nu(x) = \omega$ in $N'$, otherwise the value of the clock $x$ in $N$ is $\nu(x)$.

– The stack alphabet $\Gamma'$ is $(\Gamma \times ([0 \mathinner{..} max] \cup \{\omega\})^2) \cup \{(\perp, 0, 0)\}$ where $(\perp, 0, 0)$ is a special symbol used to mark the bottom of the stack. A stack symbol of the form $(\gamma, y, z)$ on the top of the stack of $N'$ corresponds to the fact that $\gamma$ is the topmost stack symbol of $N$ such that if its initial age $i$ is strictly greater than $max$ then $y = \omega$, otherwise $y = i$. Moreover, if $e$ is the elapsed time while $\gamma$ is in the stack of $N$, then if $e$ is strictly greater than $max$ then $z = \omega$, otherwise $z = e$. Notice that the age of $\gamma$ in $N$ is $(i + e)$.

– The set $\Delta'$ is the smallest set of rules satisfying the following conditions:

**1. Simulating a discrete transition of** $N$**:** Let $(\gamma, y, z) \in \Gamma'$ be a stack symbol of $N'$. Then, for every discrete transition $\delta \in \Delta$ of $N$ of the form $(q, (\alpha_1, J_1) \cdots (\alpha_m, J_m)) \xrightarrow{(\phi, R)} (q', (\gamma_1, I_1) \cdots (\gamma_n, I_n))$, $N'$ has a transition of the form: $((q, \nu), (\alpha_1, y_1, z_1) \cdots (\alpha_m, y_m, z_m)(\gamma, y, z)) \to ((q', \nu'), (\gamma_1, y'_1, z'_1) \cdots (\gamma_n, y'_n, z'_n)\ (\gamma, y', z'))$ if the following conditions hold:

- For every clock $x \in X$, $\nu(x) \in \phi(x)$. The valuation of each clock $x$ should satisfy the time constraint given by $\phi$.
- For every clock $x \in X$, $\nu'(x) = 0$ if $x \in R$, otherwise $\nu'(x) = \nu(x)$. Since no unit of time has been elapsed, only the clocks that are in $R$ are reset.
- For every $i : 1 \leq i \leq m$, we have $(y_i + \sum_{j=1}^{i} z_j) \in J_i$. This means that the age of the stack symbol $\alpha_i$ is given by the sum of the time elapsed at each stack symbol $\alpha_j$ with $j : 1 \leq j \leq i$ and its initial age $y_i$ when it was pushed.
- If $(\gamma, y, z) = (\perp, 0, 0)$ then $(\gamma, y', z') = (\perp, 0, 0)$. This means that the bottom stack symbol can never be popped.
- If $\gamma \in \Gamma$ then let $e = z + \sum_{j=1}^{m} z_j$. If $e \leq max$ then $z' = e$, otherwise $z' = \omega$. The elapsed time of the new topmost stack symbol $\gamma$ must be updated using the elapsed time of each individual symbol $\alpha_i$ with $j : 1 \leq j \leq m$. Moreover, we have $y' = y$ since the initial age of $\gamma$ remains the same.

- For every $i : 1 \leq i \leq n$, we have $y'_i \in ([0 \mathinner{..} max] \cup \{\omega\}) \cap I_i$ and $z'_i = 0$. The newly pushed stack symbol $\gamma_i$ has an initial age in $I_i$. Moreover, the elapsed time of $\gamma_i$ is 0 since it is newly pushed into the stack.

**2. Simulating a timed transition of $N$:** For every state $q \in Q$, $\nu, \nu' \in [X \to ([0 \mathinner{..} max] \cup \{\omega\})]$, and $(\gamma, y, z), (\gamma, y', z') \in \Gamma'$, the pushdown system $N'$ has a transition $\delta_{time}$ of the form $((q, \nu), (\gamma, y, z)) \to ((q, \nu'), (\gamma, y', z'))$ if the following conditions hold:

- For every clock $x \in X$, if $\nu(x)+1 \leq max$ then $\nu'(x) = \nu(x)+1$, otherwise $\nu'(x) = \omega$. Since a timed transition is performed, the pushdown system $N'$ should update the valuation of all its clocks accordingly.
- If $(\gamma, y, z) = (\bot, 0, 0)$ then $(\gamma, y', z') = (\bot, 0, 0)$. This means that the bottom stack symbol can never be popped or modified.
- If $\gamma \in \Gamma$ then let $e = z + 1$. If $e \leq max$ then $z' = e$, otherwise $z' = \omega$. Moreover, we have $y' = y$. The elapsed time of the topmost stack symbol must be updated since one time unit has passed.

Finally, the relation between $N$ and $N'$ is given by the following lemma:

**Lemma 7.** $(q_0, \nu, \epsilon) \to^*_N (q_f, \nu, \epsilon)$ *iff* $((q_0, \nu), (\bot, 0, 0)) \to^*_{N'} ((q_f, \nu), (\bot, 0, 0))$.

Since the reachability problem for pushdown systems is decidable (see for example [7,13]), and from Theorem 3 and Theorem 6, we can conclude that the cost-threshold problem is also decidable.

**Corollary 8.** *The cost-threshold problem for PTPS is decidable.*

Moreover, from Theorem 2 and Corollary 8, we obtain:

**Corollary 9.** *Let $N$ be a priced timed pushdown system with an initial configuration $c_0 \in Conf(N)$ and a regular set $F$ of final configurations. Then, it possible to compute the minimal possible costs of reaching $F$.*

## 6 Conclusion

We introduced the model of (discrete) timed pushdown systems which is an extension of pushdown systems. We showed that the reachability problem for timed pushdown systems is decidable and can be reduced to the reachability problem for standard pushdown systems (which is a decidable problem). Moreover, we have considered priced timed pushdown systems which is an extension of timed pushdown systems with a cost model. We proved that the cost-threshold problem is decidable (by a reduction to the reachability problem for timed pushdown systems). As a consequence of this result, the minimal cost reachability problem for priced timed pushdown systems is decidable.

A challenging problem which we are currently considering is to extend our results to the case of dense-timed pushdown systems.

## References

1. Abdulla, P.A., Mayr, R.: Minimal Cost Reachability/Coverability in Priced Timed Petri Nets. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 348–363. Springer, Heidelberg (2009)
2. Abdulla, P.A., Mayr, R.: Computing optimal coverability costs in priced timed Petri nets. In: LICS 2011, pp. 399–408. IEEE Computer Society (2011)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
4. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. Theor. Comput. Sci. 318(3), 297–322 (2004)
5. Benerecetti, M., Minopoli, S., Peron, A.: Analysis of timed recursive state machines. In: Markey, N., Wijsen, J. (eds.) TIME, pp. 61–68. IEEE Computer Society (2010)
6. Bouajjani, A., Echahed, R., Robbana, R.: On the Automatic Verification of Systems with Continuous Variables and Unbounded Discrete Data Structures. In: Antsaklis, P.J., Kohn, W., Nerode, A., Sastry, S.S. (eds.) HS 1994. LNCS, vol. 999, pp. 64–85. Springer, Heidelberg (1995)
7. 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)
8. Bouyer, P., Cassez, F., Fleury, E., Larsen, K.G.: Optimal Strategies in Priced Timed Game Automata. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 148–160. Springer, Heidelberg (2004)
9. Dang, Z., Ibarra, O., Bultan, T., Kemmerer, R., Su, J.: Binary Reachability Analysis of Discrete Pushdown Timed Automata. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 69–84. Springer, Heidelberg (2000)
10. Emmi, M., Majumdar, R.: Decision Problems for the Verification of Real-Time Software. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 200–211. Springer, Heidelberg (2006)
11. Esparza, J., Knoop, J.: An Automata-Theoretic Approach to Interprocedural Data-Flow Analysis. In: Thomas, W. (ed.) FOSSACS 1999. LNCS, vol. 1578, pp. 14–30. Springer, Heidelberg (1999)
12. Higman, G.: Ordering by divisibility in abstract algebras. Proc. London Math. Soc (3) 2(7), 326–336 (1952)
13. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley Longman Publishing Co., Inc. (2006)
14. Trivedi, A., Wojtczak, D.: Recursive Timed Automata. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 306–324. Springer, Heidelberg (2010)
15. Valk, R., Jantzen, M.: The residue of vector sets with applications to decidability problems in Petri nets. Acta Inf. 21, 643–674 (1985)

# Unification Modulo Chaining

Siva Anantharaman[1], Christopher Bouchard[2,⋆], Paliath Narendran[2,⋆⋆], and Michael Rusinowitch[3,⋆⋆⋆]

[1] LIFO - Université d'Orléans, France
siva@univ-orleans.fr
[2] University at Albany–SUNY, USA
dran@cs.albany.edu
[3] Loria-INRIA Lorraine, Nancy, France
rusi@loria.fr

**Abstract.** We model chaining in terms of a simple, convergent, rewrite system over a signature with two disjoint sorts: *list* and *element.* By interpreting a particular symbol of this signature suitably, the rewrite system can model several practical situations of interest. An inference procedure is presented for deciding the unification problem modulo this rewrite system. The procedure is modular in the following sense: any given problem is handled by a system of 'list-inferences', and the set of equations thus derived between the element-terms of the problem is then handed over to any ('black-box') procedure which is complete for solving these element-equations. An example of application of this unification procedure is given, as attack detection on a Needham-Schroeder like protocol employing the CBC encryption mode.

**Keywords:** Equational unification, Block chaining, Protocol.

## 1 Introduction

The technique of *chaining* is applicable in many situations. A simple case is e.g., when we want to calculate the partial sums (resp. products) of a (not necessarily bounded) list of integers, with a given 'base' integer; such a list of partial sums (resp. products) can be calculated, incrementally, with the help of the following set of equations:

$$bc(nil,\, z) = nil, \qquad bc(cons(x, Y),\, z) = cons(h(x, z),\ bc(Y,\, h(x, z)))$$

where $nil$ is the empty list, $z$ is the given base integer, $x$ is an integer variable, and $Y$ is the given list of integers. The partial sums (resp. products) are returned as a list, by evaluating the function $bc$, when $h(x, z)$ is interpreted as the sum (resp. product) of $x$ with the given base integer $z$.

A more sophisticated example is the Cipher Block Chaining encryption mode (CBC, in short), employed in cryptography. This mode uses the AC-operator

⋆ Research supported in part by NSF grant CNS-0905286.
⋆⋆ Research supported in part by NSF grant CNS-0905286.
⋆⋆⋆ Research supported in part by FP7 NESSOS Project.

exclusive-or (XOR) for 'chaining the ciphers across the message blocks'; here is how this is done: let $\oplus$ stand for XOR (which we let distribute over block concatenation), and $M = m \cdot M'$ be a message decomposed as a concatenation of a single message block $m$ with the rest of the message $M'$. Then the encryption $e(M,x)$ of $M$ with $x$ as the key is given by $e(M,x) = e(m,x) \cdot e(M' \oplus e(m,x), x)$ (cf. e.g., [12]). The above set of equations also models the CBC encryption mode: the function $h(x,y)$ will stand in this case for the encryption $e(x \oplus y, k)$ of the message-term $x$ XOR-ed with the initialization vector $y$, using the public key $k$ of the recipient of the message. Actually, our interest in the equational theory defined by the above two equations was motivated by the possibility of such a modeling for Cipher Block Chaining, and the fact that rewrite as well as unification techniques are often employable, with success, for the formal analysis of cryptographic protocols (cf. e.g., [3,5,7,6], and also the concluding section).

This paper is organized as follows. In Section 2 we introduce our notation and the basic notions used in the sequel; we shall observe, in particular, that the two equations above can be turned into rewrite rules and form a convergent rewrite system over a 2-sorted signature: *lists* and *elements*. Our concern in Section 3 is the unification problem modulo this rewrite system; we present a 2-level inference system (corresponding, in a way, to the two sorts of the signature) for solving this problem. Although our main aim is to develop on the unification problem under the assumption that $h$ is an interpreted function symbol (as in the two situations illustrated above), for the sake of completeness we shall also consider the case where $h$ is a free uninterpreted symbol. The soundness and completeness of our inference procedure are established in Section 4. We shall see that while the complexity of the unification problem is polynomial over the size of the problem when $h$ is uninterpreted, it turns out to be NP-complete when $h$ is interpreted so that the rewrite system models CBC encryption.

## 2 Notation and Preliminaries

We consider a ranked signature $\Sigma$, with two *disjoint* sorts: $\tau_e$ and $\tau_l$, consisting of binary functions *bc, cons, h*, and a constant *nil*, and typed as follows:

$$bc: \ \tau_l \times \tau_e \to \tau_l \quad , \quad cons: \ \tau_e \times \tau_l \to \tau_l \quad , \quad h: \ \tau_e \times \tau_e \to \tau_e \quad , \quad nil: \tau_l .$$

We also assume given a set $\mathcal{X}$ of countably many variables; the objects of our study are the (well-typed) terms of the algebra $\mathcal{T}(\Sigma, \mathcal{X})$; terms of the type $\tau_e$ will be referred to as *elements*; and those of the type $\tau_l$ as *lists.* For better readability, the set of variables $\mathcal{X}$ will be divided into two subsets: those to which 'lists' can get assigned will be denoted with upper-case letters as: $X, Y, Z, U, V, W, \ldots$, with possible suffixes or primes; these will be said to be variables of type $\tau_l$; variables to which 'elements' can get assigned will be denoted with lower-case letters, as: $x, y, z, u, v, w, \ldots$, with possible suffixes or primes; these will be said to be variables of type $\tau_e$. The theory we shall be studying in this paper is defined by the two axioms (equations) already mentioned in the Introduction:

$$bc(nil,\, z) = nil, \qquad bc(cons(x, Y),\, z) = cons(h(x, z),\, bc(Y,\, h(x, z)))$$

It is easy to see that these axioms can both be oriented left-to-right under a suitable *lexicographic path ordering (lpo)* (cf. e.g., [8]), and that they form then a convergent – i.e., confluent and terminating – 2-sorted rewrite system. The (sorted) equational theory defined by the two axioms above will be denoted in the sequel as $\mathcal{BC}$, and referred to as 'block chaining'.

In the case where $h$ is a free uninterpreted symbol, $h$ will be fully cancellative in the sense that for any terms $s_1, t_1, s_2, t_2$, we have: $h(s_1, t_1) \approx_{\mathcal{BC}} h(s_2, t_2)$ if and only if $s_1 \approx_{\mathcal{BC}} s_2$ and $t_1 \approx_{\mathcal{BC}} t_2$. But when $h$ is interpreted, e.g., as for CBC, this is no longer true; in that case, $h$ will only be *semi-cancellative* in the following sense: for any terms $s_1, s_2, t$, we have: $h(s_1, t) \approx_{\mathcal{BC}} h(s_2, t)$ if and only if $s_1 \approx_{\mathcal{BC}} s_2$ and $h(t, s_1) \approx_{\mathcal{BC}} h(t, s_2)$ if and only if $s_1 \approx_{\mathcal{BC}} s_2$. *In the sequel, we shall always assume the symbol h to be semi-cancellative.*

Our concern in this work is the equational unification problem modulo $\mathcal{BC}$. We assume without loss of generality (wlog) that any given $\mathcal{BC}$-unification problem $\mathcal{P}$ is in a *standard form,* i.e., $\mathcal{P}$ is given as a set of equations $\mathcal{EQ}$, each having one of the following forms:

$$U =^? V,\ U =^? bc(V, y),\ U =^? cons(v, W),\ U =^? nil,$$
$$u =^? v,\ \ v =^? h(w, x),\ u =^? const$$

where *const* stands for any ground constant of sort $\tau_e$. The first four kinds of equations – the ones with a list variable on the left-hand side – are called *list equations,* and the rest (those which have an element variable on the left-hand side) are called *element equations.* For any problem $\mathcal{P}$ in standard form, $\mathcal{L}(\mathcal{P})$ will denote the subset formed of its list equations, and $\mathcal{E}(\mathcal{P})$ the subset of element equations. A set of element equations is said to be in *dag-solved form* (or *d-solved form*) if and only if they can be arranged as a list

$$x_1 =^? t_1,\ \ldots,\ x_n =^? t_n$$

where: (a) each left-hand side $x_i$ is a distinct variable, and (b) $\forall\, 1 \leq i \leq j \leq n$: $x_i$ does not occur in $t_j$ ([11]). Such a notion is naturally extended to sets of list equations as well. In the next section we give an inference system for solving any $\mathcal{BC}$-unification problem in standard form. Its rules will transform any given problem $\mathcal{P}$ into one in d-solved form.

For better comprehension, and to facilitate presentation, in the sequel we shall denote by $\mathcal{BC}_0$ the theory defined by the rewrite system $\mathcal{BC}$ when $h$ is a free uninterpreted symbol; and by $\mathcal{BC}_1$ the theory defined in the case where $h$ is interpreted so that $\mathcal{BC}$ models the CBC encryption mode. Any development presented below for the theory $\mathcal{BC}$ – without further precision on $h$ – is meant as one which will be valid for both $\mathcal{BC}_0$ and $\mathcal{BC}_1$.

## 3 Inference System for $\mathcal{BC}$-Unification

The inference rules have to consider two kinds of equations: the rules for the *list equations* in $\mathcal{P}$, i.e., equations whose left-hand sides (lhs) are variables of type $\tau_l$, and the rules for the *element equations,* i.e., equations whose lhs are variables of type $\tau_e$. Our method of solving any given unification problem will be 'modular' on these two sets of equations: The list inference rules will be shown to terminate under suitable conditions, and then all we will need to do is to solve the resulting set of element equations for $h$.

A few technical points need to be mentioned before we formulate our inference rules. Note first that it is not hard to see that *cons* is cancellative; by this we mean that $cons(s_1, t_1) \approx_{\mathcal{BC}} cons(s_2, t_2)$, for terms $s_1$, $s_2$, $t_1$, $t_2$, if and only if $s_1 \approx_{\mathcal{BC}} s_2$ and $t_1 \approx_{\mathcal{BC}} t_2$. On the other hand, since we assume that $h$ is semi-cancellative we can show, by structural induction, that *bc* is also *conditionally* semi-cancellative (depending on whether its first argument is *nil* or not) [1].

Note that $U =^? bc(U, x)$ is solvable by the substitution $\{U := nil\}$; in fact this equation forces $U$ to be *nil*, as would also the set of equations: $U =^? bc(V, y),\ V =^? bc(U, x)$. Cycles of this kind have therefore to be checked to determine whether a list variable is forced to be *nil*. This can be effectively done by defining a relation $>_{bc}$ over type $\tau_l$ variables:

$$U \ >_{bc} \ V \text{ iff there is an equation } U =^? bc(V, X).$$

If $X >_{bc}{}^{+} X$ then $X$ has to be *nil*. A set **nonnil** of variables that cannot be *nil* for any unifying substitution is defined, recursively, as follows:

- if $U =^? cons(x, V)$ is an equation then $U \in$ **nonnil**.
- if $U =^? bc(V, x)$ is an equation and $V \in$ **nonnil** then $U \in$ **nonnil**.
- if $U =^? bc(V, x)$ is an equation and $U \in$ **nonnil** then $V \in$ **nonnil**.

We also have to account for cases where an 'occur-check' succeeds on some list variable, and the problem will be unsolvable. The simplest among such cases is when we have an equation of the form $U =^? cons(z, U)$ in $\mathcal{EQ}$. But one could have more complex unsolvable cases, where the equations involve both *cons* and *bc*; e.g., when $\mathcal{EQ}$ contains equations of the form: $U =^? cons(x, V), U =^? bc(V, y)$; the problem will be unsolvable in such a case: indeed, from the axioms of $\mathcal{BC}$, one deduces that $V$ must be of the form $V =^? cons(v, V')$, for some $v$ and $V'$, then $x$ must be of the form $x =^? h(v, y)$, and subsequently $V =^? bc(V', x)$, and we are back to a set of equations of the same format. We need to infer failure in such a case; for that, we define two relations on the list variables of $\mathcal{EQ}$:

$U \ >_{cons} \ V$ iff $U =^? cons(z, V)$, for some $z$.

$U \sim_{bc} V$ iff $U =^? bc(V, w)$, or $V =^? bc(U, w)$, for some $w$.

Note that $\sim_{bc}$ is the symmetric closure of the relation $>_{bc}$. The reflexive, symmetric and transitive closure of $>_{bc}$ will be denoted as $\sim^*_{bc}$. In what follows, for any list variable $U$, we denote by $[U]$ the equivalence class of list variables that get equated to $U$; that is to say: $[U] = \{V \mid U =^? V \in \mathcal{P} \text{ or } V =^? U \in \mathcal{P}\}$

**Definition 1.** *Let $G_l = G_l(\mathcal{P})$ be the graph whose nodes are the equivalence classes on the list variables of $\mathcal{P}$, with arcs defined as follows: From a node $[U]$ on $G_l$ there is a* directed *arc to a (not necessarily different) node $[V]$ on $G_l$ iff:*

*(a) Either $U \ >_{cons} \ V$: in which case the arc is labeled with $>_{cons}$,*
*Or $U >_{bc} V$: in which case the arc is labeled with $>_{bc}$.*

*(b) In the latter case, $G_l$ will also have a* two-sided (undirected) *arc between $[U]$ and $[V]$, which is labeled with $\sim_{bc}$.*

*(c) On the set of nodes on $G_l$, we define a partial relation $\succ_l$ by setting: $[U] \succ_l [V]$ iff there is a path on $G_l$ from $[U]$ to $[V]$, at least one arc of which has label $>_{cons}$.* In other words: $\succ_l = \sim_{bc}^{*} \circ >_{cons} \circ (\sim_{bc} \cup >_{cons})^{*}$.

*(d) A list variable $U$ of $\mathcal{P}$ is said to* violate occur-check *iff $[U] \succ_l [U]$ on $G_l$.*

The graph $G_l = G_l(\mathcal{P})$ is called the *propagation graph* for $\mathcal{P}$. We formulate now the inference rules for the list equations in $\mathcal{P}$.

### 3.1 Inference System $\mathcal{INF}_l$ for List-Equations

(L1) *Variable Elimination*:

$$\frac{\{U =^? V\} \uplus \mathcal{EQ}}{\{U =^? V\} \cup [V/U](\mathcal{EQ})} \qquad \text{if } U \text{ occurs in } \mathcal{EQ}$$

(L2) *Cancellation on cons*:

$$\frac{\mathcal{EQ} \uplus \{U =^? cons(v, W),\ U =^? cons(x, V)\}}{\mathcal{EQ} \cup \{U =^? cons(x, V),\ v =^? x,\ W =^? V\}}$$

(L3.a) *Nil solution-1*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? nil\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil\}}$$

(L3.b) *Nil solution-2*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ V =^? nil\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil\}}$$

(L3.c) *Nil solution-3*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x)\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil\}} \qquad \text{if } V >_{bc}^{*} U$$

(L4.a) *Semi-Cancellation on bc*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? bc(W, x)\}}{\mathcal{EQ} \cup \{U =^? bc(W, x),\ V =^? W\}}$$

(L4.b) *Pushing bc below cons*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V, x),\ U =^? bc(W, y)\}}{\begin{array}{c}\mathcal{EQ} \cup \{V =^? cons(v, Z),\ W =^? cons(w, Z),\ U =^? cons(u, U'),\\ U' =^? bc(Z, u),\ u =^? h(v, x),\ u =^? h(w, y)\}\end{array}}$$

if $U \in$ **nonnil**

(L5) *Splitting*:

$$\frac{\mathcal{EQ} \uplus \{U =^? cons(x, U_1),\ U =^? bc(V, z)\}}{\mathcal{EQ} \cup \{U =^? cons(x, U_1),\ x =^? h(y, z),\ U_1 =^? bc(V_1, x),\ V =^? cons(y, V_1)\}}$$

(L6) *Occur-Check Violation*:

$$\frac{\mathcal{EQ}}{FAIL} \qquad \text{if } U \text{ occurs in } \mathcal{EQ}, \text{ and } U \succ_l U \text{ on the graph } G_l$$

(L7) *Size Conflict*:

$$\frac{\mathcal{EQ} \uplus \{U =^? cons(v, W),\ U =^? nil\}}{FAIL}$$

The symbol '$\uplus$' in the premises of the above inference rules stands for disjoint set union (and '$\cup$' for usual set union). The role of the Variable Elimination inference rule (L1) is to keep the propagation graph of $\mathcal{P}$ irredundant: each variable has a unique representative node on $G_l(\mathcal{P})$, up to variable equality. This rule is applied most eagerly. Rules (L2), (L3.a)–(L3.c) and (L4.a) come next in priority, and then (L4.b). The Splitting rule (L5) is applied in the "laziest" fashion, i.e., (L5) is applied only when no other rule is applicable. The above inference rules are all "don't-care" nondeterministic. (Note: Type-inference failure is assumed to be checked implicitly; no explicit rule is given.)

The validity of the rule (L4.b) ('Pushing *bc* below *cons*') results from the cancellativity of *cons* and the semi-cancellativity of *bc*; note that the variables $Z, U', u$ in the 'inferred part' of this rule (L4.b) might need to be fresh; the same is true also for the variables $y, V_2$ in the inferred part of the Splitting rule (but, in either case this is not obligatory, if the equations already present can be used for applying these rules). We show now that such an introduction of fresh variables cannot go for ever, and that the above 7 don't-care nondeterministic rules suffice, essentially, for deciding *unifiability* modulo the axioms of $\mathcal{BC}$.

**Proposition 2.** *Let $\mathcal{P}$ be any $\mathcal{BC}$-unification problem, given in standard form. The system $\mathcal{INF}_l$ of list inference rules, given above, terminates on $\mathcal{P}$ in polynomially many steps.*

*Proof.* The variable elimination rule (L1) removes nodes from the propagation graph, while the list inference rules (L2) through (L4.a) eliminate a (directed) outgoing arc from some node of $G_l$. Thus their termination is easy to check. Therefore the system of list inference rules will terminate if the splitting rule (L5) terminates and the rule (L4.b) (*Pushing bc below cons*) terminates. We show that if occur-check violation (L7) does not occur, then applications of the rule (L5) or of the rule (L4.b) cannot go on forever.

First of all, observe that though the splitting rule may introduce new variables, the number of $\sim_{bc}^*$-equivalence classes of nodes cannot increase, since the (possibly) new variable $V_1$ belongs to the same equivalence class as $U_1$ ($V_1 \sim_{bc} U_1$). Thus applying the splitting rule (L5) on a list-equation $U = bc(V, z)$ removes that equation and creates a list equation of the form $U_1 = bc(V_1, x)$ for some list variables $U_1$ and $V_1$, such that $V \sim_{bc} U >_{cons} U_1 \sim_{bc} V_1$.

Suppose now that applying the splitting rule does not terminate. Then, at some stage, the graph of the derived problem will have a sequence of nodes of the form $U_0 = U >_{cons} U_1 >_{cons} \cdots >_{cons} U_n$, such that its number of nodes $n$ strictly exceeds the initial number of $\sim_{bc}^*$-equivalence classes – which cannot increase under splitting, as was observed above. So there must exist indices $0 \leq i < j \leq n$ such that $U_j \sim_{bc}^* U_i$; in other words, we would have $U_i \succ_l U_i$, and that would have caused the inference procedure to terminate with FAIL. We conclude therefore that applying the splitting rule must terminate.

The same kind of reasoning also works for the rule (L4.b): an application of that rule removes two list equations of the form $U =^? bc(V, x), U =^? bc(W, y)$ and creates a list equation of the form $U' = bc(Z, u)$ (plus some other *cons* and/or element equations), where $U = cons(u, U')$ so $U >_{cons} U'$. We conclude, for the same reason as above, that applying rule (L4.b) must terminate.

To show that the number of steps is polynomial on the input problem size, note first that the number of nodes and edges on the Propagation Graph of $\mathcal{P}$ is polynomial in the size of $\mathcal{P}$, and that number decreases under all the inferences other than (L4.b) and (L5). But these latter rules do not increase the number of $\sim_{bc}$-edges. Let the *level* of a $\sim_{bc}$-edge be the length of the longest *cons*-path to that edge. (L4.b) and (L5) remove one $\sim_{bc}$-edge and add a new one at a greater level. Since, the length of any *cons*-chain is polynomially bounded, (L4.b) and (L5) can only be applied a polynomial number of times. □

A set of equations will be said to be *L-reduced* if none of the above inference rules (L1) through (L7) is applicable.

**Unification modulo $\mathcal{BC}$:** The rules (L1) through (L7) are not enough to show the existence of a unifier modulo $\mathcal{BC}$. The subset of element equations, $\mathcal{E}(\mathcal{P})$, may not be solvable; for example, the presence of an element equation of the form $\{x =^? h(x, z)\}$ should lead to failure. However, we have the following:

**Proposition 3.** *If $\mathcal{L}(\mathcal{P})$ is in L-reduced form, then $\mathcal{P}$ is unifiable modulo $\mathcal{BC}$ if and only if the set $\mathcal{E}(\mathcal{P})$ of its element equations is solvable.*

*Proof.* If $\mathcal{L}(\mathcal{P})$ is $L$-reduced, then setting every list variable that is not in **nonnil** to *nil* will lead to *a unifier* for $\mathcal{L}(\mathcal{P})$, modulo $\mathcal{BC}$, provided $\mathcal{E}(\mathcal{P})$ is solvable. □

Recall that $\mathcal{BC}_0$ is the theory defined by $\mathcal{BC}$ when $h$ is uninterpreted.

**Proposition 4.** *Let $\mathcal{P}$ be any $\mathcal{BC}_0$-unification problem, given in standard form. Unifiability of $\mathcal{P}$ modulo $\mathcal{BC}_0$ is decidable in polynomial time (wrt the size of $\mathcal{P}$).*

*Proof.* If the inferences of $\mathcal{INF}_l$ applied to $\mathcal{P}$ lead to failure, then $\mathcal{P}$ is not unifiable modulo $\mathcal{BC}$; so assume that this is not the case, and replace $\mathcal{P}$ by an equivalent problem which is $L$-reduced, deduced in polynomially many steps by Proposition 2. By Proposition 3, the unifiability modulo $\mathcal{BC}$ of such a $\mathcal{P}$ amounts to checking if the set $\mathcal{E}(\mathcal{P})$ of its element equations is solvable. We are in the case where $h$ is uninterpreted, so to solve $\mathcal{E}(\mathcal{P})$ we apply the rules for standard unification, and check for their termination without failure; this can be done in polynomial time [4]. (In this case, $h$ is fully cancellative.) □

It can be seen that while termination of the above inference rules guarantees the *existence* of a unifier (provided the element equations are syntactically solvable), the resulting $L$-reduced system may not lead directly to a unifier. For instance, the $L$-reduced system of list equations $\{U =^? bc(V, x),\ U =^? bc(V, y)\}$ is unifiable, with two incomparable unifiers, namely

$$\{x := y,\, U := bc(V, y)\} \text{ and } \{U := nil,\, V := nil\}$$

To get a complete set of unifiers we need three more inference rules, which are "don't-know" nondeterministic, and to be applied only to $L$-reduced systems:

(L8) *Nil-solution-Branch*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V,x),\ U =^? bc(W,y)\}}{\mathcal{EQ} \cup \{U =^? nil,\ V =^? nil,\ W =^? nil\}}$$

(L9) *Cancellation-Branch on bc*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V,x),\ U =^? bc(W,y)\}}{\begin{array}{c}\mathcal{EQ} \cup \{V =^? cons(v,Z),\ W =^? cons(w,Z),\ U =^? cons(u,U'),\\ U' =^? bc(Z,u),\ u =^? h(v,x),\ u =^? h(w,y)\}\end{array}}$$

(L10) *Standard Unification on bc*:

$$\frac{\mathcal{EQ} \uplus \{U =^? bc(V,x),\ U =^? bc(W,y)\}}{\mathcal{EQ} \cup \{U =^? bc(W,y),\ V =^? W,\ x =^? y\}}$$

Rule (L9) nondeterministically "guesses" $U$ to be in **nonnil**. The inference system thus extended will be referred to as $\mathcal{INF}_l'$. We establish now a technical result, valid whether or not $h$ is interpreted:

**Proposition 5.** *Let $\mathcal{P}$ be any $\mathcal{BC}$-unification problem in standard form, to which none of the inferences of $\mathcal{INF}_l'$ is applicable. Then its set of list-equations is in d-solved form.*

*Proof.* If none of the equations in $\mathcal{P}$ involve $bc$ or $cons$ (i.e., all equations are between list variables), then the proposition is proved by rule (L1).

Observe first that if $\mathcal{INF}_l$ is inapplicable to $\mathcal{P}$, then, on the propagation graph $G_l$ for $\mathcal{P}$, there is *at most one outgoing directed arc* of $G_l$ at any node $U$: Otherwise, suppose there are two distinct outgoing arcs at some node $U$ on $G_l$; if both directed arcs bear the label $>_{cons}$, then rule (L2) of $\mathcal{INF}_l$ would apply; if both bear the label $>_{bc}$, then one of (L4.a), (L4.b), (L9), (L10) would apply; the only remaining case is where one of the outgoing arcs is labeled with $>_{cons}$ and the other has label $>_{bc}$, but then the splitting rule (L5) would apply.

Consider now any given connected component $\Gamma$ of $G_l$. There can be no directed cycle from any node $U$ on $\Gamma$ to itself: otherwise the Occur-Check-Violation rule (L6) would have applied. It follows, from this observation and the preceding one, that there is a unique *end-node* $U_0$ on $\Gamma$ – i.e., a node from which there is *no directed outgoing arc* –, and also that for any given node $U$ on $\Gamma$, there is a unique well-defined directed path leading from $U$ to that end-node $U_0$.

It follows easily from these that the left-hand-side list-variables of $\mathcal{P}$ (on the different connected components of $G_l$) can be ordered suitably so as to satisfy the condition for $\mathcal{P}$ to be in $d$-solved form. □

*Example 6.* The following $\mathcal{BC}_0$-unification problem is in standard form:

$$U =^? cons(x,W),\ U =^? bc(V,y),\ W =^? bc(V_3,y),\ x =^? h(z,y),\ y =^? a$$

We apply (L5) (*Splitting*) and write $V =^? cons(v_2,V_2)$, with $v_2, V_2$ fresh:

$$U =^? cons(x,W),\ W =^? bc(V_2,x),\ W =^? bc(V_3,y),\ V =^? cons(v_2,V_2),$$
$$x =^? h(v_2,y),\ x =^? h(z,y),\ y =^? a$$

We apply the cancellativity of $h$ and (element-)variable elimination:

$$U =^? cons(x,W),\ W =^? bc(V_2,x),\ W =^? bc(V_3,y),\ V =^? cons(v_2,V_2),$$
$$x =^? h(v_2,y),\ z =^? v_2,\ y =^? a$$

No rule of $\mathcal{INF}_l$ is applicable; but we can nondeterministically apply (L8):

$$U =^? cons(x, W),\ W =^? nil,\ V_2 =^? nil,\ V_3 =^? nil,\ V =^? cons(v_2, V_2),$$
$$x =^? h(v_2, y),\ z =^? v_2,\ y =^? a$$

These equations, in $d$-solved form, give a solution to the original problem. $\square$

Our concern in this paper is the unification problem modulo $\mathcal{BC}$. When $h$ is uninterpreted, we saw that this unification is decidable in polynomial time. But when $h$ is interpreted so that $\mathcal{BC}$ models CBC, we shall see that unification modulo $\mathcal{BC}_1$ is NP-complete.

## 4 Solving a $\mathcal{BC}$-Unification Problem

Let $\mathcal{P}$ be a $\mathcal{BC}$-Unification problem, given in standard form. We assume that $\mathcal{INF}_l$ has terminated without failure on $\mathcal{P}$; we saw, in the preceding section (Proposition 5), that $\mathcal{P}$ is then in $d$-solved form. We also assume that we have a sound and complete procedure for solving the element equations of $\mathcal{P}$, that we shall denote as $\mathcal{INF}_e$. For the theory $\mathcal{BC}_0$ where $h$ is uninterpreted, we know (Proposition 4) that $\mathcal{INF}_e$ is standard unification, with cancellation rules for $h$, and failure in case of 'symbol clash'. For the theory $\mathcal{BC}_1$, where $h(x, y)$ is interpreted as $e(x \oplus y, k)$ for some fixed key $k$, $\mathcal{INF}_e$ will have rules for semi-cancellation on $h$ and $e$, besides the rules for unification modulo XOR in some fixed procedure, that we assume given once and for all.

In all cases, we shall consider $\mathcal{INF}_e$ as a black-box that either returns most general unifiers ($mgu$s) for the element equations of $\mathcal{P}$, or a failure message when these are not satisfiable. Note that $\mathcal{INF}_e$ is unitary for $\mathcal{BC}_0$ and finitary for $\mathcal{BC}_1$. For any problem $\mathcal{P}$ in $d$-solved form, satisfiable under the theory $\mathcal{BC}_0$, there is a unique mgu, as expressed by the equations of $\mathcal{P}$ themselves (cf. also [11]), that we shall denote by $\theta_{\mathcal{P}}$. Under $\mathcal{BC}_1$ there could be more than one (but finitely many) mgu's; we shall agree to denote by $\theta_{\mathcal{P}}$ any one among them. The entire procedure for solving any $\mathcal{BC}$-unification problem $\mathcal{P}$, given in standard form, can now be synthesized as a nondeterministic algorithm:

**The Algorithm $\mathcal{A}$:** Given a $\mathcal{BC}$-unification problem $\mathcal{P}$, in standard form.
$G_l$ = Propagation graph for $\mathcal{P}$. $\mathcal{INF}_l'$ = Inference procedure for $\mathcal{L}(\mathcal{P})$.
$\mathcal{INF}_e$ = (Complete) Procedure for solving the equations of $\mathcal{E}(\mathcal{P})$.

1 Compute a standard form for $\mathcal{P}$, to which the mandatory inferences of $\mathcal{INF}_l$ are no longer applicable. If this leads to failure, exit with FAIL. Otherwise, replace $\mathcal{P}$ by this standard form.
2 Apply the "don't-know" nondeterministic rules (L8)-(L10), – followed by the rules of $\mathcal{INF}_l$ as needed – until the equations no longer get modified by the inference rules (L1)-(L10). If this leads to failure, exit with FAIL.
3 Apply the procedure $\mathcal{INF}_e$ for solving the element-equations in $\mathcal{E}(\mathcal{P})$; if this leads to failure, exit with FAIL.
4 Otherwise let $\sigma$ be the substitution on the variables of $\mathcal{P}$ as expressed by the resulting equations. Return $\sigma$ as a solution to $\mathcal{P}$.

**Proposition 7.** *The algorithm $\mathcal{A}$ is sound and complete.*

*Proof.* The soundness of $\mathcal{A}$ follows from the soundness (assumed) of $\mathcal{INF}_e$ and that of $\mathcal{INF}'_l$, which is easily checked: If $\mathcal{P}'$ is any problem derived from $\mathcal{P}$ by applying any of these inference rules, then any solution for $\mathcal{P}'$ corresponds to a solution for $\mathcal{P}$. The completeness of $\mathcal{A}$ follows from the completeness (assumed) of $\mathcal{INF}_e$, and the completeness of $\mathcal{INF}'_l$ that we prove below:

**Lemma 8.** *If $\sigma$ is a solution for a given $\mathcal{BC}$-unification problem $\mathcal{P}$ in standard form, then there is a sequence of $\mathcal{INF}'_l$-inference steps that transforms $\mathcal{P}$ into a problem $\mathcal{P}'$ in d-solved form such that $\sigma$ is an instance of $\theta_{\mathcal{P}'}$ (modulo $\mathcal{BC}$).*

*Proof sketch.* The proof is by case analysis. We may assume, without loss of generality, that $\mathcal{P}$ is $L$-reduced (i.e., the mandatory inferences of $\mathcal{INF}'_l$ have all been applied). If $\mathcal{P}$ is already in $d$-solved form then we are done since $\sigma \preceq_{\mathcal{BC}} \theta_{\mathcal{P}}$, for some mgu $\theta_{\mathcal{P}}$. If $\mathcal{P}$ is not in $d$-solved form, then we have to consider several cases, depending on the possible inference branches. We will just illustrate one such case. Suppose there are two equations $U =^? bc(Z, v)$ and $U =^? bc(Y, w)$ in $\mathcal{P}$. If $\sigma(v) =_{\mathcal{BC}} \sigma(w)$, then we must have $\sigma(Z) =_{\mathcal{BC}} \sigma(Y)$, so $\sigma$ must be a solution for the problem obtained by applying the rule (L10). If $\sigma(v) \neq_{\mathcal{BC}} \sigma(w)$, then $\sigma$ must be a solution to the problem derived under rule (L8) or (L9). Now, we know that the inference steps always terminate, so such a reasoning can be completed into an inductive argument, to prove the lemma. □

## 4.1 $\mathcal{BC}_1$-Unification Is NP-Complete

Recall that $\mathcal{BC}_1$ is the theory defined by $\mathcal{BC}$ when the symbol $h$ is interpreted so that $\mathcal{BC}$ models CBC.

**Proposition 9.** *Unifiability modulo the theory $\mathcal{BC}_1$ is NP-complete.*

*Proof.* NP-hardness follows from the fact that general unification modulo XOR is NP-complete [10]. We deduce the NP-upper bound from the following facts:
- For any given $\mathcal{BC}$-unification problem, computing a standard form is in polynomial time, wrt the size of the problem.
- Given a standard form, the propagation graph can be constructed in polynomial time (wrt its number of variables).
- Applying (L1)-(L10) till termination takes only polynomially many steps.
- Extracting the set of element-equations from the set of equations obtained in the previous step is in P.
- Solving the element-equations with the procedure $\mathcal{INF}_e$ using unification modulo XOR is in NP. □

## 4.2 An Illustrative Example

The following public key protocol is known to be secure ([9]):

$$A \to B : \{A, m\}_{kb}$$
$$B \to A : \{B, m\}_{ka}$$

where $A, B$ are the participants of the protocol session, $m$ is a message that they intend secret for others, and $kb$ (resp. $ka$) is the public key of $B$ (resp. $A$).

However, if the CBC encryption mode is assumed and the message blocks are all of the same size, then this protocol becomes insecure; here is why. Let $e_Z(x)$ stand for the encryption $e(x, kz)$ with the public key $kz$ of any principal $Z$. What $A$ sends to $B$ is $cons(A, cons(m, nil))$, or $[A, m]$ in ML-notation, encrypted using $B$'s public-key. Under the CBC encryption mode, this will be sent as:

$$A \to B : [\, e_B(A \oplus v),\ e_B(m \oplus e_B(A \oplus v))\,].$$

Here $\oplus$ stands for XOR and $v$ is the initialization vector that $A$ and $B$ have agreed upon. But then, some other agent $I$, entitled to open a session with $B$ with initialization vector $w$, can get hold of the first block (namely: $e_B(A \oplus v)$) as well as the second block of what $A$ sent to $B$, namely $e_B(m, e_B(A \oplus v))$; (s)he can then send the following as a 'bona fide message' to $B$:

$$I \to B : [\, e_B(I \oplus w),\ e_B(m \oplus e_B(A \oplus v))\,];$$

upon which $B$ will send back to $I$ the following:

$$B \to I : [\, e_I(B \oplus w),\ e_I(\, m \oplus e_B(A \oplus v) \oplus e_B(I \oplus w) \oplus e_I(B \oplus w)\,)\,].$$

It is clear then that the intruder $I$ can get hold of the message $m$ which was intended to remain secret for him/her.

*Example 10.* The above attack (which exploits the properties of XOR: $x \oplus x = 0,\ x \oplus 0 = x$) can be reconstructed by solving a certain $\mathcal{BC}_1$-unification problem. We assume that the names $A, B, I$, as well as the initialization vector $w$, are constants. The message $m$ and the initialization vector $v$, that $A$ and $B$ have agreed upon, are constants intended to be secret for $I$. We shall interpret the function symbol $h$ of $\mathcal{BC}$ in terms of encryption with the public key of $B$: i.e., $h(x, y)$ is $e_B(x \oplus y)$. Due to our CBC-assumption, the ground terms $h(A, v),\ h(m, h(A, v))$ both become 'public' (i.e., accessible to $I$). We shall agree that a ground constant $d$ stands for the public term $h(m, h(A, v))$.

The above attack then amounts to saying that the intruder $I$ can send the following term, as a message to $B$:

$$cons(\, h(I, w),\ [d]\,) = cons(\, h(I, w),\ cons(\, h(m, h(A, v),\ nil\,)\,);$$

which will be considered a legitimate message by $B$ (as mentioned earlier).

The possibility for $I$ to construct the attack along this scheme can then be deduced by solving the following $\mathcal{BC}_1$-unification problem:

$$bc(X, h(I, w)) =^? cons(\, h(m, h(A, v)),\ nil\,)$$

under the condition that terms in the solution are all 'public'. After transforming into standard form, we apply rule (L5) ('*Splitting*') and write: $X =^? cons(z, Y)$, where $z$ and $Y$ are fresh variables. By applying rule (L2) ('*Cancellation on cons*') we deduce $Y := nil$, but we still have to show that following problem:

$$h(z, h(I, w)) =^? h(m, h(A, v))$$

is solvable for $z$, with public terms; using the properties of XOR we get the solution $z := h(\, h(m, h(A, v)),\ h(I, w)\,)$, i.e., $z := h(\, d, h(I, w)\,)$. The solution thus derived for $X$ is $X := [h(\, d, h(I, w)\,)]$. □

It might be of interest to note that the above reasonings do not go through if the namestamps form the *second block* in the messages sent; the protocol could be secure, in such a case, even under CBC.

## 5 Conclusion

We have addressed the unification problem modulo a convergent 2-sorted rewrite system $\mathcal{BC}$, which can model, in particular, the CBC encryption mode of cryptography, by interpreting suitably the function $h$ in $\mathcal{BC}$. A procedure is given for deciding unification modulo $\mathcal{BC}$, which has been shown to be sound and complete when $h$ is either uninterpreted, or interpreted in such a manner. In the uninterpreted case, the procedure is a combination of the inference procedure $\mathcal{INF}_l^{'}$ presented in this paper, with standard unification; in the case where $h$ is interpreted as mentioned above, our unification procedure is a combination of $\mathcal{INF}_l^{'}$ with any complete procedure for deciding unification modulo the associative-commutative theory for XOR.

Although we have given an example of attack detection using our unification procedure on a cryptographic protocol employing CBC encryption, for the formal analysis of cryptographic protocols, unification needs to be generalized as a procedure for solving deduction constraints [13] or, equivalently, as a cap unification procedure [2]; that forms part of our projected future work.

## References

1. Anantharaman, S., Bouchard, C., Narendran, P., Rusinowitch, M.: On cipher block chaining. Tech. Rep. SUNYA-CS-11-04, CS Dept., University at Albany–SUNY (November 2011), http://www.cs.albany.edu/~dran/cbc.pdf
2. Anantharaman, S., Lin, H., Lynch, C., Narendran, P., Rusinowitch, M.: Cap unification: application to protocol security modulo homomorphic encryption. In: Feng, D., Basin, D.A., Liu, P. (eds.) ASIACCS, pp. 192–203. ACM (2010)
3. Anantharaman, S., Narendran, P., Rusinowitch, M.: Intruders with Caps. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 20–35. Springer, Heidelberg (2007)
4. Baader, F., Snyder, W.: Unification theory. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 445–532. Elsevier, MIT Press (2001)
5. Baudet, M.: Deciding security of protocols against off-line guessing attacks. In: Atluri, V., Meadows, C., Juels, A. (eds.) ACM Conference on Computer and Communications Security, pp. 16–25. ACM (2005)
6. Comon-Lundh, H., Shmatikov, V.: Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In: LICS, pp. 271–280. IEEE Computer Society (2003)
7. Comon-Lundh, H., Treinen, R.: Easy Intruder Deductions. In: Dershowitz, N. (ed.) Verification: Theory and Practice. LNCS, vol. 2772, pp. 225–242. Springer, Heidelberg (2004)
8. Dershowitz, N.: Termination of rewriting. J. Symb. Comput. 3(1/2), 69–116 (1987)
9. Dolev, D., Even, S., Karp, R.M.: On the security of ping-pong protocols. Information and Control 55(1-3), 57–68 (1982)

10. Guo, Q., Narendran, P., Wolfram, D.A.: Complexity of nilpotent unification and matching problems. Inf. Comput. 162(1-2), 3–23 (2000)
11. Jouannaud, J.P., Kirchner, C.: Solving equations in abstract algebras: A rule-based survey of unification. In: Computational Logic - Essays in Honor of Alan Robinson, pp. 257–321 (1991)
12. Kremer, S., Ryan, M.: Analysing the vulnerability of protocols to produce known-pair and chosen-text attacks. Electr. Notes Theor. Comput. Sci. 128(5), 87–104 (2005)
13. Millen, J.K., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: ACM Conference on Computer and Communications Security, pp. 166–175 (2001)

# Isomorphism Testing of Boolean Functions Computable by Constant-Depth Circuits

Vikraman Arvind and Yadu Vasudev

The Institute of Mathematical Sciences, Chennai, India
{arvind,yadu}@imsc.res.in

**Abstract.** Given two $n$-variable Boolean functions $f$ and $g$, we study the problem of computing an *$\varepsilon$-approximate isomorphism* between them. I.e. a permutation $\pi$ of the $n$ variables such that $f(x_1, x_2, \ldots, x_n)$ and $g(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$ differ on at most an $\varepsilon$ fraction of all Boolean inputs $\{0,1\}^n$. We give a randomized $2^{O(\sqrt{n}\,\mathrm{polylog}(n))}$ algorithm that computes a $\frac{1}{2^{\mathrm{polylog}(n)}}$-approximate isomorphism between two isomorphic Boolean functions $f$ and $g$ that are given by depth $d$ circuits of poly($n$) size, where $d$ is a constant independent of $n$. In contrast, the best known algorithm for computing an *exact isomorphism* between $n$-ary Boolean functions has running time $2^{O(n)}$ [9] even for functions computed by poly($n$) size DNF formulas. Our algorithm is based on a result for hypergraph isomorphism with bounded edge size [3] and the classical Linial-Mansour-Nisan result on approximating small depth and size Boolean circuits by small degree polynomials using Fourier analysis.

## 1 Introduction

Given two Boolean functions $f, g : \{0,1\}^n \to \{0,1\}$ the *Boolean function isomorphism* is the problem of checking if there is a permutation $\pi$ of the variables such that the Boolean functions $f(x_1, x_2, \ldots, x_n)$ and $g(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$ are equivalent. The functions $f$ and $g$ could be given as input either by Boolean circuits that compute them or simply by black-box access to them. This problem is known to be coNP-hard even when $f$ and $g$ are given by DNF formulas (there is an easy reduction from $\overline{\mathsf{CNFSAT}}$). The problem is in $\Sigma_2^p$ but not known to be in $\mathsf{coNP}$. Furthermore, Agrawal and Thierauf [1] have shown that the problem is not complete for $\Sigma_2^p$ unless the polynomial hierarchy collapses to $\Sigma_3^p$.

On the other hand, the best known algorithm for Boolean function isomorphism, which reduces the problem to Hypergraph Isomorphism, runs in time $2^{O(n)}$ where $n$ is the number of variables in $f$ and $g$. This algorithm works even when $f$ and $g$ are given by only black-box access: First, the truth-tables of the functions $f$ and $g$ can be computed in time $2^{O(n)}$. The truth tables for $f$ and $g$ can be seen as hypergraphs representing $f$ and $g$. Hypergraph Isomorphism for $n$-vertex and $m$-edge hypergraphs has a $2^{O(n)}m^{O(1)}$ algorithm due to Luks[9] which yields the claimed $2^{O(n)}$ time algorithm for testing if $f$ and $g$ are isomorphic. This is the current best known algorithm for general hypergraphs and hence the current best algorithm for Boolean function isomorphism as well. Indeed, a hypergraph on $n$ vertices and $m$ edges can be represented as a DNF formula on $n$ variables with $m$ terms. Thus, even when $f$ and $g$ are DNF formulas the best known isomorphism test takes $2^{O(n)}$ time. In contrast, Graph Isomorphism has a $2^{O(\sqrt{n \log n})}$ time algorithm due to Luks and Zemlyachenko (see [4]). More recently, Babai and Codenotti [3] have shown for hypergraphs of edge size bounded by $k$ that isomorphism testing can be done in $2^{\widetilde{O}(k^2\sqrt{n})}$ time.

## Our Results

Since the exact isomorphism problem for Boolean functions is as hard as Hypergraph Isomorphism, and it appears difficult to improve the $2^{O(n)}$ bound, we investigate the problem of computing *approximate* isomorphisms (which we define below). An interesting question is whether the *circuit complexity* of $f$ and $g$ can be exploited to give a faster *approximate* isomorphism test. Specifically, in this paper we study the approximation version of boolean function isomorphism for functions computed by *small size and small depth* circuits and give a faster algorithm for computing approximate isomorphisms. Before we explain our results we give some formal definitions.

Let $\mathcal{B}_n$ denote the set of all $n$-ary boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$. Let $g : \{0,1\}^n \rightarrow \{0,1\}$ be a boolean function and let $\pi : [n] \rightarrow [n]$ be any permutation. The Boolean function $g^\pi : \{0,1\}^n \rightarrow \{0,1\}$ obtained by applying the permutation $\pi$ to the function $g$ is defined as follows $g^\pi(x_1, x_2, \ldots, x_n) = g(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$.

This defines a (faithful) group action of the permutation group $S_n$ on the set $\mathcal{B}_n$. I.e. $g^{(\pi\psi)} = (g^\pi)^\psi$ for all $g \in \mathcal{B}_n$ and $\pi, \psi \in S_n$, and $g^\pi = g^\psi$ for all $g \in \mathcal{B}_n$ if and only if $\pi = \psi$.

**Definition 1.** *Two boolean functions $f, g \in \mathcal{B}_n$ are said to be isomorphic (denoted by $f \cong g$) if there exists a permutation $\pi : [n] \rightarrow [n]$ such that $\forall x \in \{0,1\}^n, f(x) = g^\pi(x)$.*

Our notion of approximate isomorphism of Boolean functions is based on the notion of *closeness* of Boolean functions which we now recall.

**Definition 2.** *Two boolean functions $f, g$ are $\frac{1}{2^\ell}$-close if* $\Pr_{x \in \{0,1\}^n}\big[f(x) \neq g(x)\big] \leq \frac{1}{2^\ell}$.

**Definition 3.** *Two boolean functions $f, g$ are $\frac{1}{2^\ell}$-approximate isomorphic if there exists a permutation $\pi : [n] \rightarrow [n]$ such that the functions $f$ and $g^\pi$ are $\frac{1}{2^\ell}$-close.*

Let $\mathcal{AC}_{s,d,n}$ denote the class of $n$-ary boolean functions computed by circuits of depth $d$ and size $s$, where the gates allowed are unbounded fan-in AND and OR gates, and negation gates. Suppose $f, g \in \mathcal{AC}_{s,d,n}$ are isomorphic boolean functions. As a consequence of the main result, in Section 2, we show that there is a randomized algorithm that computes a $\frac{1}{2^{\log^{O(1)} n}}$-approximate isomorphism between $f$ and $g$ in time $2^{\log(ns)^{O(d)}\sqrt{n}}$. This is substantially faster than the $2^{O(n)}$ time algorithm for computing an exact isomorphism. We show how to achieve this running time by combining some Fourier analysis of boolean functions with the Babai-Codenotti algorithm mentioned above.

We note that, in a different context, approximate Boolean function isomorphism has been studied in the framework of *property testing*, and nearly matching upper and lower bounds are known ([2],[7],[5]). In property testing the objective is to test whether two given Boolean functions are close to being isomorphic or far apart. The goal is to design a property test with low *query* complexity. In contrast, our result is algorithmic and the goal is to efficiently compute a good approximate isomorphism.

We also note that approximate versions of Graph Isomorphism have been studied in the literature as graph edit distance, graph similarity and graph matching with respect to various distance measures (e.g. [6]). There are various heuristic algorithms for the problem. These results do not appear related to the topic of our paper.

The rest of the paper is organized into four sections. In Section 2 we explain our approximate isomorphism algorithm for constant-depth small size Boolean circuits. Finally, in Section 3 we study a general problem: given two $n$-variable Boolean functions $f$ and $g$ consider the optimization problem where the objective is to find a permutation $\pi$ that maximizes $|\{x \in \{0,1\}^n \mid f(x) = g^\pi(x)\}|$. This problem is coNP-hard under Turing reductions. We give a simple $2^{O(n)}$ time deterministic approximation algorithm that, when given as input Boolean functions $f$ and $g$ such that $f$ and $g^\pi$ agree on a constant fraction of the inputs for some permutation $\pi$, outputs a permutation $\sigma$ such that $f$ and $g^\sigma$ agree on an $O(\frac{1}{\sqrt{n}})$ fraction of the inputs.

## 2 Main Result

In this section we focus on the problem of computing an approximate isomorphism for two Boolean functions $f, g \in \mathcal{AC}_{s,d,n}$. We first recall some Fourier analysis of Boolean functions which is an important ingredient in our algorithm. For Fourier analytic purposes, it is convenient for us to consider Boolean functions with domain $\{-1,1\}^n$ and range $\{-1,1\}$. The range $\{-1,1\}$ makes it convenient to define the Fourier basis.

The set $\mathcal{F} = \{f : \{-1,1\}^n \to \mathbb{R}\}$ of real-valued functions forms a $2^n$-dimensional vector space over $\mathbb{R}$, where vector addition is defined as $(f+g)(x) = f(x)+g(x)$. The vector space $\mathcal{F}$ forms an inner product space with inner product defined as:

$$\langle f, g\rangle = \mathbb{E}_{x\in\{-1,1\}^n}[f(x)g(x)] = \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} f(x)g(x).$$

The $\ell_2$-norm of a function $f \in \mathcal{F}$ is $\|f\|_2 = \sqrt{\langle f, f\rangle}$. Clearly every Boolean function $f : \{-1,1\}^n \to \{-1,1\}$ has unit norm under this inner product.

The *Fourier basis*, $\{\chi_S | S \subseteq [n]\}$ is defined as $\chi_S(x) = \prod_{i\in S} x_i$. It is easy to observe that $\mathbb{E}_x[\chi_S(x)] = 0$ for nonempty sets $S$ and $\mathbb{E}_x[\chi_\emptyset(x)] = 1$. Furthermore, $\langle \chi_S, \chi_T\rangle = \mathbb{E}_x[\chi_{S\triangle T}(x)]$. It follows that the Fourier basis is an orthonormal basis with respect to the inner product. Thus, any $f \in \mathcal{F}$ can be written as $f = \sum \widehat{f_S}\chi_S$. This is the *Fourier representation* of $f$, and the numbers $\widehat{f_S} = \langle f, \chi_S\rangle$ are the *Fourier coefficients* of $f$. The orthonormality of the Fourier basis yields *Parseval's identity:* $\langle f, f\rangle = \sum_{S\subseteq[n]} \widehat{f}(S)^2$. In particular, since any Boolean function $f : \{-1, 1\}^n \to \{-1.1\}$ has unit norm, we note that $\sum_{S\subseteq[n]} \widehat{f}(S)^2 = 1$.

In the next two propositions we relate the isomorphism of Boolean functions $f$ and $g$ to their Fourier coefficients.

**Proposition 1.** *Let $\pi : [n] \to [n]$ be any permutation, $g$ any Boolean function and $S \subseteq [n]$, then $\widehat{g^\pi}(S) = \widehat{g}(S^\pi)$ where $S^\pi = \{i | \pi(i) \in S\}$.*

*Proof.* $\widehat{g^\pi}(S) = \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} g^\pi(x)\chi_S(x) = \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} g(x)\chi_{S^\pi}(x) = \widehat{g}(S^\pi)$. □

**Proposition 2.** *Two Boolean functions $f, g : \{-1.1\}^n \to \{-1, 1\}$ are isomorphic via permutation $\pi$ if and only if $\widehat{f}(S) = \widehat{g}(S^\pi)$ for each subset $S$.*

*Proof.* Suppose $\pi : [n] \to [n]$ is an isomorphism. I.e. $f(x) = g^\pi(x)$ for all $x \in \{-1, 1\}^n$. Consider any subset $S \subseteq [n]$

$$\widehat{f}(S) = \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} f(x)\chi_S(x) = \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} g^\pi(x)\chi_S(x) = \widehat{g^\pi}(S) = \widehat{g}(S^\pi).$$

Conversely, if $\widehat{f}(S) = \widehat{g}(S^\pi)$ for each subset $S$, by the previous proposition we have $\widehat{f}(S) = \widehat{g^\pi}(S)$ which implies that $f = g^\pi$. □

## 2.1 Approximate Isomorphism for $\mathcal{AC}_{s,d,n}$

Now we turn to isomorphism for Boolean functions from the class $\mathcal{AC}_{s,d,n}$. We first outline our approach to Boolean function isomorphism via Fourier coefficients.

A crucial theorem that we will use is the celebrated result of Linial-Mansour-Nisan [8] which gives the distribution of Fourier coefficients for Boolean functions computed by small depth circuits.

**Theorem 1 ([8]).** *Let $f : \{-1, 1\}^n \to \{-1, 1\}$ be computed by an $\mathcal{AC}_{s,d,n}$ circuit. Then for all $t > 0$,*

$$\sum_{S\subseteq[n],|S|>t} \widehat{f}(S)^2 \le 2s2^{-t^{1/d}/20}.$$

*Consequently, for $\widetilde{f} = \sum_{S\subseteq[n],|S|\le t} \widehat{f}(S)\chi_S$ we have $\|f - \widetilde{f}\|_2^2 \le 2s2^{-t^{1/d}/20}$.*

Notice that each $\chi_S = \prod_{i \in S} x_i$ is a monomial, and hence $\widetilde{f}$ is a degree-$t$ polynomial that approximating $f$. Given Boolean functions $f, g \in \mathcal{AC}_{s,d,n}$ as an instance of Boolean function isomorphism, our aim is to work with the polynomials $\widetilde{f}$ and $\widetilde{g}$:

$$\widetilde{f} = \sum_{S \subseteq [n], |S| \leq t} \widehat{f}(S)\chi_S \quad \text{and} \quad \widetilde{g} = \sum_{S \subseteq [n], |S| \leq t} \widehat{g}(S)\chi_S. \tag{1}$$

This is because $\widetilde{f}$ and $\widetilde{g}$ are of degree $t$ and have only $n^t$ terms. I.e. we will check if there is an approximate isomorphism between $\widetilde{f}$ and $\widetilde{g}$. Notice that the polynomials $\widetilde{f}, \widetilde{g} : \{-1,1\}^n \to \mathbb{R}$ are not Boolean-valued functions. We need an appropriate notion of isomorphism here.

**Definition 4.** *Let $f', g' : \{-1,1\}^n \to \mathbb{R}$ be two functions from $\mathcal{F}$. We say that $f'$ and $g'$ are $\frac{1}{2^\ell}$-approximate isomorphic witnessed by a permutation $\pi : [n] \to [n]$ if $\|f' - g'^\pi\|_2^2 \leq \frac{1}{2^\ell}$.*

We now explain the connection between $\frac{1}{2^\ell}$-approximate isomorphism of two functions and their Fourier coefficients.

**Proposition 3.** *If $f, g : \{-1,1\}^n \to \{-1,1\}$ are $\frac{1}{2^\ell}$-close, then $\|f - g\|_2^2 \leq 4\frac{1}{2^\ell}$*

*Proof.* Follows from $\|f - g\|^2 = \mathbb{E}[(f-g)^2] = 4\Pr[f \neq g]$. $\square$

**Lemma 1.** *Let $f$ and $g$ be two Boolean functions that are $\frac{1}{2^\ell}$-approximate isomorphic via permutation $\pi : [n] \to [n]$. Then $\forall S \subseteq [n]$ : $\left|\widehat{f}(S) - \widehat{g^\pi}(S)\right| \leq \frac{2}{2^{\ell/2}}$.*

*Proof.* Notice that $\sum_{S \subseteq [n]} (\widehat{f}(S) - \widehat{g^\pi}(S))^2 = \|f - g^\pi\|_2^2$. Suppose $f, g$ are $\frac{1}{2^\ell}$-approximate isomorphic via permutation $\pi$. By Proposition 3 we know that $\sum_{S \subseteq [n]} (\widehat{f}(S) - \widehat{g^\pi}(S))^2 = \|f - g^\pi\|_2^2 \leq \frac{4}{2^\ell}$. Hence for each subset $S \subseteq [n]$ we have $\left(\widehat{f}(S) - \widehat{g^\pi}(S)\right)^2 \leq \frac{4}{2^\ell}$. $\square$

Suppose $f$ and $g$ be two Boolean functions that are $\frac{1}{2^\ell}$-approximate isomorphic via permutation $\pi : [n] \to [n]$. By the above proposition $|\widehat{f}(S) - \widehat{g^\pi}(S)|$ is bounded by $\frac{2}{2^{\ell/2}}$. Furthermore, since both $\widehat{f}(S)$ and $\widehat{g^\pi}(S)$ are Fourier coefficients of Boolean functions $f$ and $g^\pi$, we have $0 \leq |\widehat{f}(S)| \leq 1$ and $0 \leq |\widehat{g^\pi}(S)| \leq 1$. Hence, the bound implies that the $\lfloor \ell/2 \rfloor - 1$ most significant positions in the binary representation of $\widehat{f}(S)$ and $\widehat{g^\pi}(S)$ are identical.

For each subset $S$, let $\widehat{f_\ell}(S)$ denote the truncation of $\widehat{f}(S)$ to the first $\lfloor \ell/2 \rfloor - 1$ bits. Thus, $|\widehat{f_\ell}(S) - \widehat{f}(S)| \leq \frac{1}{2^{\lfloor \ell/2 \rfloor - 1}}$ for each $S$. Similarly, $\widehat{g_\ell}(S)$ denotes the truncation of $\widehat{g}(S)$ to the first $\lfloor \ell/2 \rfloor - 1$ bits. We define the following two functions $f_\ell$ and $g_\ell$ from $\{-1,1\}^n \to \mathbb{R}$:

$$f_\ell = \sum_{S \subseteq [n]} \widehat{f_\ell}(S)\chi_S \quad \text{and} \quad g_\ell = \sum_{S \subseteq [n]} \widehat{g_\ell}(S)\chi_S. \tag{2}$$

The following lemma summarizes the above discussion. It gives us a way to go from approximate isomorphism to exact isomorphism.

**Lemma 2.** *If $f$ and $g$ be two Boolean functions that are $\frac{1}{2^\ell}$-approximate isomorphic via permutation $\pi : [n] \to [n]$ then $f_\ell = g_\ell^\pi$, i.e. the functions $f_\ell$ and $g_\ell$ are (exactly) isomorphic via the permutation $\pi$.*

Lemma 1 and Proposition 3 yield the following observation.

**Lemma 3.** *Suppose $f, g$ are two Boolean functions that are $\frac{1}{2^\ell}$-approximate isomorphic via permutation $\pi$. Then $\|\widetilde{f} - \widetilde{g^\pi}\|_2^2 \leq \frac{4}{2^\ell}$. I.e. $\widetilde{f}$ and $\widetilde{g}$ are $\frac{4}{2^\ell}$-approximate isomorphic via the same permutation $\pi$. Furthermore, $|\widehat{f}(S) - \widehat{g^\pi}(S)| \leq \frac{2}{2^{\ell/2}}$ for all $S : |S| \leq t$.*

*Proof.* By Lemma 1 and Proposition 3 we have $\sum_{S \subseteq [n]} \big(\widehat{f}(S) - \widehat{g^\pi}(S)\big)^2 \leq \frac{4}{2^\ell}$, which implies $\|\widetilde{f} - \widetilde{g^\pi}\|_2^2 = \sum_{|S| \leq t} \big(\widehat{f}(S) - \widehat{g^\pi}(S)\big)^2 \leq \frac{4}{2^\ell}$. It follows that $|\widehat{f}(S) - \widehat{g^\pi}(S)| \leq \frac{2}{2^{\ell/2}}$ for all $S : |S| \leq t$. □

Now, if $|\widehat{f}(S) - \widehat{g^\pi}(S)| \leq \frac{2}{2^{\ell/2}}$ for all $S : |S| \leq t$, it implies that $\widehat{f_\ell}(S) = \widehat{g_\ell^\pi}(S)$ for all $S \ : |S| \leq t$, where $\widehat{f_\ell}(S)$ and $\widehat{g_\ell}(S)$ are defined in Equation 2. Indeed, if we truncate the coefficients of the polynomials $\widetilde{f}$ and $\widetilde{g}$ also to the first $\lfloor \ell/2 \rfloor - 1$ bits we obtain the polynomials

$$\widetilde{f_\ell} = \sum_{S:|S| \leq t} \widehat{f_\ell}(S)\chi_S \quad \text{and} \quad \widetilde{g_\ell} = \sum_{S:|S| \leq t} \widehat{g_\ell}(S)\chi_S. \tag{3}$$

It clearly follows that $\pi$ is an exact isomorphism between $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$. We summarize the above discussion in the following lemma which is crucial for our algorithm.

**Lemma 4.** *Suppose $f, g$ are two Boolean functions that are $\frac{1}{2^\ell}$-approximate isomorphic via permutation $\pi$. Then:*

1. *$\|\widetilde{f} - \widetilde{g^\pi}\|_2^2 \leq \frac{4}{2^\ell}$. I.e. $\widetilde{f}$ and $\widetilde{g}$ are $\frac{4}{2^\ell}$-approximate isomorphic via the same permutation $\pi$, and hence $|\widehat{f}(S) - \widehat{g^\pi}(S)| \leq \frac{2}{2^{\ell/2}}$ for all $S : |S| \leq t$.*
2. *Consequently, $\pi$ is an exact isomorphism between $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$.*

We can represent $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$ as weighted hypergraphs with hyperedges $S : |S| \leq t$ of weight $\widehat{f_\ell}(S)$ and $\widehat{g_\ell}(S)$ respectively. We can then apply the Babai-Codenotti hypergraph isomorphism algorithm [3] to compute an exact isomorphism $\pi$ between $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$. Now, if $\pi$ is an exact isomorphism $\pi$ between $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$ what can we infer about $\pi$ as an approximate isomorphism between $f$ and $g$? The following lemma quantifies it.

**Lemma 5.** *Suppose $f$ and $g$ are Boolean functions in $\mathcal{AC}_{s,d,n}$ such that $\pi$ is an exact isomorphism between $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$ (where $\widetilde{f}$ and $\widetilde{g}$ are given by Equation 1). Then $\pi$ is an $(\delta + \varepsilon)^2$-approximate isomorphism between $f$ and $g$, where $\delta = 2s2^{-t^{1/d}/20}$ and $\varepsilon = \frac{2n^{t/2}}{2^{(\ell-1)/2}}$.*

*Proof.* Since $\pi$ is an exact isomorphism between $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$ we have $\widetilde{f_\ell} = \widetilde{g_\ell^\pi}$. Now consider $\|f - g^\pi\|_2^2$. By triangle inequality

$$\begin{aligned}\|f - g^\pi\|_2 &\le \|f - \widetilde{f}\| + \|\widetilde{f} - \widetilde{f_\ell}\| + \|\widetilde{f_\ell} - \widetilde{g^\pi}_\ell\| + \|\widetilde{g^\pi} - \widetilde{g^\pi}_\ell\| + \|g^\pi - \widetilde{g^\pi}\| \\ &= \|f - \widetilde{f}\| + \|\widetilde{f} - \widetilde{f_\ell}\| + \|\widetilde{g^\pi} - \widetilde{g^\pi}_\ell\| + \|g^\pi - \widetilde{g^\pi}\|.\end{aligned}$$

By Theorem 1, both $\|f - \widetilde{f}\|$ and $\|g^\pi - \widetilde{g^\pi}\|$ are bounded by $\delta$. Furthermore,

$$\|\widetilde{f} - \widetilde{f_\ell}\|_2^2 = \sum_{S:|S|\le t} (\widehat{f}(S) - \widehat{f}_\ell(S))^2 \le \sum_{S:|S|\le t} \frac{4}{2^{\ell-1}} \le \frac{4n^t}{2^{\ell-1}}.$$

Hence, $\|\widetilde{f} - \widetilde{f_\ell}\| \le \frac{2n^{t/2}}{2^{(\ell-1)/2}}$ and, likewise, $\|\widetilde{g^\pi} - \widetilde{g^\pi}_\ell\| \le \frac{2n^{t/2}}{2^{(\ell-1)/2}}$. Putting it together with Proposition 3 we get

$$4\Pr[f \ne g^\pi] = \|f - g^\pi\|_2^2 \le \left(2\delta + \frac{4n^{t/2}}{2^{(\ell-1)/2}}\right)^2.$$

It follows that $f$ and $g$ are $(\delta + \varepsilon)^2$-approximate isomorphic via the permutation $\pi$. □

Suppose $f$ and $g$ are in $\mathcal{AC}_{s,d,n}$ and are given by circuits $C_f$ and $C_g$. Our goal now is to design an efficient algorithm that will compute the polynomials $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$, where $\ell$ will be appropriately chosen in the analysis. In order to compute $\widetilde{f_\ell}$ and $\widetilde{g_\ell}$ we need to estimate to $\lfloor \ell/2 \rfloor - 1$ bits of precision the Fourier coefficients $\widehat{f}(S)$ and $\widehat{g}(S)$ for each subset $S : |S| \le t$. Now, by definition, $\widehat{f}(S)$ is the average of $f(x)\chi_S(x)$ where $x$ is uniformly distributed in $\{-1,1\}^n$. Hence, following a standard Monte-Carlo sampling procedure, we can estimate $\widehat{f}(S)$ quite accurately from a random sample of inputs from $\{-1,1\}^n$ and hence with high probability we can exactly compute $f_\ell(S)$ for all $S : |S| \le t$. We formally explain this in the next lemma.

**Lemma 6.** *Given* $f : \{-1,1\}^n \to \{-1,1\}$ *computed by an* $\mathcal{AC}_{s,d,n}$ *circuit, there is a randomized algorithm* $\mathcal{C}$ *with running time* $\mathrm{poly}(s, n^t, 2^\ell)$ *that outputs the set* $\{f_\ell(S) \mid |S| \le t\}$ *with probability* $1 - \frac{1}{2^{\Omega(n)}}$.

*Proof.* We use the same technique as [8]to estimate the required Fourier coefficients.

1. For each subset $S \subset [n]$ such that $|S| \le t$ do the following two steps:
2. Pick $x_i \in_r \{-1,1\}^n$ and compute the value $f(x_i)\chi_S(x_i)$ for $i \in [m]$.
3. Estimate the Fourier coefficient as $\alpha_f(S) = \frac{1}{m}\sum_{i=1}^m f(x_i)\chi_S(x_i)$.

Applying Chernoff bounds, for each subset $S$ we have $\Pr\left[\left|\widehat{f}(S) - \alpha_f(S)\right| \ge \lambda\right] \le 2e^{-\lambda^2 m/2}$. In our case we set $\lambda = \frac{1}{2^{\lfloor \ell/2 \rfloor - 1}}$. In order to estimate $\widehat{f}(S)$ for each $S : |S| \le t$ within the prescribed accuracy and with small error probability, we set $m = tn \log n 2^\ell$. The entire procedure runs in $\mathrm{poly}(s, n^t, 2^\ell)$ time. Furthermore, by a simple union bound it follows that with probability $1 - 2^{-\Omega(n)}$ we have $\alpha_f(S) = f_\ell(S)$ for each $S : |S| \le t$ with probability. Thus, the randomized algorithm computes the polynomial $\widetilde{f_\ell}$ with high probability. □

## 2.2 Exact Isomorphism Test for Low Degree Polynomials

We now focus on the problem of checking if the polynomials $\widetilde{f_\ell} = \sum_{S:|S|\leq t} \widehat{f_\ell}(S)\chi_S$ and $\widetilde{g_\ell} = \sum_{S:|S|\leq t} \widehat{g_\ell}(S)\chi_S$ are isomorphic, and if so to compute an exact isomorphism $\pi$. To this end, we shall encode $f_\ell$ and $g_\ell$ as *weighted* hypergraphs $G_f$ and $G_g$, respectively.

The vertex sets for both graphs is $[n]$. Let $E$ denote the set of all subsets $S \subset [n]$ of size at most $t$. The weight functions for the edges are $w_f$ and $w_g$ for $G_f$ and $G_g$ defined as follows

$$w_f(S) = \begin{cases} \alpha_f(S) & \forall S \subseteq [n], |S| \leq t \\ 0 & otherwise, \end{cases} \quad \text{and} \quad w_g(S) = \begin{cases} \alpha_g(S) & \forall S \subseteq [n], |S| \leq t \\ 0 & otherwise. \end{cases}$$

The isomorphism problem for the polynomials the $f_\ell$ and $g_\ell$ is now the edge-weighted hypergraph isomorphism problem, where $G_f$ and $G_g$ are the two edge-weighted graphs, and the problem is to compute a permutation on $[n]$ that maps edges to edges (preserving edge weights) and non-edges to non-edges. Our aim is to apply the Babai-Codenotti isomorphism algorithm for hypergraphs with hyperedge size bounded by $k$ [3]. Their algorithm has running time $2^{\widetilde{O}(k^2\sqrt{n})}$. We need to adapt their algorithm to work for hypergraphs with edge weights. Since the edge weights for the graphs $G_f$ and $G_g$ are essentially $\lfloor \ell/2 \rfloor - 1$ bit strings, we can encode the weights into the hyperedges by introducing new vertices.

More precisely, we create new graphs $G'_f$ and $G'_g$ corresponding to $f$ and $g$, where the number of vertices is now $n + O(\ell)$. Let the set of new vertices be $\{v_1, \ldots, v_r\}$, where $r = O(\ell)$. Let $S \subset [n]$ be a hyperedge in the original graph $G_f$. A subset $T \subset \{v_1, \ldots, v_r\}$ encodes an $r$-bit string via a natural bijection (the $j^{th}$ bit is 1 if and only if $v_j \in T$). Let $T(S) \subset \{v_1, \ldots, v_r\}$ denote the encoding of the number $\widehat{f_\ell}(S)$ for each hyperedge $S \in E$. Similarly, let $T'(S) \subset \{v_1, \ldots, v_r\}$ denote the encoding of the number $\widehat{g_\ell}(S)$. The hyperedge $S \cup T(S)$ encodes $S$ along with its weight $\widehat{f_\ell}(S)$ for each $S$ in $G'_f$. Similarly, $S \cup T'(S)$ encodes $S$ along with its weight $\widehat{g_\ell}(S)$ for each $S$ in $G'_g$. As candidate isomorphisms between $G'_f$ and $G'_g$ we wish to consider only permutations on $[n] \cup \{v_1, \ldots, v_r\}$ that fix each $v_i, 1 \leq i \leq r$. This can be easily ensured by standard tricks like coloring each $v_i$ with a unique color, where the different colors can be implemented by putting directed paths of different lengths, suitably long so that the vertices $v_i$ are forced to be fixed by any isomorphism between $G'_f$ and $G'_g$.

This will ensure that $G'_f$ and $G'_g$ are isomorphic iff there is a weight preserving isomorphism between $G_f$ and $G_g$. Now we invoke the algorithm of [3] on $G'_f$ and $G'_g$ which will yield an isomorphism $\psi$ between $f'$ and $g'$. In summary, the algorithm for isomorphism testing $f_\ell$ and $g_\ell$ carries out the following steps.

**Isomorphism Test for Polynomials**

1. Construct the hypergraphs $G'_f$ and $G'_g$ as defined above.
2. Run the algorithm of Babai and Codenotti[3] on the hypergraphs $G'_f$ and $G'_g$ and output isomorphism $\psi$ or report they are non-isomorphic.

**Lemma 7.** *The isomorphism of polynomials $\widetilde{f}_\ell$ and $\widetilde{g}_\ell$ (defined by Equation 2) can be tested in time $2^{O(\sqrt{n})(\ell+t)^2 \log^{O(1)} n}$, and if the polynomials are isomorphic an exact isomorphism can be computed in the same running time bound.*

## 2.3 The Approximate Isomorphism Algorithm

We now give an outline of the entire algorithm.

**Input**: $f, g \in \mathcal{AC}_{s,d,n}$ given by circuits of size $s$ along with parameters $t$ and $\ell$.
**Step 1.** Compute the polynomials $\widetilde{f}_\ell$ and $\widetilde{g}_\ell$ using randomized algorithm of Lemma 6.
**Step 2.** Check if $\widetilde{f}_\ell$ and $\widetilde{g}_\ell$ are isomorphic using the polynomial isomorphism algorithm described above. If they are not isomorphic *reject* else **output** the computed exact isomorphism $\pi$.

Suppose $\pi$ is an exact isomorphism between $\widetilde{f}_\ell$ and $\widetilde{g}_\ell$ computed by the above algorithm. By Lemma 5 $\pi$ is a $(\delta+\varepsilon)^2$-approximate isomorphism between $f$ and $g$, where $\delta = 2s2^{-t^{1/d}/20}$ and $\varepsilon = \frac{2n^{t/2}}{2^{(\ell-1)/2}}$. From Lemmas 6 and 7 it follows that the overall running time of the algorithm is $\mathrm{poly}(s, n^t, 2^\ell) + 2^{O(\sqrt{n})(\ell+t)^2 \log^{O(1)} n}$ and the error probability, as argued in Lemma 6, is at most $2^{-\Omega(n)}$.

We now set parameters to obtain the main result of the paper. Suppose $f$ and $g$ are $\frac{1}{2^\ell}$-approximate isomorphic, where $\ell = (\log n + \log s)^{kd}$ for a suitably large constant $k > 1$. Then we choose $t = (\log n + \log s)^{O(d)}$ so that $(\delta+\varepsilon)^2$ is bounded by $2^{-(\log n)^{O(1)}}$.

**Theorem 2.** *Given two Boolean functions $f, g \in \mathcal{AC}_{s,d,n}$ which are $\frac{1}{2^{(\log n)^{O(d)}}}$-isomorphic, there is a randomized algorithm running in time $2^{O(\log^{O(d)}(n)\sqrt{n})}$ to compute a permutation $\pi$ such that $f, g$ are $\frac{1}{2^{(\log n)^{O(1)}}}$-approximate isomorphic with respect to $\pi$.*

## 3 A General Approximate Isomorphism Algorithm

Given two $n$-variable Boolean functions $f$ and $g$ (either by Boolean circuits computing them or just by black-box access) consider the optimization problem of finding a permutation $\pi$ that minimizes $|\{x \in \{0,1\}^n \mid f(x) \neq g^\pi(x)\}|$. This problem is coNP-hard under Turing reductions. We reduce the coNP-complete problem TAUTOLOGY (checking if a propositional formula is a tautology) to the problem MinBooleanIso of computing a permutation $\pi$ that minimizes $|\{x \in \{0,1\}^n \mid f(x) \neq g^\pi(x)\}|$.

**Lemma 8.** TAUTOLOGY *is polynomial-time Turing reducible to* MinBooleanIso.

*Proof.* Given $f : \{0,1\}^n \to \{0,1\}$ as an $n$-variable propositional formula, we define functions $g_i : \{0,1\}^n \to \{0,1\}$ for $i \in [n]$ such that $g_i(1^i 0^{n-i}) = 0$

and $g_i(x) = 1$ for all $x \neq 1^i 0^{n-i}$. Notice that if $f$ is a tautology then for each $i$ $|\{x \in \{0,1\}^n \mid f(x) \neq g_i^\pi(x)\}| = 1$ for all permutations $\pi$.

We now describe a polynomial-time algorithm for TAUTOLOGY with MinBooleanIso as oracle. For each $g_i$, we compute (with a query to the function oracle MinBooleanIso) a permutation $\pi_i$ that minimizes $|\{x \in \{0,1\}^n \mid f(x) \neq g_i^\pi(x)\}|$. If $f(\pi_i^{-1}(1^i 0^{n-i})) = 1$ for each $i$, the algorithm describing the Turing reduction "accepts" $f$ as a tautology and otherwise it "rejects" $f$.

We now show the correctness of the reduction. If $f$ is a tautology, then clearly for each $\pi_i$ we have $f(\pi_i^{-1}(1^i 0^{n-i})) = 1$. Conversely, suppose $f$ is not a tautology. Then $f^{-1}(0) = \{x \in \{0,1\}^n \mid f(x) = 0\}$ is nonempty. Let $|f^{-1}(0)| = N$. Then for any permutation $\pi$ the cardinality $|\{x \in \{0,1\}^n \mid f(x) \neq g_i^\pi(x)\}|$ is either $N+1$ or $N-1$ for each $i$. Furthermore, suppose $x \in f^{-1}(0)$ has Hamming weight $i$. Then for any permutation $\pi_i$ that maps $x$ to $1^i 0^{n-i}$ we have $|\{x \in \{0,1\}^n \mid f(x) \neq g_i^{\pi_i}(x)\}| = N-1$. Hence, $f(\pi_i^{-1}(1^i 0^{n-i})) = f(x) = 0$. □

A brute-force search that runs in $n!$ time by cycling through all permutations yields a trivial algorithm for the optimization problem MinBooleanIso.

The corresponding *maximization problem* is: Find $\pi$ that maximizes $|\{x \in \{0,1\}^n \mid f(x) = g^\pi(x)\}|$. Of course computing an optimal solution to this problem is polynomial-time equivalent to MinBooleanIso. In remainder of this section we design a simple approximate isomorphism algorithm for the *maximization* problem. Our simple algorithm is based on the method of conditional probabilities. We first examine how good a random permutation is as an approximate isomorphism. Then we describe a deterministic algorithm for computing a permutation with the same solution quality.

For Boolean functions $f$ and $g$, consider the random variable $|\{x|f(x) = g^\pi(x)\}|$ when the permutation $\pi$ is picked uniformly at random from $S_n$.

Let $s_i(f)$ denote the cardinality $|\{x \in \{0,1\}^n| \operatorname{wt}(x) = i, f(x) = 1\}|$ where $\operatorname{wt}(x)$ is the hamming weight of the Boolean string $x$. Clearly, $s_i(f) \leq \binom{n}{i}$. For each $u \in \{0,1\}^n$ define the 0-1 random variable $X_u$ which takes value 1 if and only if $f(u) = g^\pi(u)$ for $\pi \in S_n$ picked uniformly at random. If $\operatorname{wt}(u) = i$, then

$$\Pr_\pi[X_u = 1] = \frac{s_i(g)}{\binom{n}{i}} f(u) + \frac{\binom{n}{i} - s_i(g)}{\binom{n}{i}} (1 - f(u)).$$

The sum $X = \sum_{u \in \{0,1\}^n} X_u$ is the random variable $|\{x|f(x) = g^\pi(x)\}|$ for a random permutation $\pi \in S_n$. We have

$$\mathbb{E}_\pi[X] = \sum_{i=0}^{n} \sum_{u \,:\, \operatorname{wt}(u)=i} \frac{s_i(g)}{\binom{n}{i}} f(u) + \sum_{i=0}^{n} \sum_{u \,:\, \operatorname{wt}(u)=i} \frac{\binom{n}{i} - s_i(g)}{\binom{n}{i}} (1 - f(u)) \quad (4)$$

$$= \sum_{i=0}^{n} \frac{s_i(g) s_i(f)}{\binom{n}{i}} + \sum_{i=0}^{n} \frac{(\binom{n}{i} - s_i(g))(\binom{n}{i} - s_i(f))}{\binom{n}{i}} \quad (5)$$

$$\geq \max_i \left( \frac{s_i(f) s_i(g)}{\binom{n}{i}}, \frac{(\binom{n}{i} - s_i(g))(\binom{n}{i} - s_i(f))}{\binom{n}{i}} \right). \quad (6)$$

**Theorem 3.** *There is a deterministic $2^{O(n)}$ time algorithm that takes as input Boolean functions $f, g : \{0,1\}^n \to \{0,1\}$ as input (either by Boolean circuits or by black-box access) and outputs a permutation $\sigma$ with the following property: If $f$ and $g^\pi$ are $\delta$-close for some permutation $\pi$ and constant $\delta$, then*

$$|\{x|f(x) = g^\sigma(x)\}| \geq \Omega\left(\frac{2^n}{\sqrt{n}}\right).$$

*Proof.* First we show how to compute a permutation $\sigma$ such that $|\{x|f(x) = g^\sigma(x)\}| \geq \mathbb{E}_\pi[X]$ (see Equation 4 and discussion preceding it for the definition of random variable $X$). Firstly, notice that given a partial permutation $\sigma_i$ defined on $\{1, 2, \ldots, i\}$, we can define random variables $X_{\sigma_i,u}$ for each $u \in \{0,1\}^n$ and $X_{\sigma_i} = \sum_u X_{\sigma_i,u}$, defined by a uniformly picked random permutation $\pi$ in $S_n$ that extends $\sigma_i$. Similar to Equation 4, we can write an expression for $\mathbb{E}[X_{\sigma_i}]$ and compute it exactly in time $2^{O(n)}$ for a given $\sigma_i$. For $j \in [n] \setminus \{\sigma_i(1), \sigma_i(2), \ldots, \sigma_i(i)\}$ let $\sigma_{i,j}$ denote the extension of $\sigma_i$ that maps $i+1$ to $j$. In time $2^{O(n)}$ we can compute $\mathbb{E}[X_{\sigma_{i,j}}]$ for every $j$, and choose the permutation $\sigma_{i+1}$ as that $\sigma_{i,j}$ which maximizes $\mathbb{E}[X_{\sigma_{i,j}}]$. In particular, this will satisfy $\mathbb{E}[X_{\sigma_{i+1}}] \geq \mathbb{E}[X_{\sigma_i}]$. Continuing this process until $i = n$ yields $\sigma_n = \sigma$ such that $|\{x \mid f(x) = g^\sigma(x)\}| \geq \mathbb{E}_\pi[X]$, where $\pi$ is randomly picked from $S_n$.

Now, consider the expected value $\mathbb{E}_\pi(X)$. It is promised that for some permutation $\tau \in S_n$ the fraction $\delta = \max_{\sigma \in S_n} |\{x|f(x) = g^\sigma(x)\}|/2^n$ is a constant (independent of $n$). For $0 \leq i \leq n$ let

$$\delta_i = \frac{|\{x \mid f(x) = g^\tau(x), \mathrm{wt}(x) = i\}|}{\binom{n}{i}}.$$

Thus $\sum_{i=0}^n \delta_i \binom{n}{i} = \delta 2^n$ which we can write as

$$\sum_{i=0}^{\sqrt{n}} (\delta_i + \delta_{n-\sqrt{n}+i}) \binom{n}{i} + \sum_{i=n/2-\sqrt{n}}^{n/2+\sqrt{n}} \delta_i \binom{n}{i} = \delta 2^n.$$

Since each $\delta_i \leq 1$, $\sum_{i=0}^{\sqrt{n}} (\delta_i + \delta_{n-\sqrt{n}+i}) \binom{n}{i} \leq 2n^{\sqrt{n}+1} \leq 2^{2\sqrt{n}\log n}$ for sufficiently large $n$.

Let $A$ denote the sum $\sum_{i=n/2-\sqrt{n}}^{n/2+\sqrt{n}} \delta_i \binom{n}{i}$. Then

$$A \geq \delta 2^n \left(1 - \frac{2^{2\sqrt{n}\log n}}{\delta 2^n}\right) \geq \frac{\delta}{2} 2^n.$$

By averaging, there is some hamming weight $i$ in the range $n/2 - \sqrt{n} \leq i \leq n/2 + \sqrt{n}$, such that

$$\delta_i \binom{n}{i} = |\{u \mid \mathrm{wt}(u) = i \text{ and } f(u) = g^\pi(u)\}| \geq \frac{\delta 2^n}{4\sqrt{n}}.$$

We fix this value of $i$ and let $S$ denote the set $\{u \mid \mathrm{wt}(u) = i \text{ and } f(u) = g^\pi(u)\}$. Assume without loss of generality that $|f^{-1}(1) \cap S| > \frac{\delta 2^n}{8\sqrt{n}}$ (Otherwise we consider $f^{-1}(0) \cap S$). Thus, we have $s_i(f) \geq |f^{-1}(1) \cap S| = |(g^\pi)^{-1}(1) \cap S| \geq \frac{\delta 2^n}{8\sqrt{n}}$.

Now, $\{u \mid \mathrm{wt}(u) = i \text{ and } g^\pi(u) = 1\} \supseteq (g^\pi)^{-1}(1) \cap S$. Hence $|(g^\pi)^{-1}(1) \cap S| \leq |\{u \mid \mathrm{wt}(u) = i \text{ and } g^\pi(u) = 1\}| = |\{u \mid \mathrm{wt}(u) = i \text{ and } g(u) = 1\}| = s_i(g)$. Combined with Equation 4 and using the inequality $\binom{n}{i} \leq \frac{2^n}{\sqrt{n}}$ for large enough $n$, we get the desired lower bound on $\mathbb{E}[X]$:

$$\mathbb{E}[X] \geq \frac{s_i(f)s_i(g)}{\binom{n}{i}} \geq \frac{\delta^2 2^{2n}}{64n\binom{n}{i}} \geq \frac{\delta^2 2^n}{64\sqrt{n}} = \Omega\left(\frac{2^n}{\sqrt{n}}\right).$$

□

**Concluding Remarks.** Motivated by the question whether Boolean function isomorphism testing has algorithms faster than Luks's $2^{O(n)}$ time algorithm [9], we initiate the study of approximate Boolean function isomorphism. As our main result we show a substantially faster algorithm that for Boolean functions having small depth and small size circuits computes an approximate isomorphism. Precisely characterizing the approximation threshold for this problem for various Boolean function classes based on their circuit complexity is an interesting direction of research.

**Acknowledgment** We are grateful to Johannes Köbler and Sebastian Kuhnert for discussions on the topic, especially for their help with Lemma 8.

## References

1. Agrawal, M., Thierauf, T.: The formula isomorphism problem. SIAM Journal on Computing 30(3), 990–1009 (2000)
2. Alon, N., Blais, E.: Testing Boolean Function Isomorphism. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX and RANDOM 2010. LNCS, vol. 6302, pp. 394–405. Springer, Heidelberg (2010)
3. Babai, L., Codenotti, P.: Isomorphism of hypergraphs of low rank in moderately exponential time. In: 49th FOCS, pp. 667–676. IEEE (2008)
4. Babai, L., Luks, E.M.: Canonical labeling of graphs. In: 15th STOC, pp. 171–183. ACM (1983)
5. Blais, E., O'Donnell, R.: Lower bounds for testing function isomorphism. In: 25th CCC, pp. 235–246. IEEE Computer Society (2010)
6. Bunke, H.: Graph matching: Theoretical foundations, algorithms, and applications. In: Proceedings of Vision Interface, Montreal, Canada, pp. 82–88 (2000)
7. Chakraborty, S., García-Soriano, D., Matsliah, A.: Nearly tight bounds for testing function isomorphism. In: 22nd SODA, pp. 1683–1702. SIAM (2011)
8. Linial, N., Mansour, Y., Nisan, N.: Constant depth circuits, fourier transform, and learnability. J. ACM 40(3), 607–620 (1993)
9. Luks, E.M.: Hypergraph isomorphism and structural equivalence of boolean functions. In: 31st STOC, pp. 652–658. ACM (1999)

# Reversible Multi-head Finite Automata Characterize Reversible Logarithmic Space

Holger Bock Axelsen

DIKU, Department of Computer Science, University of Copenhagen, Denmark
funkstar@diku.dk

**Abstract.** Deterministic and non-deterministic multi-head finite automata are known to characterize the deterministic and non-deterministic logarithmic space complexity classes, respectively. Recently, Morita introduced *reversible* multi-head finite automata (RMFAs), and posed the question of whether RMFAs characterize reversible logarithmic space as well. Here, we resolve the question affirmatively, by exhibiting a clean RMFA simulation of logarithmic space reversible Turing machines. Indirectly, this also proves that reversible and deterministic multi-head finite automata recognize the same languages.

## 1 Introduction

Recently, Morita proposed *reversible multi-head finite automata* (RMFAs) [10, 11]. Multi-head finite automata (MFAs) and variations thereof have seen significant study in automata theory, cf. [6], but to our knowledge this is the first reversible variant. Besides the interest from automata theory, MFAs are interesting in computational complexity theory, because deterministic MFAs are known to exactly characterize *deterministic logarithmic space*, $\mathcal{L}(DMFA) = \mathsf{DSPACE}(\log n)$; an analogous result holds for non-deterministic MFAs [5]. In a very concrete way, MFAs embody the idea that logarithmic space consists of languages that can each be recognized with only a fixed amount of pointers. Morita poses the question of whether *reversible* MFAs characterize *reversible* logarithmic space [11, p. 253], which we answer in this paper.

RMFAs also present an intriguing sub-recursive model from the viewpoint of reversible computing theory. Morita gives examples of non-context-free context-sensitive languages accepted by RMFAs, so these automata accept languages across the levels of the Chomsky hierachy.[1] As regards the exact relationship of RMFAs and DMFAs, he conjectures that any $k$-head DMFA can be simulated by a $k$-head RMFA. Now, it is not obvious that reversible and deterministic variants of automata should be equally powerful: reversible finite automata and reversible pushdown automata are known to be strictly less powerful than their

[1] Recall that the context-sensitive languages are exactly recognized by linear-space non-deterministic Turing machines, *linear bounded automata*. By the space hierarchy theorem these are strictly more powerful than multi-head finite automata.

A.-H. Dediu and C. Martín-Vide (Eds.): LATA 2012, LNCS 7183, pp. 95–105, 2012.

deterministic counterparts [8,13]. Furthermore, well-known reversibilization techniques such as Bennett's method [4] rely on generating a run-time trace, and so cannot be leveraged to show the expressiveness of RMFAs by a naïve reversible simulation of DMFAs, since MFAs are read-only.

Our contribution is to show that RMFAs characterize reversible logspace, $\mathcal{L}(\mathit{RMFA}) = \mathsf{RevSPACE}(\log n)$, using a direct simulation of logarithmic space reversible Turing machines (RTMs) by RMFAs *without* intermediate reversibilization. In combination with the result that deterministic space equals reversible space [9], this provides an indirect proof that reversible and deterministic MFAs are equally expressive. Our simulation is quite efficient in terms of the number of heads: if the simulated RTM is bounded by $k \log n$ bits on the work tape, then it is simulated by an RMFA with $k + 4$ heads; one head better than the (irreversible) MFA simulation of logspace Turing machines [5].

## 2 RTMs and RMFAs

We shall assume that the reader is somewhat familiar with (reversible) Turing machines (RTMs) and multi-head finite automata (MFAs). We therefore only outline the aspects that influence our simulation. Readers seeking a more thorough introduction are referred to [3,4] for RTMs, and [6,11] for (R)MFAs.

### 2.1 Reversible Turing Machines

We shall concern ourselves with a sub-linear space complexity class. The Turing machines (TMs) used to study such classes are equipped with two tapes: a read-only *input tape* containing an input word $w \in \Sigma^*$ over some input alphabet $\Sigma$, written in delimited form $\triangleright w \triangleleft$, where $\triangleright$ and $\triangleleft$ are left and right endmarkers, respectively; and a read-write *work tape* with $k$ binary tracks (with all cells initially $0^k$). The input tape head starts by pointing at $\triangleright$ and must not move left of $\triangleright$ nor right of $\triangleleft$ (this can be statically enforced in the transition function). To simplify the simulation below, TMs are here assumed to work with one-way infinite tapes.[2] (See Fig. 3 for a conceptual representation of an RTM.)

We use a triple format for the transition rules that separates head movement from symbol substitution, cf. [3]. For example, we write $(q, [\leftarrow, \downarrow], p)$ for the *move rule* that from state $q$ moves the input head left, keeps the work head in place and goes to state $p$; and $(q, [a, 100 \mapsto 101], p)$ for the *symbol rule* that in state $q$, when reading $a$ on the input tape and 100 on the work tape, writes 101 to the work tape and goes to state $p$. (This example uses a 3-track work tape.) Note that because the input tape is read-only, there is no need to specify a target symbol for the input tape.

A TM is *reversible* iff it is deterministic, and, for every disjoint transition rule pair $(q_1, [a_1, b_1], p_1)$ and $(q_2, [a_2, b_2], p_2)$, if $p_1 = p_2$ then $b_1 = s_1 \mapsto t_1$ and $b_2 = s_2 \mapsto t_2$, and $t_1 \neq t_2$ or $a_1 \neq a_2$. That is, if multiple rules target the same state, then they are all symbol rules, and they write different symbols on the tape(s).

[2] These assumptions do not affect the expressive power or space complexity of TMs.

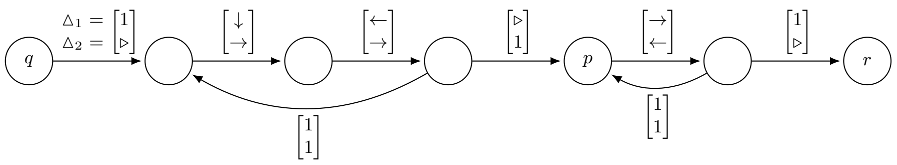


**Fig. 1.** Transition diagram for a 2-head RMFA that doubles the unary counter in $\triangle_1$

Inversion of an RTM is accomplished by straightforward transition rule inversion. For example, the inverse of move rule $(q, [\leftarrow, \downarrow], p)$ is $(p, [\rightarrow, \downarrow], q)$, and the inverse of symbol rule $(q, [a, 100 \mapsto 101], p)$ is $(p, [a, 101 \mapsto 100], q)$. This leads to an RTM with inverse functionality.

We define $\mathsf{RevSPACE}(s(n))$ to be the set of languages that can be decided by RTMs which use $\mathcal{O}(s(n))$ space on the work tape, given input of size $n$. In particular, $\mathsf{RevSPACE}(\log n)$ is the *reversible logarithmic space* complexity class.

### 2.2 Reversible Multihead Finite Automata

A formal definition of RMFAs can be found in [11] and we shall not reproduce the definition here. Instead, we note that RMFAs can be regarded as RTMs with multiple heads on the input tape and no work tape. Thus, we can use the RTM framework for RMFAs, including the triple transition rule format and the definition of reversibility.[3] For a $k$-head MFA over alphabet $\Sigma$ we shall write $(q, [s_1, \ldots, s_k], p)$ for the symbol rule that in state $q$ reads $s_i \in (\Sigma \cup \{\triangleright, \triangleleft\})$ with the $i$th head (for all $i \in \{1, \ldots, k\}$), and goes into state $p$; and we write $(q, [d_1, \ldots, d_k], p)$ for the move rule that in state $q$ moves the $i$th head in direction $d_i \in \{\leftarrow, \downarrow, \rightarrow\}$. We follow Morita's lead and label heads by $\triangle$ (with subscripts of various kinds to differentiate them).

Let $\mathcal{L}(RMFA)$ be the set of languages accepted by RMFAs with any number of heads, which all start by pointing at the left delimiter, $\triangleright$. $\mathcal{L}(RMFA)$ contains many interesting non-trivial languages: palindromes, unary primes, and the Dyck language of well-balanced parentheses are explicated in [11]. Rather than giving a full example here, we shall show a subroutine where we double a unary counter with a 2-head RMFA, Fig. 1. This will be useful for the RTM simulation below.

*Example (doubling unary counter).* Assume that the input tape is $\triangleright 1^n \triangleleft$, *i.e.* $n$ repetitions of 1. We begin in state $q$ with a configuration where the first head $\triangle_1$ of the 2-head RMFA points at the $k$th cell of the input (*i.e.*, the value of the counter $\triangle_1$ is $k$) with $2k \leq n$, and the second head $\triangle_2$ points at $\triangleright$ (*i.e.*, counter $\triangle_2$ is zero). Fig. 1 shows a simple RMFA for doubling $\triangle_1$ as follows. We move $\triangle_1$ sequentially to the left, while moving $\triangle_2$ two steps right for each left step of $\triangle_1$. Once $\triangle_1$ points at $\triangleright$, $\triangle_2$ will be $2k$ places to the right of $\triangleright$, and we go to state $p$. From here, we swap the positions of $\triangle_1$ and $\triangle_2$, and end in state $r$.

[3] MFA transition rules can also use a quadruple format analogous to the quintuple format for TMs, as done in [11].

$$\begin{tikzcd}
\mathcal{L}(\text{RMFA}) \arrow[r, phantom, "\overset{\text{Conjecture [11]}}{=\!=}"] \arrow[d, phantom, "\| \text{ here}"] & \mathcal{L}(\text{DMFA}) \arrow[d, phantom, "\| \ [5]"] \\
\mathsf{RevSPACE}(\log n) \arrow[r, phantom, "\overset{[9]}{=\!=}"] & \mathsf{DSPACE}(\log n)
\end{tikzcd}$$

**Fig. 2.** Equivalences between the languages recognized by reversible and deterministic MFAs and logarithmic space complexity classes. Our contribution is the direct proof of $\mathcal{L}(\text{RMFA}) = \mathsf{RevSPACE}(\log n)$.

This small program fragment can fairly easily be used to recognize *e.g.* (unary) $2^m$. Furthermore, because the subroutine is reversible, the *inverse* of the automaton will be an RMFA that *halves* an even number in $\triangle_1$, which will also be useful for the RTM simulation.

## 3 RMFAs Characterize Reversible Logspace

To establish the relationship between RMFAs and RTMs there is an immediate, but indirect, approach: simulate DMFAs by RMFAs. Because DMFAs characterize deterministic logarithmic space [5], and because this is equal to reversible logarithmic space [9], such a simulation would prove that RTMs characterize reversible logarithmic space (see Fig. 2).

Of concern in this approach is that we would use reversibilizations (simulations of irreversible machines) in both directions, which is wasteful: we know that in many contexts we do not have to use reversibilization at all when relating two reversible models, *e.g.* [2,14]. Also, the trace approach which is usually used for reversibilization is not (naïvely) applicable to MFAs: there is nowhere to write the trace. As an alternative, Morita [11] suggests adapting the reversibilization technique of [9] to RMFAs, but this does not alleviate our concern.

We propose a more direct proof strategy which sidesteps irreversible machine models entirely and goes directly for $\mathsf{RevSPACE}(\log n)$: we shall construct a simulation of logspace RTMs by RMFAs. Now, as mentioned above, it is known that MFAs are equivalent to logspace TMs. The proof of this is a briefly sketched simulation in [5, pp. 338–339], omitting most of the technical details. As one might suspect, these details turn out to contain a fair amount of devilry; especially when we want to prove the analogous relationship in a reversible setting. Still, the fundamental idea of the simulation holds, as we shall see.

We proceed as follows: first, we show how to encode the configuration of a logspace RTM by an RMFA; then, we show how to simulate RTM transitions.

### 3.1 Encoding an RTM Configuration

We must show that for any logspace RTM $T$, there exists an RMFA $R$ which accepts exactly the same language as $T$. $R$ will have as its input exactly the input tape of $T$. Now, if $L \in \mathsf{RevSPACE}(\log n)$, then there exists an RTM which

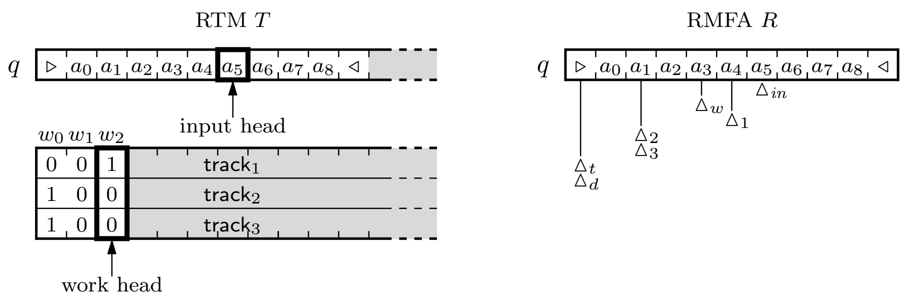


**Fig. 3.** Example configuration for a logspace RTM $T$ with 3 work tracks and the 7-head RMFA $R$ encoding $T$ (both in internal state $q$). The gray parts of the RTM tapes are 'out-of-bounds'—the read/write heads will never go there. The input head of $T$ points to input cell $a_5$, so $\triangle_{in}$ does as well. The work tape head of $T$ points to work cell $w_2$, so $\triangle_w$ is $2^2$ right shifts from $\triangleright$. The content of a work track from $T$ is simulated by the position of a track head in $R$: track$_1$ contains 001 which is the reverse binary representation of 4, so $\triangle_1$ points to $a_4$. Similarly, $\triangle_2$ and $\triangle_3$ point to $a_1$ to encode 100. The auxiliary RMFA heads $\triangle_t$ and $\triangle_d$ are in their default 'zero' position at $\triangleright$.

both decides $L$ and is $\log n$ tape bounded (by the tape compression theorem[4]). Without loss of generality we therefore assume that $T$ uses space $s(n) \leq \log n$ for all input lengths $n$. An example RTM configuration and its corresponding RMFA encoding is shown in Fig. 3; it works as follows.

The internal state of $T$ will be directly reflected by $R$'s internal state, so that for every state $q$ in $T$ there is a corresponding state $q$ in $R$. We us one of $R$'s heads, labelled $\triangle_{in}$ (the *input head*), to directly simulate $T$'s input tape head. Thus, the major concern is how to simulate $T$'s work tape. We assumed that $T$ is strictly $\log n$ tape bounded, and that $T$'s work tape is a $k$-track binary tape, *i.e.* with each work tape symbol encoded as a $k$-bit tuple. This means that there are at most $2^{k \log n}$ different possible work tape contents for $T$. Now, $2^{k \log n} = n^k$, which is the amount of possible configurations of a $k$-head RMFA, so there is exactly enough room to represent each such content by a unique RMFA configuration of $k$ heads.

Thus, we shall use $k$ heads in $R$ (labelled $\triangle_1$ through $\triangle_k$) to represent the work tape contents of $T$. The position of each such *track head* $\triangle_i$ will be a straightforward *unary* representation of the $\log n$ bits of a track when read from left to right, such that pointing at the first cell in the RMFA input represents 0. (Note that except for $\triangle_{in}$, all the heads of $R$ shall ignore the actual *symbols* in the input word.) Flipping a bit in a track is simulated as follows. To flip the $p$th bit[5] from 0 to 1, we add $2^p$ to the unary counter by moving the tape head

[4] Strictly speaking, the tape compression theorem has not yet been established for RTMs. However, the standard method of compressing cells into symbols also works for RTMs, as this is conceptually the inverse operation of symbol reduction, which *is* well established, cf. [3,12].

[5] As usual, bits are numbered starting from 0.

corresponding to this track $2^p$ positions to the right. To flip the bit from 1 to 0, move the tape head $2^p$ positions to the left, subtracting $2^p$ from the counter.

As an example, $\triangle_2$ in Fig. 3 points at input cell $a_1$, which encodes the bit string 100. To flip the the middle bit from 0 to 1 we move $\triangle_2$ $2^1$ places to the right. It will then point at input cell $a_3$, *i.e.* it will encode 110, as desired.

The work tape head of $T$ is simulated by the *work head* $\triangle_w$ in $R$ which maintains $2^p$ as a unary counter: if $T$'s work tape head points to cell (bit) $p$, the work head $\triangle_w$ is $2^p$ places from the left delimiter $\rhd$. Note that neither $\triangle_w$ nor any of the $k$ track heads will ever "overflow" by hitting the right delimiter $\lhd$, as this would break the assumption that $T$ is strictly $\log n$ tape bounded. Using an auxiliary head $\triangle_d$ (which initially points to $\rhd$) we can easily double or halve the $\triangle_w$ counter using the procedure in Fig. 1. This corresponds to moving the work tape head right or left. (Note that we do not need a counter for $p$ itself.)

A final auxiliary head $\triangle_t$ will be used when reading symbols from the encoding, meaning that $R$ has $k+4$ heads in total. To initialize the simulation, we only need to move the work head $\triangle_w$ and track heads $\triangle_1, \ldots, \triangle_k$ one place to the right of $\rhd$.

## 3.2 Simulating RTM Transitions

We now turn to how a transition rule of RTM $T$ is simulated by RMFA $R$.

**Move Rule.** The above encoding yields a completely straightforward simulation of a move rule $(q, [d_{in}, d_w], p)$: if the internal state of $R$ is $q$, then move the input head $\triangle_{in}$ as specified directly by $d_{in}$, and double or halve the work tape counter $\triangle_w$ as specified by $d_w$ (using the auxiliary head $\triangle_d$ and the doubling procedure of Fig. 1), ending by going into state $p$. This will be reversible as $(q, [d_{in}, d_w], p)$ is guaranteed to be the only rule out of $q$, and only rule that targets $p$, by the reversibility of $T$.

**Symbol Rule.** Simulating a symbol rule is trickier. The internal state $q$ of $T$ is directly reflected in $R$, but there are usually multiple symbol rules going out of each state and we do not know which one to simulate unless we read off the encoding.[6] To read a symbol off the simulated work tape, we further need to consult *all* $k$ tracks (meaning heads in $R$) because each of them carry one bit of the symbol's binary representation.[7] We thus need procedures for reading a

[6] We remark that while [5] mentions how to flip bits between known bit patterns, how to *read* the encoded symbols with the MFA is not discussed at all.

[7] One can equally well assume the inverse trade-off between tape bound and alphabet size, *i.e.*, that $T$ has only 2 symbols in its work tape alphabet rather than $2^k$, but that it is then $k \log n$ tape bounded rather than $\log n$ bounded. We then consult just a single head to read off a symbol, but we must also maintain which of $k$ segments of $\log n$ bits is active. The head count for such a simulation is the same as the one here, but the simulation has some tricky subtleties, so we chose Hartmanis' configuration encoding for this paper.

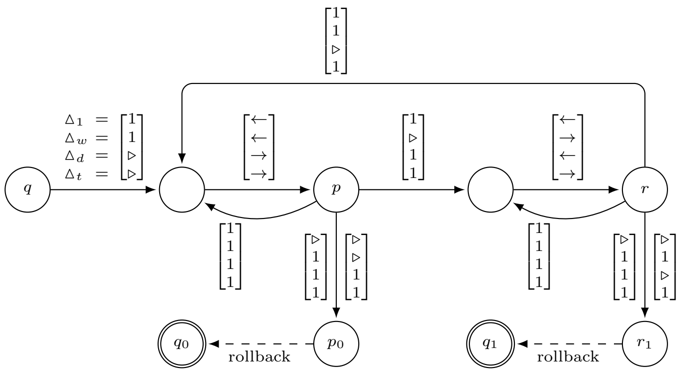


**Fig. 4.** RMFA procedure for reading off the $p$th bit in the binary representation of $\triangle_1$'s unary counter. For simplicity, this RMFA has a unary input alphabet: for richer alphabets all combinations of symbols (except for the delimiters) should be supplied for the symbol rules. Parity of the subtraction count (corresponding to the value of the bit) is maintained by switching back and forth between the loops at $p$ and $r$. The *rollbacks* are shorthand for two disjoint copies of the top five states of the automaton with their transitions inverted.

single bit off a given track (Fig. 4), and for collecting the bits from all $k$ simulated tracks and applying the corresponding symbol change (Fig. 5).

*Single bit reading.* We remark that the $p$th bit in the binary representation of an integer $k$ is $(k \text{ div } 2^p) \bmod 2$. Thus, to read off the $p$th bit of the first track $\triangle_1$, we repeatedly "subtract" $2^p$ (the value of $\triangle_w$) from $\triangle_1$ by moving it $2^p$ positions to the left, until $\triangle_1$ reaches the left endmarker $\triangleright$. Simultaneously, we conserve the original position of $\triangle_1$ by moving the auxiliary head $\triangle_t$ from $\triangleright$ to the right. We also use the auxiliary head $\triangle_d$ to conserve the $2^p$ counter in $\triangle_w$ (which is consumed by a subtraction otherwise). Depending on whether we subtract $2^p$ an *odd* or *even* number of times (maintained internally in the state of machine) before running into $\triangleright$, the $p$th bit on this track is 0 or 1.[8] The value of the bit from track 1 can thus be encoded by disjoint internal states. Importantly, this procedure can be completely reversible.

Now, the involved heads $\triangle_1$, $\triangle_w$, $\triangle_d$, and $\triangle_t$ will be displaced from their original position by this procedure. To restore them to their positions before the reading, we do as follows. We link up the two states that encode the different bit readings to their corresponding states in two disjoint, inverse copies of the procedure automaton. The effect of these inverse copies will be to *roll back* the calculation, restoring the initial positions of all heads, while maintaining the

[8] As every logic circuit designer knows, projecting the sequence $0, 1, 2, \ldots$ to the $p$th bit in its binary representation gives a periodic sequence consisting of $2^p$ 0's followed by $2^p$ 1's.

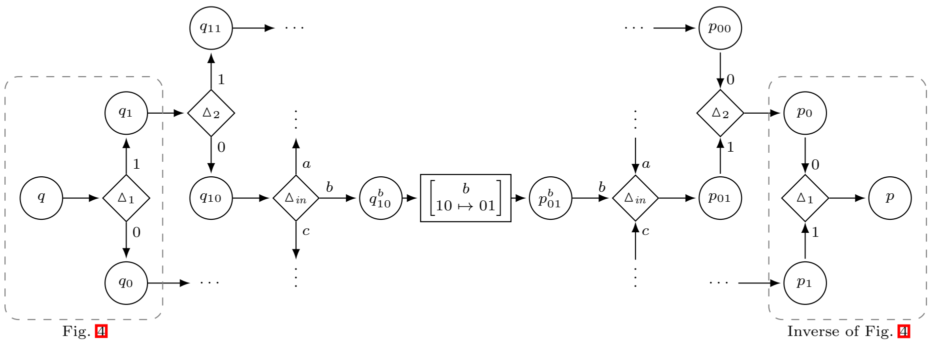


**Fig. 5.** Implementation of symbol rules by joining the leaves of decision trees for source states and symbols to the leaves of *inverse* decision tree for the *target* states and symbols. The fragment shown corresponds to the rule $(q, [b, 10 \mapsto 01], p)$ for a 2-track RTM. The conditionals on $\triangle_1$ and $\triangle_2$ use the reading procedure of Fig. 4 (and its inverse), while the ones on $\triangle_{in}$ read directly from the input. The centre box denotes applying the hardcoded directions for bit flips for this rule (left for $\triangle_1$, right for $\triangle_2$).

read bit in the internal state, ending up in, say, state $q_0$ or $q_1$.[9] Fig. 4 shows the RMFA for this clean bit reading procedure.

*Symbol substitution.* We now proceed to read off the other track heads in similar fashion, until we end up with the full $k$-bit string for the symbol of the current work cell encoded in the internal RMFA state (along with $T$'s internal state). This, along with the symbol directly on the (input) tape, then exactly defines which symbol rule $(q, (a, s \mapsto t), p)$ to apply. We then simply move the $k$ work track heads $2^p$ places left or right (or not at all) simultaneously, as defined by the bit flips in the binary string mapping $s \mapsto t$.

At this point, the internal state of the RMFA will reflect information about the *exact* transition rule that was applied. Now, we want to end up in state $p$, but there is the problem that many different transition rules can target $p$. In the irreversible MFA simulations we can simply join these cases directly, but in the RMFA they must be orthogonalized to preserve reversibility. To do so we (again) exploit the reversibility of $T$: only the symbol rule $(q, (a, s \mapsto t), p)$ can have been applied if the current symbol on the input tape is $a$, the current symbol on the work tape is $t$, and the current state is $p$. Thus, by using the *inverse* of the track reading procedures, which we get by RTM inversion, we can orthogonalize the case where this particular rule was applied from other symbol rules targeting $p$.

The resulting structure is effectively that *decision trees* for reading the simulated symbols starting from each internal state link up at the leaves with *inverse* decision trees for reading the target symbols and states, see Fig. 5.[10]

[9] This is effectively the construction for garbage-less RMFAs [10, Thm. 1]. The author used a similar technique for reversible string comparison in a universal RTM, cf. [2].

[10] This technique of joining up the leaves of the decision trees was used for symbol reduction in single-tape RTMs [3, 12].

The simulation of either rule type thus ends up in an RMFA configuration reflecting *only* the corresponding RTM configuration (with no trace information in the RMFA configuration). We conclude that $\mathsf{RevSPACE}(\log n) \subseteq \mathcal{L}(RMFA)$. The simulation circumvents the problem of having to simulate an irreversible MFA, avoiding the inherent trade-offs of reversibilization. The result is that an RTM using $k \log n$ bits of space can be simulated with an RMFA with $k + 4$ heads, so the simulation is quite efficient in terms of heads. In fact, we even use one head less than the irreversible simulation [5]. Despite the restrictions of reversibility, it is thus no more costly to simulate RTMs by RMFAs (in terms of heads) than it is to simulate TMs by MFAs in general.

### 3.3 Simulating an RMFA with a Logarithmic Space RTM

The reverse simulation is easier. Indirectly, the result is already established: RMFAs are a subset of DMFAs, which characterize logspace, which is equal to reversible logspace, see Fig. 2.

Directly, simulating an RMFA with a logspace RTM can be done as follows. The internal state of the RMFA is stored in the internal state of the RTM. The $k$ heads of the RMFA are simulated by $k$ binary tracks on the work tape of the RTM. Each track contains the cell offset of a head from the left delimiter $\triangleright$, *i.e.* a pointer. Obviously, only $\log n$ space is needed for each pointer. Moving the heads left or right is simple incrementation or decrementation of the pointer. By dereferencing each pointer in turn (moving the input head from $\triangleright$ to the referenced cell and back), we can collect the information about the symbols pointed to in the internal state, and update it as necessary (analogous to the reading of a work tape cell, Fig. 4). The only auxiliary space we need is an extra track to conserve the value of a pointer over a dereference (again, analogous to the way $\triangle_t$ and $\triangle_d$ was used above), and a track to mark off the $\log n$ cells allowed to the pointers (to avoid overflowing when incrementing and decrementing reversibly). The RTM simulation thus requires only $(k + 2) \log n$ bits of space to simulate a $k$-head RMFA.

Thus, $\mathcal{L}(RMFA) \subseteq \mathsf{RevSPACE}(\log n)$, and in combination with the RMFA simulation of logspace RTMs we conclude that $\mathcal{L}(RMFA) = \mathsf{RevSPACE}(\log n)$: reversible multi-head finite automata characterize reversible logarithmic space.

## 4 Further Discussion of RMFAs

Reversible multi-head finite automata have several other properties that set them apart from their irreversible counterparts, and which make them interesting for separate study. We shall briefly discuss some here.

RMFAs always terminate: either we end in a proper halting state, or get stuck. They cannot diverge, as this would mean that some configuration reachable from the starting state has more than one predecessor, which violates reversibility.

(No transition may target the starting state, so finite orbits with the starting state are forbidden.) In fact, this argument extends to any space-bounded reversible computation.[11]

By eliminating stuck states we can therefore to turn all reversible space-bounded language recognizers (acceptors) into language *deciders* [11].[12] That is, we can without loss of generality assume that RMFAs are total. Now, we can also assume this for DMFAs as they recognize decidable languages, but the construction to enforce it is *much* simpler for RMFAs.

Finally, if we grant write privileges to RMFAs, we end up with another interesting computation model: *reversible linear bounded automata*, RLBAs. These are easily seen to characterize linear space, $\mathcal{L}(RLBA) = \mathsf{RevSPACE}(n)$, and share many of the features of RMFAs. The non-deterministic LBAs recognize exactly the context-sensitive languages, and the longstanding problem in automata (and complexity) theory of whether non-deterministic and deterministic LBAs recognize the same languages can thus also be phrased in terms of *reversible* LBAs versus non-deterministic LBAs.

## 5 Conclusion

We showed that reversible multi-head finite automata (RMFAs) characterize reversible logarithmic space, by a simulation of logspace reversible Turing machines by RMFAs (and vice versa). This answers a question posed by Morita [11]. The construction is based on that of Hartmanis [5], but with considerably more detail and intricacy because of the requirements of reversibility. The simulation incorporates several recent techniques and ideas from other clean reversible simulations of reversible models [1–3,11,14]. We slightly improve on the irreversible construction [5] by requiring one less head, for $k + 4$ heads in total.

Indirectly, by the results of Lange *et al.* [9] this also shows that RMFAs and DMFAs recognize the same languages. The slightly stronger conjecture of Morita, that any $k$-head DMFA can be simulated by a $k$-head RMFA, is still open, but we join in the belief that the simulation of [9] should be applicable, placing RMFAs in the DMFA hierarchy [7]. However, a novel solution to this problem would be more welcome as, currently, the arsenal of reversibilizations is very limited.

We believe that the characterization of reversible complexity classes by automata could prove fruitful for future research. For example, considering the $\mathsf{L}$ vs. $\mathsf{NL}$ problem in terms of RMFAs could yield new insights.

**Acknowledgements.** Thanks are due to Michael Kirkedal Thomsen and Robert Glück for comments on a draft of this paper.

[11] The argument also applies conversely: if, say, an RTM diverges, then it necessarily uses unbounded space.

[12] An equivalent construction was given by the author for RTMs [1].

## References

1. Axelsen, H.B.: Time Complexity of Tape Reduction for Reversible Turing Machines. In: De Vos, A., Wille, R. (eds.) RC 2011. LNCS, vol. 7165, Springer, Heidelberg (2012)
2. Axelsen, H.B., Glück, R.: A Simple and Efficient Universal Reversible Turing Machine. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 117–128. Springer, Heidelberg (2011)
3. Axelsen, H.B., Glück, R.: What Do Reversible Programs Compute? In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 42–56. Springer, Heidelberg (2011)
4. Bennett, C.H.: Logical reversibility of computation. IBM Journal of Research and Development 17(6), 525–532 (1973)
5. Hartmanis, J.: On non-determinancy in simple computing devices. Acta Informatica 1(4), 336–344 (1972)
6. Holzer, M., Kutrib, M., Malcher, A.: Multi-head finite automata: Characterizations, concepts and open problems. In: Neary, T., Woods, D., Seda, A.K., Murphy, N. (eds.) Complexity of Simple Programs. EPTCS, vol. 1, pp. 93–107 (2008)
7. Ibarra, O.H.: On two-way multihead automata. J. Comput. and Sys. Sci. 7(1), 28–36 (1973)
8. Kutrib, M., Malcher, A.: Reversible Pushdown Automata. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 368–379. Springer, Heidelberg (2010)
9. Lange, K.-J., McKenzie, P., Tapp, A.: Reversible space equals deterministic space. J. Comput. and Sys. Sci. 60(2), 354–367 (2000)
10. Morita, K.: Reversible multi-head finite automata and languages accepted by them (extended abstract). In: Wille, R., De Vos, A. (eds.) Proceedings of the 3rd Workshop on Reversible Computation, pp. 25–30. Universiteit Gent (2011)
11. Morita, K.: Two-way reversible multi-head finite automata. Fundamenta Informaticae 110(1-4), 241–254 (2011)
12. Morita, K., Shirasaki, A., Gono, Y.: A 1-tape 2-symbol reversible Turing machine. Transactions of the IEICE E 72(3), 223–228 (1989)
13. Pin, J.-E.: On the Languages Accepted by Finite Reversible Automata. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 237–249. Springer, Heidelberg (1987)
14. Yokoyama, T., Axelsen, H.B., Glück, R.: Reversible Flowchart Languages and the Structured Reversible Program Theorem. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 258–270. Springer, Heidelberg (2008)

# Defining Contexts in Context-Free Grammars*

Mikhail Barash[1,2] and Alexander Okhotin[1]

[1] Department of Mathematics, University of Turku, Turku FI-20014, Finland
[2] Turku Centre for Computer Science, Turku FI-20520, Finland
{mikhail.barash,alexander.okhotin}@utu.fi

**Abstract.** Conjunctive grammars (Okhotin, 2001) are an extension of the standard context-free grammars with a conjunction operation, which maintains most of their practical properties, including many parsing algorithms. This paper introduces a further extension to the model, which is equipped with quantifiers for referring to the left context, in which the substring being defined does occur. For example, a rule $A \to a \,\&\, \lhd B$ defines a string $a$, as long as it is preceded by any string defined by $B$. The paper gives two equivalent definitions of the model—by logical deduction and by language equations—and establishes its basic properties, including a transformation to a normal form, a cubic-time parsing algorithm, and another recognition algorithm working in linear space.

## 1 Introduction

Context-free grammars are a logic for defining the syntax of languages. In this logic, the properties of strings are defined inductively, so that the properties of a string are determined by the properties of its substrings. This is how a rule $S \to aSb$ asserts, that if a string $a^{n-1}b^{n-1}$ has the property $S$, then the string $a^n b^n$ has the property $S$ as well. Besides the concatenation, the formalism of this logic has an implicit disjunction operation, represented by having multiple rules for a single symbol. This logic can be further augmented with conjunction and negation operations, which was done by the second author [11, 13] in *conjunctive grammars* and *Boolean grammars*, respectively. These grammars preserve the main idea of the context-free grammars—that of defining syntax inductively, as described above—maintain most of their practically important features, such as efficient parsing algorithms [13, 15–17], and have been a subject of diverse research [1, 4, 7, 8, 10, 18]. As the applicability of a rule of a Boolean grammar to a substring is independent of the context, in which the substring occurs, Boolean grammars constitute a natural general case of context-free grammars. Standard context-free grammars can be viewed as their disjunctive fragment.

When Chomsky [2] introduced the term "context-free grammar" for an intuitively obvious model of syntax, he had a further idea of a more powerful model, in which one could define rules applicable only in some particular contexts. However, Chomsky's attempt to formalize his idea using the tools available at the

* Supported by the Academy of Finland under grant 134860.

time (namely, the string-rewriting systems) led to nothing but space-bounded nondeterministic Turing machines. Even though the resulting devices are still known under the name of "context-sensitive grammars", they have nothing to do with the syntax of languages: the nonterminal symbols of these grammars are, in general, nothing but bits in a memory of a general-purpose computer, and do not correspond to any syntactic notions. In particular, these grammars fail to implement Chomsky's original idea of a phrase-structure rule applicable in a context.

This paper undertakes to reconsider Chomsky's [2] idea of contexts in grammars, this time using the appropriate tools of deduction systems and language equations, and drawing from the experience of developing the conjunctive grammars. The model proposed in this paper are *grammars with one-sided contexts*, which introduce two special quantifiers for representing left contexts of a string. The first quantifier refers to the "past" of the current substring within the entire string being defined: an expression $\lhd\alpha$ defines any substring that is directly preceded by a prefix of the form $\alpha$. This quantifier is meant to be used along with usual, unquantified specifications of the structure of the current substring, using conjunction to combine several specifications. For example, consider the rule $A \to BC \,\&\, \lhd D$, which represents any substring of the form $BC$ preceded by a substring of the form $D$. If the grammar contains additional rules $B \to b$, $C \to c$ and $D \to d$, then the above rule for $A$ shall specify that a substring $bc$ of a string $w = dbc\ldots$ has the property $A$; however, this rule shall not produce the same substring occurring in the strings $w' = abc$ or $w'' = adbc$. The other quantifier, $\trianglelefteq\alpha$, represents the form of the current substring together with its left context, so that the rules $A \to B \,\&\, \trianglelefteq E$, $B \to b$, $E \to ab$ define that the substring $b$ occurring in the string $w = ab$ has the property $A$. One can symmetrically define right contexts, denoted by quantifiers $\rhd\alpha$ and $\trianglerighteq\alpha$.

In the literature, related ideas have occasionally arisen in connection with parsing, where right contexts—$\trianglerighteq\alpha\Sigma^*$, in the terminology of this paper—are considered as "lookahead strings" and are used to guide a deterministic parser. If $\alpha$ represents a regular language, these simple forms of contexts occur in LR-regular [3], LL-regular [9] and LL(*) [19] parsers. Some software tools for engineering parsers, such as those developed by Parr and Fischer [19] and by Ford [5], allow specifying contexts $\trianglerighteq\alpha\Sigma^*$, with $\alpha$ defined within the grammar, and such specifications can be used by a programmer to adjust the behaviour of a deterministic recursive descent parser.

In this paper, the above intuitive definition of grammars with one-sided contexts is formalized in two equivalent ways. The first formalization, pursued in Section 2, uses *deduction* of elementary propositions of the form $[A, \langle u\rangle v]$, where $\langle u\rangle v$ denotes a substring $v$ in left context $u$ (that is, occurring in a string $uvw$) and $A$ is a syntactic property defined by the grammar ("nonterminal symbol" in Chomsky's terminology); this proposition asserts that $v$ has the property $A$ in the context $u$. Then, each rule of the grammar, which is of the general form $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd\beta_1 \,\&\, \ldots \,\&\, \lhd\beta_m \,\&\, \trianglelefteq\gamma_1 \,\&\, \ldots \,\&\, \trianglelefteq\gamma_n$, becomes a deduction scheme for inferring elementary propositions of this form from each other, and the language generated by the grammar is ultimately defined as the set of all strings $w$, such that $[S, \langle\varepsilon\rangle w]$ can be deduced. The standard proof tree of such a deduction constitutes a parse tree of the string $w$. This definition generalizes the representation of standard context-free grammars by deduction—assumed, for instance, in the monograph by Sikkel [20]—as well as the extension of this representation to conjunctive grammars [14].

An alternative, equivalent definition given in Section 3 uses a generalization of *language equations*, in which the unknowns are *sets of pairs* of a string and its left contexts. All connectives and quantifiers in the rules of a grammar—that is, concatenation, disjunction, conjunction and both context quantifiers—are then interpreted as operations on such sets, and the resulting system of equations is proved to have a least fixpoint, as in the known cases of standard context-free grammars [6] and conjunctive grammars [12]. This least solution defines the language generated by the grammar.

These definitions ensure that the proposed grammars with one-sided contexts define the properties of strings inductively from the properties of their substrings and the contexts, in which these substrings occur. There is no rewriting of "sentential forms" involved, and hence the proposed model avoids falling into the same pit as Chomsky's "context-sensitive grammars", that of being able to simulate computations of space-bounded Turing machines.

This paper settles the basic properties of grammars with one-sided contexts. First, a transformation to a normal form generalizing the Chomsky normal form is devised in Section 4; the construction proceeds in the usual way, first by eliminating empty strings, and then by removing cyclic dependencies. This normal form is then used to extend the basic Cocke–Kasami–Younger parsing algorithm to grammars with one-sided contexts; the algorithm, described in Section 5, works in time $O(n^3)$, where $n$ is the length of the input string. Finally, in Section 6, it is demonstrated that every language defined by a grammar with one-sided contexts can be recognized in deterministic linear space.

In this extended abstract, most proofs are omitted due to space constraints.

## 2 Definition by Deduction

A grammar with one-sided contexts uses concatenation, conjunction and disjunction, as well as quantifiers, either only $\{\lhd, \trianglelefteq\}$ for left contexts, or only $\{\rhd, \trianglerighteq\}$ for right contexts. Though left contexts are assumed throughout this paper, all results symmetrically hold for grammars with right contexts.

**Definition 1.** *A grammar with left contexts is a quadruple $G = (\Sigma, N, R, S)$, where*

- *$\Sigma$ is the alphabet of the language being defined;*
- *$N$ is a finite set of auxiliary symbols ("nonterminal symbols" in Chomsky's terminology), disjoint with $\Sigma$, which denote the properties of strings defined in the grammar;*

– *$R$ is a finite set of grammar rules, each of the form*

$$A \to \alpha_1 \,\&\, \dots \,\&\, \alpha_k \,\&\, \lhd\beta_1 \,\&\, \dots \,\&\, \lhd\beta_m \,\&\, \unlhd\gamma_1 \,\&\, \dots \,\&\, \unlhd\gamma_n, \qquad (1)$$

*with $A \in N$, $k, m, n \geqslant 0$, $\alpha_i, \beta_i, \gamma_i \in (\Sigma \cup N)^*$;*

– *$S \in N$ is a symbol representing correct sentences ("start symbol" in the common jargon).*

A grammar with left contexts degenerates to a conjunctive grammar, if the context quantifiers are never used, that is, if $m = n = 0$ for every rule (1); and further to a standard context-free grammar, if conjunction is never used, that is, if $k = 1$ in every rule.

For each grammar rule (1), each term $\alpha_i$, $\lhd\beta_i$ and $\unlhd\gamma_i$ is called a conjunct. Each unquantified conjunct $\alpha_i$ gives a representation of the string being defined. A conjunct $\lhd\beta_i$ similarly describes the form of the *left context* or the *past*, relative to the string being defined. Conjuncts of the form $\unlhd\gamma_i$ refer to the form of the left context and the current string, concatenated into a single string. Intuitively, such a rule asserts that every substring $v$ occurring in the left context $u$, such that $v$ is representable as each $\alpha_i$, $u$ is representable as each $\beta_i$ and and $uv$ is representable as each $\gamma_i$, therefore has the property $A$.

Formally, the semantics of grammars with contexts are defined by a deduction system of elementary propositions (items) of the form "a string $v \in \Sigma^*$ written in left context $u \in \Sigma^*$ has the property $\alpha \in (\Sigma \cup N)^*$", denoted by $[\alpha, \langle u \rangle v]$.

**Definition 2.** *Let $G = (\Sigma, N, R, S)$ be a grammar with contexts, and define the following deduction system of items of the form $[X, \langle u \rangle v]$, with $X \in \Sigma \cup N$ and $u, v \in \Sigma^*$. There is a single axiom scheme:*

$$\vdash_G [a, \langle x \rangle a] \quad (\textit{for all } a \in \Sigma \textit{ and } x \in \Sigma^*).$$

*Each rule $A \to \alpha_1 \,\&\, \dots \,\&\, \alpha_k \,\&\, \lhd\beta_1 \,\&\, \dots \,\&\, \lhd\beta_m \,\&\, \unlhd\gamma_1 \,\&\, \dots \,\&\, \unlhd\gamma_n$ in the grammar defines a scheme $I \vdash_G [A, \langle u \rangle v]$ for deduction rules, for all $u, v \in \Sigma^*$ and for every set of items $I$ satisfying the below properties:*

– *For every unquantified conjunct $\alpha_i = X_1 \dots X_\ell$ with $\ell \geqslant 0$ and $X_j \in \Sigma \cup N$, there should exist a partition $v = v_1 \dots v_\ell$ with $[X_j, \langle u v_1 \dots v_{j-1} \rangle v_j] \in I$ for all $j \in \{1, \dots, \ell\}$;*

– *For every conjunct $\lhd\beta_i = \lhd X_1 \dots X_\ell$ with $\ell \geqslant 0$ and $X_j \in \Sigma \cup N$, there should be such a partition $u = u_1 \dots u_\ell$, that $[X_j, \langle u_1 \dots u_{j-1} \rangle u_j] \in I$ for all $j \in \{1, \dots, \ell\}$;*

– *Every conjunct $\unlhd\gamma_i = \unlhd X_1 \dots X_\ell$ with $\ell \geqslant 0$ and $X_j \in \Sigma \cup N$ should have a corresponding partition $uv = w_1 \dots w_\ell$ with $[X_j, \langle w_1 \dots w_{j-1} \rangle w_j] \in I$ for all $j \in \{1, \dots, \ell\}$.*

*Note, that if $\alpha_i = \varepsilon$ in the first case, then the given condition implies $v = \varepsilon$, and similarly, $\beta_i = \varepsilon$ implies $u = \varepsilon$, and $\gamma_i = \varepsilon$ implies $u = v = \varepsilon$.*

*Then the language generated by a nonterminal symbol $A$ is defined as*

$$L_G(A) = \{\, \langle u \rangle v \mid u, v \in \Sigma^*, \vdash_G [A, \langle u \rangle v] \,\}.$$

*The language generated by the grammar $G$ is the set of all strings with left context $\varepsilon$ generated by $S$:*

$$L(G) = \{\, w \mid w \in \Sigma^*,\ \vdash_G [S, \langle\varepsilon\rangle w] \,\}.$$

Using both kinds of past quantifiers ($\lhd$ and $\unlhd$) is actually redundant, because each of them can be expressed through the other as follows:

– $\lhd D$ is equivalent to $D'\Sigma^*$, for a new symbol $D'$ with a rule $D' \to \varepsilon \,\&\, \unlhd D$;
– $\unlhd E$ can be replaced by $\Sigma^* E''$, where $E''$ has the unique rule $E'' \to \varepsilon \,\&\, \lhd E$.

Using both types of quantifiers together shall be essential later, when transforming a grammar into a normal form, where the empty string cannot be defined.

The following sample grammar with contexts defines a rather simple language, which is an intersection of two standard context-free languages. The value of this example is in demonstrating the machinery of contexts in action.

*Example 3.* The following grammar generates the language $\{\, a^n b^n c^n d^n \mid n \geqslant 0 \,\}$:

$$\begin{aligned} S &\to aSd \mid bSc \mid \varepsilon \,\&\, \lhd A \\ A &\to aAb \mid \varepsilon \end{aligned}$$

The symbol $A$ generates all strings $a^n b^n$ with $n \geqslant 0$ in any context. Without the context specification $\lhd A$, the symbol $S$ would define all strings of the form $wh(w^R)$, where $w \in \{a, b\}^*$ and the homomorphism $h$ maps $a$ to $d$ and $b$ to $c$. However, the rule $S \to \varepsilon \,\&\, \lhd A$ ensures that the first half of the string (the prefix ending with the last $b$) is of the form $a^n b^n$ for some $n \geqslant 0$, and therefore $S$ generates $\{\, a^n b^n c^n d^n \mid n \geqslant 0 \,\}$. Consider the following logical derivation of the fact that the string $abcd$ with the left context $\varepsilon$ is defined by $S$.

$$\begin{array}{rll}
 & \vdash [a, \langle\varepsilon\rangle a] & (\mathit{axiom}) \\
 & \vdash [b, \langle a\rangle b] & (\mathit{axiom}) \\
 & \vdash [c, \langle ab\rangle c] & (\mathit{axiom}) \\
 & \vdash [d, \langle abc\rangle d] & (\mathit{axiom}) \\
 & \vdash [A, \langle a\rangle \varepsilon] & (A \to \varepsilon) \\
[a, \langle\varepsilon\rangle a], [A, \langle a\rangle\varepsilon], [b, \langle a\rangle b] & \vdash [A, \langle\varepsilon\rangle ab] & (A \to aAb) \\
[A, \langle\varepsilon\rangle ab] & \vdash [S, \langle ab\rangle \varepsilon] & (S \to \varepsilon \,\&\, \lhd A) \\
[b, \langle a\rangle b], [S, \langle ab\rangle\varepsilon], [c, \langle ab\rangle c] & \vdash [S, \langle a\rangle bc] & (S \to bSc) \\
[a, \langle\varepsilon\rangle a], [S, \langle a\rangle bc], [d, \langle abc\rangle d] & \vdash [S, \langle\varepsilon\rangle abcd] & (S \to aSd)
\end{array}$$

The tree corresponding to this deduction is given in Figure 1, where the dependence upon a context is marked by a dotted arrow.

*Example 4.* The following grammar generates the language

$$\{\, u_1 \ldots u_n \mid \text{for every } i,\ u_i \in a^*c, \textbf{ or } \text{there exist } j, k \text{ with } u_i = b^k c \text{ and } u_j = a^k c \,\}.$$

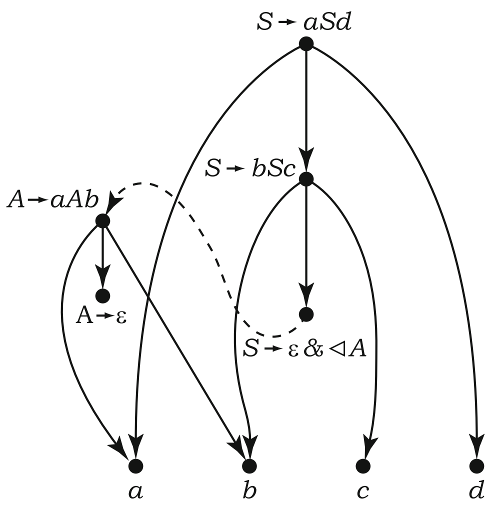


**Fig. 1.** A parse tree of the string *abcd* according to the grammar in Example 3

$$\begin{array}{lll}
S \to AcS \mid CcS \mid BcS \,\&\, DcE \mid \varepsilon & & \\
A \to aA \mid \varepsilon & C \to B \,\&\, \trianglelefteq EF & E \to AcE \mid BcE \mid \varepsilon \\
B \to bB \mid \varepsilon & D \to bDa \mid cE & F \to aFb \mid cE
\end{array}$$

This is an abstract language representing declaration of identifiers *before or after* their use. Substrings of the form $a^k c$ represent declarations, while every substring of the form $b^k c$ is a reference to a declaration of the form $a^k c$.

The idea of the grammar is that $S$ should generate a string $\langle u_1 \dots u_\ell \rangle u_{\ell+1} \dots u_n$ with $u_i \in a^*c \cup b^*c$ if every "reference" in the suffix $u_{\ell+1} \dots u_n$ has a corresponding "declaration" in the whole string $u_1 \dots u_n$. This condition is defined inductively on $\ell$. The rule $S \to \varepsilon$ is the basis of induction: the string $\langle u_1 \dots u_n \rangle \varepsilon$ has the desired property. The rule $S \to CcS$ appends a reference of the form $(b^* \,\&\, \trianglelefteq EF)c$, where the context specification ensures that this "reference" has a matching *earlier* "declaration". The possibility of a *later* "declaration" is checked by another rule $S \to BcS \,\&\, DcE$.

For the language in Example 4, no Boolean grammar is known (and hence no conjunctive grammar either). At the time of writing, this is the only such example known to the authors. This is due to the simple reason that conjunctive grammars are already quite powerful, and can define most of the standard examples of formal languages. No methods of proving that a language does not have a conjunctive grammar are currently known, and hence there is no proof that grammars with contexts generate a larger class of languages.

## 3 Definition by Language Equations

The representation of standard context-free grammars by language equations, introduced by Ginsburg and Rice [6], is one of the several ways of defining their semantics. For example, a grammar $S \to aSb \mid \varepsilon$ is regarded as an equation $S = \big(\{a\} \cdot S \cdot \{b\}\big) \cup \{\varepsilon\}$, which has a unique solution $S = \{\, a^n b^n \mid n \geqslant 0 \,\}$. Conjunctive grammars inherit the same definition by equations [12], with the conjunction represented by the intersection operation.

This important representation can be extended to grammars with contexts. However, in order to include the contexts in the equations, the whole model has to be extended from ordinary formal languages to sets of pairs of the form $\langle u \rangle v$, that is, to languages of pairs $L \subseteq \Sigma^* \times \Sigma^*$. All usual operations on languages used in equations are redefined for languages of pairs as follows. For all $K, L \subseteq \Sigma^* \times \Sigma^*$, consider their

- *union* $K \cup L = \{\, \langle u \rangle v \mid \langle u \rangle v \in K \text{ or } \langle u \rangle v \in L \,\}$,
- *intersection* $K \cap L = \{\, \langle u \rangle v \mid \langle u \rangle v \in K,\ \langle u \rangle v \in L \,\}$,
- *concatenation* $K \cdot L = \{\, \langle u \rangle vw \mid \langle u \rangle v \in K,\ \langle uv \rangle w \in L \,\}$,
- $\lhd$*-context* $\lhd L = \{\, \langle u \rangle v \mid \langle \varepsilon \rangle u \in L,\ v \in \Sigma^* \,\}$, and
- $\trianglelefteqslant$*-context* $\trianglelefteqslant L = \{\, \langle u \rangle v \mid \langle \varepsilon \rangle w \in L,\ w = uv \,\}$.

With respect to the partial order of inclusion, all these operations are monotone and continuous. Hence, every system of equations $X_i = \varphi_i(X_1, \ldots, X_n)$ with $i \in \{1, \ldots, n\}$, in which $X_i$ are unknown languages of pairs and the right-hand sides $\varphi_i$ are comprised of the above operations, has a least solution. A grammar is represented as such a system in the most direct way.

**Definition 5.** *For every grammar with contexts $G = (\Sigma, N, R, S)$, the associated system of language equations is a system of equations in variables $N$, in which each variable assumes a value of a language of pairs $L \subseteq \Sigma^* \times \Sigma^*$, and which contains the following equations for every variable $A$:*

$$A = \bigcup_{\substack{A \to \alpha_1 \& \ldots \& \alpha_k \& \\ \& \lhd\beta_1 \& \ldots \& \lhd\beta_m \& \\ \& \trianglelefteqslant\gamma_1 \& \ldots \& \trianglelefteqslant\gamma_n \in R}} \Big[ \bigcap_{i=1}^{k} \alpha_i \;\cap\; \bigcap_{i=1}^{m} \lhd\beta_i \;\cap\; \bigcap_{i=1}^{n} \trianglelefteqslant\gamma_i \Big]. \tag{2}$$

*Each instance of a symbol $a \in \Sigma$ in such a system defines the language $\{\, \langle x \rangle a \mid x \in \Sigma^* \,\}$, and each empty string denotes the language $\{\, \langle x \rangle \varepsilon \mid x \in \Sigma^* \,\}$.*

*Let $(L_{A_1}, \ldots, L_{A_n})$ with $L_{A_i} \subseteq \Sigma^* \times \Sigma^*$ be the least solution of the system. Define $L_G(A) = L_A$, and let $L(G) = \{\, w \mid \langle \varepsilon \rangle w \in L_S \,\}$.*

For instance, the grammar in Example 3 induces the system

$$\begin{aligned} S &= \langle \Sigma^* \rangle a \cdot S \cdot \langle \Sigma^* \rangle d \ \cup\ \langle \Sigma^* \rangle b \cdot S \cdot \langle \Sigma^* \rangle c \ \cup\ \big(\langle \Sigma^* \rangle \varepsilon \ \cap\ \lhd A\big) \\ A &= \langle \Sigma^* \rangle a \cdot A \cdot \langle \Sigma^* \rangle b \ \cup\ \langle \Sigma^* \rangle \varepsilon \end{aligned}$$

with a unique solution $S = \{\, \langle a^i \rangle a^{n-i} b^n c^n d^n \mid n \geqslant i \geqslant 0 \,\} \cup \{\, \langle a^n b^i \rangle b^{n-i} c^n d^n \mid n \geqslant i \geqslant 0 \,\}$, $A = \{\, \langle x \rangle a^n b^n \mid x \in \{a, b\}^* \,\}$.

Definitions 2 and 5 are proved equivalent as follows.

**Theorem 6.** *Let $G = (\Sigma, N, R, S)$ be a grammar with contexts, let $X = \varphi(X)$ be the associated system of equations. For every $A \in N$ and $\langle u \rangle v \in \Sigma^* \times \Sigma^*$,*

$$\langle u \rangle v \in \Big[ \bigsqcup_{k \geqslant 0} \varphi^k(\varnothing, \ldots, \varnothing) \Big]_A \quad \textit{if and only if} \quad \vdash_G [A, \langle u \rangle v].$$

## 4 Normal Form

Consider the Chomsky normal form for standard context-free grammars, in which all rules are of the form $A \to BC$ and $A \to a$. Its extension to conjunctive grammars allows rules of the form $A \to B_1C_1 \& \ldots \& B_kC_k$. The transformation to this normal form is done by first eliminating *$\varepsilon$-conjuncts*, that is, rules of the form $A \to \varepsilon \& \ldots$, and then removing *unit conjuncts*, or rules of the form $A \to B \& \ldots$ [11]. This transformation shall now be further extended to grammars with contexts.

The task of eliminating $\varepsilon$-conjuncts is formulated in the same way: for any given grammar with contexts, the goal is to construct an equivalent (modulo the membership of $\varepsilon$) grammar without $\varepsilon$-conjuncts. A similar construction for context-free grammars (as well as for conjunctive grammars) begins with determining the set of nonterminals that generate the empty string, which is obtained as a least upper bound of an ascending sequence of sets of nonterminals. For grammars with contexts, it is necessary to consider pairs of the form $(A, Z)$, with $A \in N$ and $Z \subseteq N$, representing the intuitive idea that $A$ generates $\varepsilon$ in the context of the form described by all nonterminals in $Z$. The set of all such pairs is obtained as a limit of a sequence of sets as follows.

**Definition 7.** *Let $G = (\Sigma, N, R, S)$ be a grammar with contexts. Assume, without loss of generality, that in each rule of the grammar, the context quantifiers are applied only to single nonterminal symbols rather than concatenations. Construct the sequence of sets $\textsc{Nullable}_i(G) \subseteq N \times 2^N$, with $i \geqslant 0$, by setting $\textsc{Nullable}_0(G) = \varnothing$ and*

$$\begin{aligned}\textsc{Nullable}_{i+1}(G) = \big\{ \, (A, \{D_1, \ldots, D_m\} \cup \{E_1, \ldots, E_n\} \cup Z_1 \cup \ldots \cup Z_k) \; \big| \\ A \to \alpha_1 \& \ldots \& \alpha_k \& \lhd D_1 \& \ldots \& \lhd D_m \& \trianglelefteq E_1 \& \ldots \& \trianglelefteq E_n \in R, \\ Z_1, \ldots, Z_k \subseteq N : \; (\alpha_1, Z_1), \ldots, (\alpha_k, Z_k) \in \textsc{Nullable}_i^*(G) \, \big\},\end{aligned}$$

*where $\mathcal{S}^*$, for any set $\mathcal{S} \subseteq N \times 2^N$, denotes the set of all pairs $(A_1 \ldots A_\ell, \, Z_1 \cup \ldots \cup Z_\ell)$ with $\ell \geqslant 0$ and $(A_i, Z_i) \in \mathcal{S}$.*

*Finally, let $\textsc{Nullable}(G) = \bigcup_{i \geqslant 0} \textsc{Nullable}_i(G)$.*

In the definition of $\mathcal{S}^*$, note that $\varnothing^* = \{(\varepsilon, \varnothing)\}$. This value of $\textsc{Nullable}_i^*(G)$ is used in the construction of $\textsc{Nullable}_1(G)$.

The next lemma explains how the set $\textsc{Nullable}(G)$ represents the generation of $\varepsilon$ by different nonterminals in different contexts.

**Lemma 8.** *Let $G = (\Sigma, N, R, S)$ be a grammar with contexts, let $A \in N$ and $u \in \Sigma^*$. Then, $\langle u \rangle \varepsilon \in L_G(A)$ if and only if there exist $K_1, \ldots, K_t \in N$, such that $(A, \{K_1, \ldots, K_t\}) \in \textsc{Nullable}(G)$ and $\langle \varepsilon \rangle u \in L_G(K_1)$, ..., $\langle \varepsilon \rangle u \in L_G(K_t)$.*

It is convenient to define the elimination of $\varepsilon$-conjuncts for a grammar with contexts $G = (\Sigma, N, R, S)$, in which every symbol $A \in N$ either has one or more rules of the form $A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n$ with $B_i, D_i, E_i \in N$, or a unique rule of the form $A \to BC$ (with $B, C \in N$), $A \to a$ (with $a \in \Sigma$) or $A \to \varepsilon$. The pre-processing necessary to reach this intermediate form is straightforward, and then an equivalent grammar $G' = (\Sigma, N, R', S)$ without $\varepsilon$-conjuncts is constructed as follows:

1. All rules of the form $A \to a$ in $R$ are added to $R'$.
2. Every rule $A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n$ in $R$ is added to $R'$. Additionally, if $\langle \varepsilon \rangle \varepsilon \in L_G(D_1) \cap \ldots \cap L_G(D_m)$, then another rule $A \to B_1 \,\&\, \ldots \,\&\, B_k \,\&\, E_1 \,\&\, \ldots \,\&\, E_n \,\&\, \lhd\varepsilon$ is added to $R'$.
3. Every rule $A \to BC \in R$ is added to $R'$, along with the following ones:
   - $A \to B \,\&\, \trianglelefteq K_1 \,\&\, \ldots \,\&\, \trianglelefteq K_t$, for all $(C, \{K_1, \ldots, K_t\}) \in \textsc{Nullable}(G)$;
   - $A \to C \,\&\, \lhd K_1 \,\&\, \ldots \,\&\, \lhd K_t$, for all $(B, \{K_1, \ldots, K_t\}) \in \textsc{Nullable}(G)$;
   - $A \to C \,\&\, \lhd\varepsilon$, if there exists a nonempty set $\{K_1, \ldots, K_t\} \subseteq N$, such that $(B, \{K_1, \ldots, K_t\}) \in \textsc{Nullable}(G)$ and $\langle \varepsilon \rangle \varepsilon \in L_G(K_1) \cap \ldots \cap L_G(K_t)$.

**Lemma 9.** *Let $G = (\Sigma, N, R, S)$ be a grammar with contexts. Then the grammar $G' = (\Sigma, N', R', S)$ constructed above generates the language $L(G) \setminus \{\varepsilon\}$.*

The second stage of the transformation to the normal form is removing the *unit conjuncts* in rules of the form $A \to B \& \ldots$. Already for conjunctive grammars, the only known transformation involves substituting all rules for $B$ into all rules for $A$, and results in a worst-case exponential blowup [11]. The same construction applies for grammars with contexts as it is.

These transformations lead to the following normal form theorem.

**Theorem 10.** *For each grammar with contexts $G = (\Sigma, N, R, S)$ there exists and can be effectively constructed a grammar with contexts $G' = (\Sigma, N', R', S)$ generating the same language, in which all rules are of the form*

$$\begin{aligned}
&A \to B_1C_1 \,\&\, \ldots \,\&\, B_kC_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n \\
&A \to a \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_n \\
&A \to B_1C_1 \,\&\, \ldots \,\&\, B_kC_k \,\&\, \lhd\varepsilon \\
&A \to a \,\&\, \lhd\varepsilon \qquad\qquad\qquad\qquad (k \geqslant 1,\ m, n \geqslant 0,\ a \in \Sigma,\ B_i, C_i, D_i, E_i \in N')
\end{aligned}$$

*The size of the resulting grammar, measured in the total number of symbols used to describe it, is at most exponential in the size of the given grammar.*

## 5 Parsing Algorithm

For each grammar with one-sided contexts in the binary normal form, there exists a parsing algorithm that determines the membership of all substrings of the input string in the languages generated by all nonterminals of the grammar, storing the results in a table. This algorithm elaborates a similar procedure for conjunctive grammars [11], which in turn generalized the Cocke–Kasami–Younger algorithm for standard context-free grammars.

Let $G = (\Sigma, N, R, S)$ be a grammar with contexts in the binary normal form, and let $w = a_1 \ldots a_n$ with $n \geqslant 1$ and $a_k \in \Sigma$ be some string. Let $0 \leqslant i < j \leqslant n$. Define

$$T_{i,j} = \{ A \mid A \in N, \vdash_G [A, \langle a_1 \ldots a_i \rangle a_{i+1} \ldots a_j] \}.$$

This table is constructed by the following algorithm.

```
1:  $T_{0,1} = \{ A \in N \mid A \to a_1 \,\&\, \lhd\varepsilon \in R \}$
2:  for $j = 1, \ldots, n$ do
3:      while $T_{0,j}$ changes do
4:          for all $A \to a \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_{m'} \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_{m''}$ do
5:              if $a_j = a$, $D_1, \ldots, D_{m'} \in T_{0,j-1}$ and $E_1, \ldots, E_{m''} \in T_{0,j}$ then
6:                  $T_{j-1,j} = T_{j-1,j} \cup \{A\}$
7:          for $i = j - 2$ to $0$ do
8:              let $U = \varnothing$ $(U \subseteq N \times N)$
9:              for $k = i + 1$ to $j - 1$ do
10:                 $U = U \cup (T_{i,k} \times T_{k,j})$
11:             for all $A \to B_1C_1 \,\&\, \ldots \,\&\, B_mC_m \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_{m'} \,\&$
                        $\trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_{m''}$ do
12:                 if $(B_1, C_1), \ldots, (B_m, C_m) \in U$, $D_1, \ldots, D_{m'} \in T_{0,i}$ and
                        $E_1, \ldots, E_{m''} \in T_{0,j}$ then
13:                     $T_{i,j} = T_{i,j} \cup \{A\}$
14:             for all $A \to B_1C_1 \,\&\, \ldots \,\&\, B_mC_m \,\&\, \lhd\varepsilon$ do
15:                 if $(B_1, C_1), \ldots, (B_m, C_m) \in U$ and $i = 0$ then
16:                     $T_{i,j} = T_{i,j} \cup \{A\}$
```

Once the table is constructed, the input is accepted if and only if $S \in T_{0,n}$.

**Theorem 11.** *For every grammar with one-sided contexts $G$ in the binary normal form, there exists an algorithm that determines the membership of a given string $w = a_1 \ldots a_n$ in $L(G)$, and does so in time $\mathcal{O}(|G|^2 \cdot n^3)$, using space $\mathcal{O}(|G| \cdot n^2)$.*

## 6 Recognition in Linear Space

The cubic-time algorithm in Section 5 uses quadratic space, as do its context-free and conjunctive prototypes. For conjunctive grammars, the membership of a string can be recognized in linear space and exponential time [13] by using deterministic rewriting of terms of a linearly bounded size. In this section, the latter method is extended to handle the case of grammars with one-sided contexts.

Let $G = (\Sigma, N, R, S)$ be a grammar with one-sided contexts and let $w = a_1 \dots a_n$ be an input string. The linear-space parsing algorithm presented below constructs the sets $T_{0,1}$, $T_{0,2}$, …, $T_{0,n}$ (as in the top row of the table in the algorithm in the last section), with

$$T_{0,i} = \{\, A \in N \mid \langle\varepsilon\rangle a_1 \dots a_i \in L_G(A) \,\}.$$

The membership of each $A \in N$ in each set $T_{0,\ell}$ is determined using term rewriting similar to the one defined for Boolean grammars [13], which operates in exponential time by trying all possible parses. Whenever the grammar refers to a context $\lhd D$, the answer is found in one of the previously computed sets $T_{0,i}$ with $i < \ell$; a reference to a context $\trianglelefteq E$ is resolved by looking in the partially computed set $T_{0,\ell}$. However, this $E$ may not yet have been added to $T_{0,\ell}$, and the procedure may give a false negative answer. Hence, up to $|N|$ iterations (similar to those in line 3 of the cubic-time algorithm) have to be done until all dependencies are propagated and all elements of $T_{0,\ell}$ are found. Once the last set $T_{0,n}$ is constructed, the input string is accepted if $S$ belongs to this set.

**Theorem 12.** *Every language generated by a grammar with one-sided contexts is in* DSPACE($n$).

## 7 Future Work

The new model leaves many theoretical questions to ponder. For instance, is there a parsing algorithm for grammars with one-sided contexts working in less than cubic time? For standard context-free grammars, Valiant [21] discovered an algorithm that offloads the most intensive computations into calls to a Boolean matrix multiplication procedure, and thus can work in time $O(n^\omega)$, with $\omega < 3$; according to the current knowledge on matrix multiplication, $\omega$ can be reduced to 2.376. The main idea of Valiant's algorithm equally applies to Boolean grammars, which can be parsed in time $O(n^\omega)$ as well [17]. However, extending it to grammars with contexts, as defined in this paper, seems to be inherently impossible, because the logical dependencies between the properties of substrings (that is, between the entries of the table $T_{i,j}$) now have a more complicated structure, and the order of calculating these entries apparently rules out grouping multiple operations into Boolean matrix multiplication. However, there might exist a different $o(n^3)$-time parsing strategy for these grammars, which would be interesting to discover.

Another direction is to develop practical parsing algorithms for grammars with one-sided contexts. An obvious technique to try is the recursive descent parsing [16], where *ad hoc* restrictions resembling contexts of the form $\trianglerighteq D\Sigma^*$ have long been used to guide deterministic computation. The Lang–Tomita Generalized LR parsing [15] is worth being investigated as well.

A more general direction for further research is to consider grammars with two-sided contexts, which would allow rules of the form $A \to BC \& \lhd D \& \trianglelefteq E \& \trianglerighteq F \& \rhd G$. Such grammars would implement Chomsky's [2] idea of defining phrase-structure rules applicable in a context in full—which is something that was for the first time properly approached in this paper.

## References

1. Aizikowitz, T., Kaminski, M.: *LR*(0) Conjunctive Grammars and Deterministic Synchronized Alternating Pushdown Automata. In: Kulikov, A., Vereshchagin, N. (eds.) CSR 2011. LNCS, vol. 6651, pp. 345–358. Springer, Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-20712-9_27
2. Chomsky, N.: On certain formal properties of grammars. Information and Control 2(2), 137–167 (1959), http://dx.doi.org/10.1016/S0019-9958(59)90362-6
3. Čulík II, K., Cohen, R.: LR-regular grammars—an extension of LR($k$) grammars. Journal of Computer and System Sciences 7(1), 66–96 (1973), http://dx.doi.org/10.1016/S0022-0000(73)80050-9
4. Ésik, Z., Kuich, W.: Boolean fuzzy sets. International Journal of Foundations of Computer Science 18(6), 1197–1207 (2007), http://dx.doi.org/10.1142/S0129054107005248
5. Ford, B.: Parsing expression grammars: a recognition-based syntactic foundation. In: Proceedings of POPL 2004, Venice, Italy, January 14-16, pp. 111–122 (2004), http://doi.acm.org/10.1145/964001.964011
6. Ginsburg, S., Rice, H.G.: Two families of languages related to ALGOL. Journal of the ACM 9, 350–371 (1962), http://dx.doi.org/10.1145/321127.321132
7. Jeż, A.: Conjunctive grammars can generate non-regular unary languages. International Journal of Foundations of Computer Science 19(3), 597–615 (2008), http://dx.doi.org/10.1142/S012905410800584X
8. Jeż, A., Okhotin, A.: Conjunctive grammars over a unary alphabet: undecidability and unbounded growth. Theory of Computing Systems 46(1), 27–58 (2010), http://dx.doi.org/10.1007/s00224-008-9139-5
9. Jarzabek, S., Krawczyk, T.: LL-regular grammars. Information Processing Letters 4, 31–37 (1975), http://dx.doi.org/10.1016/0020-0190(75)90009-5
10. Kountouriotis, V., Nomikos, C., Rondogiannis, P.: Well-founded semantics for Boolean grammars. Information and Computation 207(9), 945–967 (2009), http://dx.doi.org/10.1016/j.ic.2009.05.002
11. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics 6(4), 519–535 (2001)
12. Okhotin, A.: Conjunctive grammars and systems of language equations. Programming and Computer Science 28(5), 243–249 (2002), http://dx.doi.org/10.1023/A:1020213411126
13. Okhotin, A.: Boolean grammars. Information and Computation 194(1), 19–48 (2004), http://dx.doi.org/10.1016/j.ic.2004.03.006
14. Okhotin, A.: The dual of concatenation. Theoretical Computer Science 345(2-3), 425–447 (2005), http://dx.doi.org/10.1016/j.tcs.2005.07.019
15. Okhotin, A.: Generalized LR parsing algorithm for Boolean grammars. International Journal of Foundations of Computer Science 17(3), 629–664 (2006), http://dx.doi.org/10.1142/S0129054106004029
16. Okhotin, A.: Recursive descent parsing for Boolean grammars. Acta Informatica 44(3-4), 167–189 (2007), http://dx.doi.org/10.1007/s00236-007-0045-0
17. Okhotin, A.: Fast Parsing for Boolean Grammars: A Generalization of Valiant's Algorithm. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 340–351. Springer, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-14455-4_31

18. Okhotin, A., Reitwießner, C.: Conjunctive grammars with restricted disjunction. Theoretical Computer Science 411(26-28), 2559–2571 (2010), http://dx.doi.org/10.1016/j.tcs.2010.03.015
19. Parr, T., Fisher, K.: LL(*): the foundation of the ANTLR parser generator. In: Programming Language Design and Implementation, PLDI 2011, San Jose, USA, June 4-8, pp. 425–436 (2011), http://dx.doi.org/10.1145/1993316.1993548
20. Sikkel, K.: Parsing Schemata. Springer, Heidelberg (1997)
21. Valiant, L.G.: General context-free recognition in less than cubic time. Journal of Computer and System Sciences 10(2), 308–314 (1975), http://dx.doi.org/10.1016/S0022-0000(75)80046-8

# Longest Common Extensions via Fingerprinting

Philip Bille, Inge Li Gørtz, and Jesper Kristensen

Technical University of Denmark, DTU Informatics, Copenhagen, Denmark

**Abstract.** The *longest common extension* (LCE) problem is to preprocess a string in order to allow for a large number of LCE queries, such that the queries are efficient. The LCE value, $LCE_s(i, j)$, is the length of the longest common prefix of the pair of suffixes starting at index $i$ and $j$ in the string $s$. The LCE problem can be solved in linear space with constant query time and a preprocessing of sorting complexity. There are two known approaches achieving these bounds, which use nearest common ancestors and range minimum queries, respectively. However, in practice a much simpler approach with linear query time, no extra space and no preprocessing achieves significantly better average case performance. We show a new algorithm, $\text{FINGERPRINT}_k$, which for a parameter $k$, $1 \leq k \leq \lceil \log n \rceil$, on a string of length $n$ and alphabet size $\sigma$, gives $O(kn^{1/k})$ query time using $O(kn)$ space and $O(kn + \mathit{sort}(n, \sigma))$ preprocessing time, where $\mathit{sort}(n, \sigma)$ is the time it takes to sort $n$ numbers from $\sigma$. Though this solution is asymptotically strictly worse than the asymptotically best previously known algorithms, it outperforms them in practice in average case and is almost as fast as the simple linear time algorithm. On worst case input, this new algorithm is significantly faster in practice compared to the simple linear time algorithm. We also look at cache performance of the new algorithm, and we show that for $k = 2$, cache optimization can improve practical query time.

## 1 Introduction

The *longest common extension* (LCE) problem is to preprocess a string in order to allow for a large number of LCE queries, such that the queries are efficient. The LCE value, $LCE_s(i, j)$, is the length of the longest common prefix of the pair of suffixes starting at index $i$ and $j$ in the string $s$. The LCE problem can be used in many algorithms for solving other algorithmic problems, e.g., the Landau-Vishkin algorithm for approximate string searching [6]. Solutions with linear space, constant query time, and $O(\mathit{sort}(n, \sigma))$ preprocessing time exist for the problem [3,2]. Here $\mathit{sort}(n, \sigma)$ is the time it takes to sort $n$ numbers from an alphabet of size $\sigma$. For $\sigma = O(n^c)$, where $c$ is a constant, we have $\mathit{sort}(n, \sigma) = O(n)$. These theoretically good solutions are however not the best in practice, since they have large constant factors for both query time and space usage. Ilie et al. [4] introduced a much simpler solution with average case constant time and no space or preprocessing required other than storing the input string. This solution has significantly better practical performance for average case input as well as for average case queries on some real world strings, when compared to the asymptotically best known algorithms. However, this algorithm has linear worst case query time, and is thus only ideal when worst case performance is irrelevant. In situations where we need both average case and worst case performance to be good, none of the existing solutions are ideal. An example could be a firewall, which needs to do approximate string searching. The firewall should not allow an attacker to significantly degrade its performance by sending it carefully crafted packages. At the same time it must scan legitimate data quickly. The main goal of this paper is to design a practical algorithm that performs well in both situations, that is, achieves a good worst-case guarantee while maintaining a fast average case performance.

*Previous Results.* Throughout the paper let $s$ be a string of length $n$ over an alphabet of size $\sigma$. Ilie et al. [4] gave an algorithm, DirectComp, for solving the LCE problem, which uses no preprocessing and has $O(\mathit{LCE}(i,j))$ query time. For a query $\mathit{LCE}(i,j)$, the algorithm simply compares $s[i]$ to $s[j]$, then $s[i+1]$ to $s[j+1]$ and so on, until the two characters differ, or the end of the string is reached. The worst case query time is thus $O(n)$. However, on random strings and many real-word texts, Ilie et al. [4] showed that the average LCE is $O(1)$, where the average is over all $\sigma^n n^2$ combinations of strings and query inputs. Hence, in these scenarios DirectComp achieves $O(1)$ query time.

The LCE problem can also be solved with $O(1)$ worst case query time, using $O(n)$ space and $O(\mathit{sort}(n,\sigma))$ preprocessing time. Essentially, two different ways of doing this exists. One method, SuffixNca, uses constant time nearest common ancestor (NCA) queries [3] on a suffix tree. The LCE of two indexes $i$ and $j$ is defined as the length of the longest common prefix of the suffixes $s[i\,..\,n]$ and $s[j\,..\,n]$. In a suffix tree, the path from the root to $L_i$ has label $s[i\,..\,n]$ (likewise for $j$), and no two child edge labels of the same node will have the same first character. The longest common prefix of the two suffixes will therefore be the path label from the root to the nearest common ancestor of $L_i$ and $L_j$, i.e., $\mathit{LCE}_s(i,j) = D[\mathit{NCA}_{\mathcal{T}}(L_i, L_j)]$. The other method, LcpRmq, uses constant time range minimum queries (RMQ) [2] on a longest common prefix (LCP) array. The LCP array contains the length of the longest common prefixes of each pair of neighbor suffixes in the suffix array (SA). The length of the longest common prefix of two arbitrary suffixes in SA can be found as the minimum of all LCP values of neighbor suffixes between the two desired suffixes, because SA lists the suffixes in lexicographical ordering, i.e., $\mathit{LCE}(i,j) = \mathit{LCP}[\mathit{RMQ}_{\mathit{LCP}}(SA^{-1}[i]+1, SA^{-1}[j])]$, where $SA^{-1}[i] < SA^{-1}[j]$. Table 1 summarizes the above theoretical bounds.

*Our Results.* We present a new LCE algorithm, Fingerprint$_k$, based on multiple levels of string fingerprinting. The algorithm has a parameter $k$ in the range $1 \le k \le \lceil \log n \rceil$, which describes the number of levels used[1]. The performance of the algorithm is summarized by the following theorem:

[1] All logarithms are base two.

**Table 1.** LCE algorithms with their space requirements, worst case query times and preprocessing times. Average case query times are $O(1)$ for all shown algorithms. Rows marked with * show the new algorithm we present.

| Algorithm | Space | Query time | Preprocessing |
|---|---|---|---|
| SuffixNca | $O(n)$ | $O(1)$ | $O(sort(n,\sigma))$ |
| LcpRmq | $O(n)$ | $O(1)$ | $O(sort(n,\sigma))$ |
| DirectComp | $O(1)$ | $O(n)$ | None |
| Fingerprint$_k$* | $O(kn)$ | $O(kn^{1/k})$ | $O(sort(n,\sigma)+kn)$ |
| $k=\lceil \log n \rceil$ * | $O(n\log n)$ | $O(\log n)$ | $O(n\log n)$ |

**Theorem 1.** *For a string s of length n and alphabet size $\sigma$, the* Fingerprint$_k$ *algorithm, where k is a parameter $1 \le k \le \lceil \log n \rceil$, can solve the LCE problem in $O(kn^{1/k})$ worst case query time and $O(1)$ average case query time using $O(kn)$ space and $O(sort(n,\sigma)+kn)$ preprocessing time.*

By choosing $k$ we can obtain the following interesting tradeoffs.

**Corollary 2.** Fingerprint$_1$ *is equivalent to* DirectComp *with $O(n)$ space and $O(n)$ query time.*

**Corollary 3.** Fingerprint$_2$ *uses $O(n)$ space and $O(\sqrt{n})$ query time.*

**Corollary 4.** Fingerprint$_{\lceil \log n \rceil}$ *uses $O(n\log n)$ space and $O(\log n)$ query time.*

The latter result is equivalent to the classic Karp-Miller-Rosenberg fingerprint scheme [5]. To preprocess the $O(kn)$ fingerprints used by our algorithm, we can use Karp-Miller-Rosenberg [5], which takes $O(n\log n)$ time. For $k = o(\log n)$, we can speed up preprocessing to $O(sort(n,\sigma)+kn)$ by using the SA and LCP arrays.

In practice, existing state of the art solutions are either good in worst case, while poor in average case (LcpRmq), or good in average case while poor in worst case (DirectComp). Our Fingerprint$_k$ solution targets a worst case vs. average case query time tradeoff between these two extremes. Our solution is almost as fast as DirectComp on an average case input, and it is significantly faster than DirectComp on a worst case input. Compared to LcpRmq, our solution has a significantly better performance on an average case input, but its worst case performance is not as good as that of LcpRmq. The space usage for LcpRmq and Fingerprint$_k$ are approximately the same when $k = 6$.

For $k = 2$ we can improve practical Fingerprint$_k$ query time even further by optimizing it for cache efficiency. However for $k > 2$, this cache optimization degrades practical query time performance, as the added overhead outweighs the improved cache efficiency.

Our algorithm is fairly simple. Though it is slightly more complicated than DirectComp, it does not use any of the advanced algorithmic techniques required by LcpRmq and SuffixNca.

## 2 Preliminaries

Let $s$ be a string of length $n$. Then $s[i]$ is the $i$'th character of $s$, and $s[i\,..\,j]$ is a substring of $s$ containing characters $s[i]$ to $s[j]$, both inclusive. That is, $s[1]$ is the first character of $s$, $s[n]$ is the last character, and $s[1\,..\,n]$ is the entire string. The suffix of $s$ starting at index $i$ is written $\mathit{suff}_i = s[i\,..\,n]$.

A *suffix tree* $\mathcal{T}$ encodes all suffixes of a string $s$ of length $n$ with alphabet $\sigma$. The tree has $n$ leaves named $L_1$ to $L_n$, one for each suffix of $s$. Each edge is labeled with a substring of $s$, such that for any $1 \leq i \leq n$, the concatenation of labels on edges on the path from the root to $L_i$ gives $\mathit{suff}_i$. Any internal node must have more than one child, and the labels of two child edges must not share the same first character. The string depth $D[v]$ of a node $v$ is the length of the string formed when concatenating the edge labels on the path from the root to $v$. The tree uses $O(n)$ space, and building it takes $O(\mathit{sort}(n, \sigma))$ time [1].

For a string $s$ of length $n$ with alphabet size $\sigma$, the *suffix array* ($SA$) is an array of length $n$, which encodes the lexicographical ordering of all suffixes of $s$. The lexicographically smallest suffix is $\mathit{suff}_{SA[1]}$, the lexicographically largest suffix is $\mathit{suff}_{SA[n]}$, and the lexicographically $i$'th smallest suffix is $\mathit{suff}_{SA[i]}$. The *inverse suffix array* ($\mathrm{SA}^{-1}$) describes where a given suffix is in the lexicographical order. Suffix $\mathit{suff}_i$ is the lexicographically $SA^{-1}[i]$'th smallest suffix.

The *longest common prefix array* (LCP array) describes the length of longest common prefixes of neighboring suffixes in SA. The length of the longest common prefix of $\mathit{suff}_{SA[i-1]}$ and $\mathit{suff}_{SA[i]}$ is $LCP[i]$, for $2 \leq i \leq n$. The first element $LCP[1]$ is always zero. Building the SA, $\mathrm{SA}^{-1}$ and LCP arrays takes $O(\mathit{sort}(n, \sigma))$ time [1].

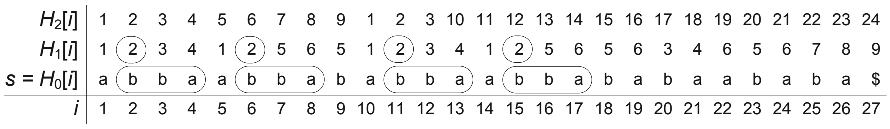
The *nearest common ancestor* (NCA) of two nodes $u$ and $v$ in a tree is the node of greatest depth, which is an ancestor of both $u$ and $v$. The ancestors of a node $u$ includes $u$ itself. An NCA query can be answered in $O(1)$ time with $O(n)$ space and preprocessing time in a static tree with $n$ nodes [3].

The range minimum of $i$ and $j$ on an array $A$ is the index of a minimum element in $A[i, j]$, i.e., $RMQ_A(i, j) = \arg\min_{k \in \{i, \ldots, j\}}\{A[k]\}$. A *range minimum query* (RMQ) on a static array of $n$ elements can be answered in $O(1)$ time with $O(n)$ space and preprocessing time [2].

The *I/O model* describes the number of memory blocks an algorithm moves between two layers of a layered memory architecture, where the size of the internal memory layer is $M$ words, and data is moved between internal and external memory in blocks of $B$ words. In the *cache-oblivious model*, the algorithm has no knowledge of the values of $M$ and $B$.

## 3 The Fingerprint$_k$ Algorithm

Our Fingerprint$_k$ algorithm generalizes DirectComp. It compares characters starting at positions $i$ and $j$, but instead of comparing individual characters, it compares fingerprints of substrings. Given fingerprints of all substrings of length $t$, our algorithm can compare two $t$-length substrings in constant time.



| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $H_2[i]$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| $H_1[i]$ | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 6 | 5 | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 6 | 5 | 6 | 3 | 4 | 6 | 5 | 6 | 7 | 8 | 9 |
| $s = H_0[i]$ | a | b | b | a | a | b | b | a | b | a | b | b | a | a | b | b | a | b | a | b | a | a | b | a | b | a | $ |
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |

**Fig. 1.** FINGERPRINT$_k$ data structure for $s = abbaabbababbaabbababaababa\$$, $n = 27$, $k = 3$, $t_1 = n^{1/3} = 3$ and $t_2 = n^{2/3} = 9$. All substrings $bba$ are highlighted with their 3-length fingerprint 2.

## 3.1 Data Structure

Given a string $s$, the fingerprint $F_t[i]$ is a natural number identifying the substring $s[i\,..\,i+t-1]$ among all $t$-length substrings of $s$. We assign fingerprints such that for any $i$, $j$ and $t$, $F_t[i] = F_t[j]$ if and only if $s[i\,..\,i+t-1] = s[j\,..\,j+t-1]$. In other words, if two substrings of $s$ have the same length, they have the same fingerprints if and only if the substrings themselves are the same.

At the end of a string when $i + t - 1 > n$, we define $F_t[i]$ by adding extra characters to the end of the string as needed. The last character of the string must be a special character \$, which does not occur anywhere else in the string.

The FINGERPRINT$_k$ data structure for a string $s$ of length $n$, where $k$ is a parameter $1 \le k \le \lceil \log n \rceil$, consists of $k$ natural numbers $t_0$, ..., $t_{k-1}$ and $k$ tables $H_0$, ..., $H_{k-1}$, each of length $n$. For each $\ell$ where $0 \le \ell \le k-1$, $t_\ell = \Theta(n^{\ell/k})$ and table $H_\ell$ contains fingerprints of all $t_\ell$-length substrings of $s$, such that $H_\ell[i] = F_{t_\ell}[i]$. We always have $t_0 = n^{0/k} = 1$, such that $H_0$ is the original string $s$. An example is shown in Fig. 1. Since each of the $k$ tables stores $n$ fingerprints of constant size, we get the following.

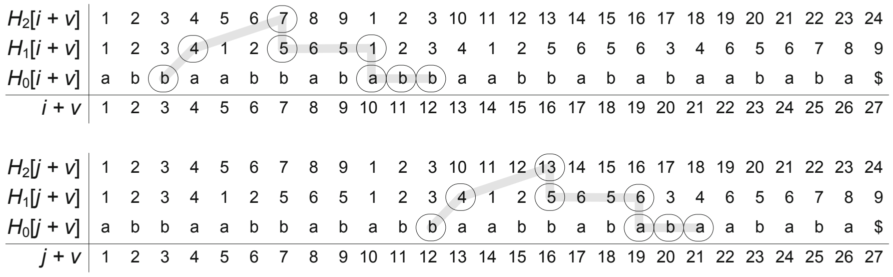
**Lemma 5.** *The* FINGERPRINT$_k$ *data structure takes* $O(kn)$ *space.*

## 3.2 Query

The FINGERPRINT$_k$ query speeds up DIRECTCOMP by comparing fingerprints of substrings of the input string instead of individual characters. The query algorithm consists of two traversals of the hierarchy of fingerprints. In the first traversal the algorithm compares progressively larger fingerprints of substrings until a mismatch is found and in the second traversal the algorithm compares progressively smaller substrings to find the precise point of the mismatch. The combination of these two traversals ensures both a fast worst case and average case performance.

The details of the query algorithm are as follows. Given the FINGERPRINT$_k$ data structure, start with $v = 0$ and $\ell = 0$, then do the following steps:

1. As long as $H_\ell[i+v] = H_\ell[j+v]$, increment $v$ by $t_\ell$, increment $\ell$ by one, and repeat this step unless $\ell = k-1$.
2. As long as $H_\ell[i+v] = H_\ell[j+v]$, increment $v$ by $t_\ell$ and repeat this step.
3. Stop and return $v$ when $\ell = 0$, otherwise decrement $\ell$ by one and go to step two.



| $H_2[i+v]$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $H_1[i+v]$ | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 6 | 5 | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 6 | 5 | 6 | 3 | 4 | 6 | 5 | 6 | 7 | 8 | 9 |
| $H_0[i+v]$ | a | b | b | a | a | b | b | a | b | a | b | b | a | a | b | b | a | b | a | b | a | a | b | a | b | a | $ |
| $i+v$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |

| $H_2[j+v]$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $H_1[j+v]$ | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 6 | 5 | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 6 | 5 | 6 | 3 | 4 | 6 | 5 | 6 | 7 | 8 | 9 |
| $H_0[j+v]$ | a | b | b | a | a | b | b | a | b | a | b | b | a | a | b | b | a | b | a | b | a | a | b | a | b | a | $ |
| $j+v$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |

**Fig. 2.** FINGERPRINT$_k$ query for $LCE(3, 12)$ on the data structure of Fig. 1. The top half shows how $H_\ell[i+v]$ moves through the data structure, and the bottom half shows $H_\ell[j+v]$.

An example of a query is shown in Fig. 2.

**Lemma 6.** *The* FINGERPRINT$_k$ *query algorithm is correct.*

*Proof.* At each step of the algorithm $v \leq LCE(i, j)$, since the algorithm only increments $v$ by $t_\ell$ when it has found two matching fingerprints, and fingerprints of two substrings of the same length are only equal if the substrings themselves are equal. When the algorithm stops, it has found two fingerprints, which are not equal, and the length of these substrings is $t_\ell = 1$, therefore $v = LCE(i, j)$.

The algorithm never reads $H_\ell[x]$, where $x > n$, because the string contains a unique character \$ at the end. This character will be at different positions in the substrings whose fingerprints are the last $t_\ell$ elements of $H_\ell$. These $t_\ell$ fingerprints will therefore be unique, and the algorithm will not continue at level $\ell$ after reading one of them. □

**Lemma 7.** *The worst case query time for* FINGERPRINT$_k$ *is* $O(kn^{1/k})$*, and the average case query time is* $O(1)$*.*

*Proof.* First we consider the worst case. Step one takes $O(k)$ time. In step two and three, the number of remaining characters left to check at level $\ell$ is $O(n^{(\ell+1)/k})$, since the previous level found two differing substrings of that length (at the top level $\ell = k-1$ we have $O(n^{(\ell+1)/k}) = O(n)$). Since we can check $t_\ell = \Theta(n^{\ell/k})$ characters in constant time at level $\ell$, the algorithm uses $O(n^{(\ell+1)/k})/\Theta(n^{\ell/k}) = O(n^{1/k})$ time at that level. Over all $k$ levels, $O(kn^{1/k})$ query time is used.

Next we consider the average case. At each step except step three, the algorithm increments $v$. Step three is executed the same number of times as step one, in which $v$ is incremented. The query time is therefore linear in the number of times $v$ is incremented, and it is thereby $O(v)$. From the proof of Lemma 6 we have $v = LCE(i, j)$. By Ilie et al. [4] the average $LCE(i, j)$ is $O(1)$ and hence the average case query time is $O(1)$. □

*Average Case vs. Worst Case* The first traversal in the query algorithm guarantees $O(1)$ average case performance. Without it, i.e., if the query started with $\ell = k-1$ and omitted step one, the average case query time would be $O(k)$. However, the worst case bound would remain $O(kn^{1/k})$. Thus, for a better practical worst-case performance we could omit the first traversal entirely. We have extensively experimented with both variants and we found that in nearly all scenarios the first traversal improved the overall performance. In the cases where performance was not improved the first traversal only degraded the performance slightly. We therefore focus exclusively on the two traversal variant in the remainder of the paper.

## 3.3 Preprocessing

The tables of fingerprints use $O(kn)$ space. In the case with $k = \lceil \log n \rceil$ levels, the data structure is the one generated by Karp-Miller-Rosenberg [5]. This data structure can be constructed in $O(n \log n)$ time. With $k < \lceil \log n \rceil$ levels, KMR can be adapted, but it still uses $O(n \log n)$ preprocessing time.

We can preprocess the data structure in $O(sort(n, \sigma) + kn)$ time using the SA and LCP arrays. First create the SA and LCP arrays. Then preprocess each of the $k$ levels using the following steps. An example is shown in Fig. 3.

1. Loop through the $n$ substrings of length $t_\ell$ in lexicographically sorted order by looping through the elements of SA.
2. Assign an arbitrary fingerprint to the first substring.
3. If the current substring $s[SA[i] \mathinner{..} SA[i]+t_\ell-1]$ is equal to the substring examined in the previous iteration of the loop, give the current substring the same fingerprint as the previous substring, otherwise give the current substring a new unused fingerprint. The two substrings are equal when $LCE[i] \geq t_\ell$.

**Lemma 8.** *The preprocessing algorithm described above generates the data structure described in Sect. 3.1.*

*Proof.* We always assign two different fingerprints whenever two substrings are different, because whenever we see two differing substrings, we change the fingerprint to a value not previously assigned to any substring.

We always assign the same fingerprint whenever two substrings are equal, because all substrings, which are equal, are grouped next to each other, when we loop through them in lexicographical order. □

**Lemma 9.** *The preprocessing algorithm described above takes $O(sort(n, \sigma) + kn)$ time.*

*Proof.* We first construct the SA and LCP arrays, which takes $O(sort(n, \sigma))$ time [1]. We then preprocess each of the $k$ levels in $O(n)$ time, since we loop through $n$ substrings, and comparing neighboring substrings takes constant time when we use the LCP array. The total preprocessing time becomes $O(sort(n, \sigma) + kn)$. □

| $i$ | SA[$i$] | $suff_{SA[i]}$ | $s$[SA[$i$]..SA[$i$]+3] | $H_l$[SA[$i$]] |
|---|---|---|---|---|
| 1 | 9 | a | a | 1 |
| 2 | 4 | ababba | aba | 2 |
| 3 | 6 | abba | abb | 3 |
| 4 | 1 | abbababba | abb | 3 |
| 5 | 8 | ba | ba | 5 |
| 6 | 3 | bababba | bab | 6 |
| 7 | 5 | babba | bab | 6 |
| 8 | 7 | bba | bba | 8 |
| 9 | 2 | bbababba | bba | 8 |

**Fig. 3.** The first column lists all substrings of $s = abbababba$ with length $t_\ell = 3$. The second column lists fingerprints assigned to each substring. The third column lists the position of each substring in $s$.

## 4 Experimental Results

In this section we show results of actual performance measurements. The measurements were done on a Windows 23-bit machine with an Intel P8600 CPU (3 MB L2, 2.4 GHz) and 4 GB RAM. The code was compiled using GCC 4.5.0 with `-O3`.

### 4.1 Tested Algorithms

We implemented different variants of the Fingerprint$_k$ algorithm in C++ and compared them with optimized versions of the DirectComp and LcpRmq algorithms. The algorithms we compared are the following:

**DirectComp** is the simple DirectComp algorithm with no preprocessing and worst case $O(n)$ query time.

**Fingerprint$_k$<$t_{k-1}$, ..., $t_1$>ac** is the Fingerprint$_k$ algorithm using $k$ levels, where $k$ is 2, 3 and $\lceil \log n \rceil$. The numbers <$t_{k-1}$, ..., $t_1$> describe the exact size of fingerprinted substrings at each level.

**RMQ<$n$, 1>** is the LcpRmq algorithm using constant time RMQ.

### 4.2 Test Inputs and Setup

We have tested the algorithms on different kinds of strings:

**Average case strings.** These strings have many small LCE values, such that the average LCE value over all $n^2$ query pairs is less than one. We use results on these strings as an indication average case query times over all input pairs $(i, j)$ in cases where most or all LCE values are small on expected input strings. We construct these strings by choosing each character uniformly at random from an alphabet of size 10.

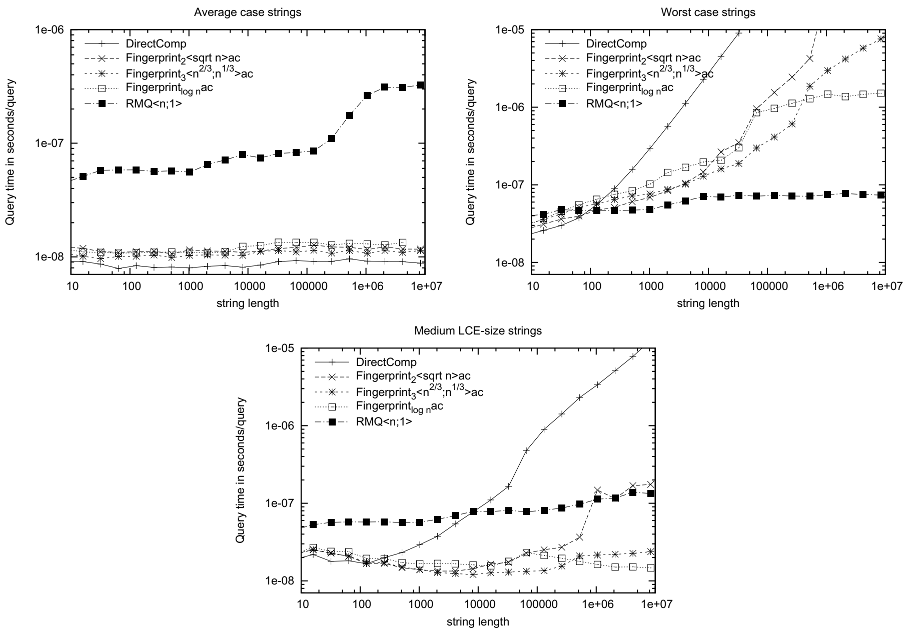


**Fig. 4.** Comparison of our new FINGERPRINT$_k$ algorithm for $k = 2$, $k = 3$ and $k = \lceil \log n \rceil$ versus the existing DIRECTCOMP and LCPRMQ algorithms

**Worst case strings.** These strings have many large LCE values, such that the average LCE value over all $n^2$ query pairs is $n/2$. We use results on these strings as an indication of worst case query times, since the query times for all tested algorithms are asymptotically at their worst when the LCE value is large. We construct these strings with an alphabet size of one.

**Medium LCE value strings.** These strings have an average LCE value over all $n^2$ query pairs of $n/2r$, where $r = 0.73n^{0.42}$. These strings where constructed to show that there exists input strings where FINGERPRINT$_k$ is faster than both DIRECTCOMP and LCPRMQ at the same time. The strings consist of repeating substrings of $r$ unique characters. The value of $r$ was found experimentally.

Each measurement we make is an average of query times over a million random query pairs $(i, j)$. For a given string length and string type we use the same string and the same million query pairs on all tested algorithms.

## 4.3 Results

Fig. 4 shows our experimental results on average case strings with a small average LCE value, worst case strings with a large average LCE value, and strings with a medium average LCE value.

**Table 2.** Query times in nano seconds for DirectComp (DC), Fingerprint$_k$ (FP$_k$) and LcpRmq (RMQ) on the five largest files from the Canterbury corpus

| File | $n$ | $\sigma$ | DC | FP$_2$ | FP$_3$ | FP$_{\log n}$ | RMQ |
|---|---:|---:|---:|---:|---:|---:|---:|
| book1 | $0.7 \cdot 2^{20}$ | 82 | 8.1 | 11.4 | 10.6 | 12.0 | 218.0 |
| kennedy.xls | $1.0 \cdot 2^{20}$ | 256 | 11.9 | 16.0 | 16.1 | 18.6 | 114.4 |
| E.coli | $4.4 \cdot 2^{20}$ | 4 | 12.7 | 16.5 | 16.6 | 19.2 | 320.0 |
| bible.txt | $3.9 \cdot 2^{20}$ | 63 | 8.5 | 11.3 | 10.5 | 12.6 | 284.0 |
| world192.txt | $2.3 \cdot 2^{20}$ | 93 | 7.9 | 10.5 | 9.8 | 12.7 | 291.7 |

On average case strings, our new Fingerprint$_k$ algorithm is approximately 20% slower than DirectComp, and it is between than 5 and 25 times faster than LcpRmq. We see the same results on some real world strings in Table 2.

On worst case strings, the Fingerprint$_k$ algorithms are significantly better than DirectComp and somewhat worse than LcpRmq. Up until $n = 30,000$ the three measured Fingerprint$_k$ algorithms have nearly the same query times. Of the Fingerprint$_k$ algorithms, the $k = 2$ variant has a slight advantage for small strings of length less than around $2,000$. For longer strings the $k = 3$ variant performs the best up to strings of length $250,000$, at which point the $k = \lceil \log n \rceil$ variant becomes the best. This indicates that for shorter strings, using fewer levels is better, and when the input size increases, the Fingerprint$_k$ variants with better asymptotic query times have better worst case times in practice.

On the plot of strings with medium average LCE values, we see a case where our Fingerprint$_k$ algorithms are faster than both DirectComp and LcpRmq.

We conclude that our new Fingerprint$_k$ algorithm achieves a tradeoff between worst case times and average case times, which is better than the existing best DirectComp and LcpRmq algorithms, yet it is not strictly better than the existing algorithms on all inputs. Fingerprint$_k$ is therefore a good choice in cases where both average case and worst case performance is important.

LcpRmq shows a significant jump in query times around $n = 1,000,000$ on the plot with average case strings, but not on the plot with worst case strings. We have run the tests in Cachegrind, and found that the number of instructions executed and the number of data reads and writes are exactly the same for both average case strings and worst case strings. The cache miss rate for average case strings is 14% and 9% for the L1 and L2 caches, and for worst case strings the miss rate is 17% and 13%, which is the opposite of what could explain the jump we see in the plot.

## 4.4 Cache Optimization

The amount of I/O used by Fingerprint$_k$ is $O(kn^{1/k})$. However if we structure our tables of fingerprints differently, we can improve the number of I/O operations to $O(k(n^{1/k}/B + 1))$ in the cache-oblivious model. Instead of storing $F_{t_\ell}[i]$ at $H_\ell[i]$, we can store it at $H_\ell[((i-1) \mod t_\ell) \cdot \lceil n/t_\ell \rceil + \lfloor (i-1)/t_\ell \rfloor + 1]$. This will group all used fingerprints at level $\ell$ next to each other in memory, such that the amount of I/O at each level is reduced from $O(n^{1/k})$ to $O(n^{1/k}/B)$.

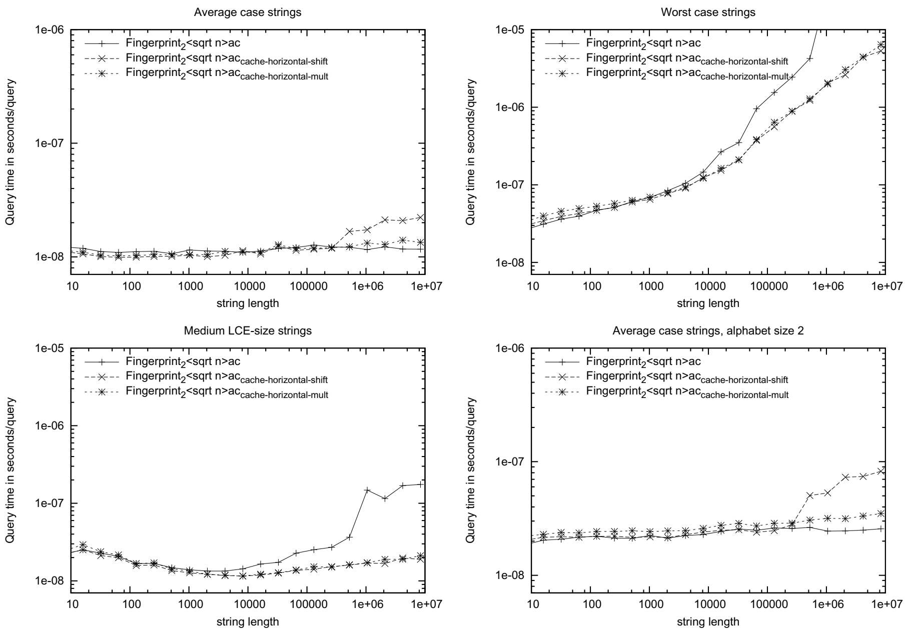


**Fig. 5.** Query times of FINGERPRINT$_2$ with and without cache optimization

The size of each fingerprint table will grow from $|H_\ell| = n$ to $|H_\ell| = n + t_\ell$, because the rounding operations may introduce one-element gaps in the table after every $n/t_\ell$ elements. We achieve the greatest I/O improvement when $k$ is small. When $k = \lceil \log n \rceil$, this cache optimization gives no asymptotic difference in the amount of I/O.

We have implemented two cache optimized variants. One as described above, and one where multiplication, division and modulo is replaced with shift operations. To use shift operations, $t_\ell$ and $\lceil n/t_\ell \rceil$ must both be powers of two. This may double the size of the used address space.

Fig. 5 shows our measurements for FINGERPRINT$_2$. On average case strings the cache optimization does not change the query times, while on worst case strings and strings with medium size LCE values, cache optimization gives a noticeable improvement for large inputs. The cache optimized FINGERPRINT$_2$ variant with shift operations shows an increase in query times for large inputs, which we cannot explain. The last plot on Fig. 5 shows a variant of average case where the alphabet size is changed to two. This plot shows a bad case for cache optimized FINGERPRINT$_2$. LCE values in this plot are large enough to ensure that $H_1$ is used often, which should make the extra complexity of calculating indexes into $H_1$ visible. At the same time the LCE values are small enough to ensure, that the cache optimization has no effect. In this bad case plot we see that the cache optimized variant of FINGERPRINT$_2$ has only slightly worse query time compared to the variant, which is not cache optimized.

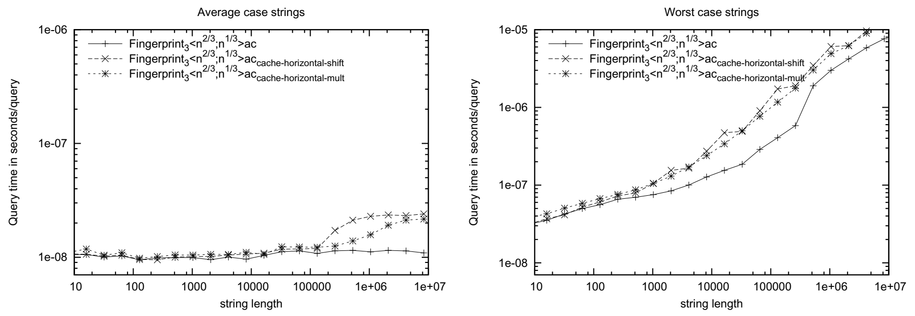


**Fig. 6.** Query times of FINGERPRINT$_3$ with and without cache optimization

Fig. 6 shows the measurements for FINGERPRINT$_3$. Unlike FINGERPRINT$_2$, the cache optimized variant is slightly slower than the unoptimized variant. Hence, our cache optimization is effective for $k = 2$ but not $k = 3$.

## 5 Conclusions

We have presented the FINGERPRINT$_k$ algorithm achieving the theoretical bounds of Thm. 1. We have demonstrated that the algorithm is able to achieve a balance between practical worst case and average case query times. It has almost as good average case query times as DIRECTCOMP, its worst case query times are significantly better than those of DIRECTCOMP, and we have shown cases between average and worst case where FINGERPRINT$_k$ is better than both DIRECTCOMP and LCPRMQ. FINGERPRINT$_k$ gives a good time space tradeoff, and it uses less space than LCPRMQ when $k$ is small.

## References

1. Farach-Colton, M., Ferragina, P., Muthukrishnan, S.: On the sorting-complexity of suffix tree construction. J. ACM 47(6), 987–1011 (2000)
2. Fischer, J., Heun, V.: Theoretical and Practical Improvements on the RMQ-Problem, with Applications to LCA and LCE. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 36–48. Springer, Heidelberg (2006)
3. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. 13(2), 338–355 (1984)
4. Ilie, L., Navarro, G., Tinta, L.: The longest common extension problem revisited and applications to approximate string searching. J. Disc. Alg. 8(4), 418–428 (2010)
5. Karp, R.M., Miller, R.E., Rosenberg, A.L.: Rapid identification of repeated patterns in strings, trees and arrays. In: Proc. 4th Symp. on Theory of Computing, pp. 125–136 (1972)
6. Landau, G.M., Vishkin, U.: Introducing efficient parallelism into approximate string matching and a new serial algorithm. In: Proc. 18th Symp. on Theory of Computing, pp. 220–230 (1986)

# Fast and Cache-Oblivious Dynamic Programming with Local Dependencies

Philip Bille and Morten Stöckel

Technical University of Denmark, DTU Informatics, Copenhagen, Denmark

**Abstract.** String comparison such as sequence alignment, edit distance computation, longest common subsequence computation, and approximate string matching is a key task (and often computational bottleneck) in large-scale textual information retrieval. For instance, algorithms for sequence alignment are widely used in bioinformatics to compare DNA and protein sequences. These problems can all be solved using essentially the same dynamic programming scheme over a two-dimensional matrix, where each entry depends locally on at most 3 neighboring entries. We present a simple, fast, and cache-oblivious algorithm for this type of local dynamic programming suitable for comparing large-scale strings. Our algorithm outperforms the previous state-of-the-art solutions. Surprisingly, our new simple algorithm is competitive with a complicated, optimized, and tuned implementation of the best cache-aware algorithm. Additionally, our new algorithm generalizes the best known theoretical complexity trade-offs for the problem.

## 1 Introduction

Algorithms for string comparison problems such as sequence alignment, edit distance computation, longest common subsequence computation, and approximate string matching are central primitives in large-scale textual information retrieval tasks. For instance, algorithms for sequence alignment are widely used in bioinformatics for comparing DNA and protein sequences.

All of these problems can be solved using essentially the same dynamic programming scheme over a two-dimensional matrix [12]. The common feature of the dynamic programming solution is that each entry $(i, j)$ in the matrix can be computed in constant time given values of the neighboring entries $(i-1, j)$, $(i-1, j-1)$, and $(i, j-1)$ and the characters at position $i$ and $j$ in the input strings. Combined we refer to these problems as *local dynamic programming string comparison problems.*

In this paper, we revisit local dynamic programming string comparison problems for large-scale input strings. We focus on worst-case and exact solutions, however, the techniques presented are straightforward to combine with the typical heuristic or inexact solutions that filter out parts of the dynamic programming matrix which do not need to be computed. In the context of large-scale strings, I/O efficiency and space usage are key issues for obtaining a fast practical solution.

Our main result is a new simple and cache-oblivious algorithm that outperforms the previous state-of-the-art solutions. Surprisingly, our new simple algorithm is competitive with a complicated, optimized, and tuned implementation of the best cache-aware algorithm. Furthermore, our new algorithm generalizes the best known theoretical trade-offs between time, space, and I/O complexity for the problem.

### 1.1 Memory Models

The memory in modern computers is typically organized in a hierarchy of caches, main memory, and disks. The access time to memory significantly increases with each level of the hierarchy. The *external memory model* [1] abstracts this hierarchy by a simple two-level model consisting of an internal memory of size $M$ and an external memory for storing all remaining data. Data can be transferred between external and internal memory in contiguous blocks of size $B$, and all data must be in internal memory before it can be manipulated. The I/O complexity of an algorithm is the number of transfers of blocks between internal and external memory, called I/O operations (or just I/Os).

The *cache-oblivious model* [11] is an extension of the external memory model with the feature that algorithms do not use knowledge of $M$ and $B$. The model assumes an optimal offline cache replacement strategy, which can be approximated within a small constant factor by standard online cache replacements algorithms such as LRU and FIFO. These properties make cache-oblivious algorithms both I/O efficient on all levels of the memory hierarchy simultaneously and portable between hardware architectures with different memory hierarchies.

### 1.2 Previous Results

Let $X$ and $Y$ be the input strings to a local dynamic programming string comparison problem. For simplicity in the presentation, we assume that $|X| = |Y| = n$. All solutions in this paper are based on two passes over an $(n+1) \times (n+1)$ dynamic programming matrix (DPM). First, a *forward pass* computes the length of an optimal path from the top-left corner to the bottom-right corner, and then a *backward pass* computes the actual path by backtracking in the DPM. Finally, the path is translated to the solution to the specific string comparison problem. An overview of the complexities of the previous bounds and of our new algorithm is listed in Table 1.

The first dynamic programming solution is due to Wagner and Fischer [21]. Here, the forward pass fills in and stores all entries in the DPM using $O(n^2)$ time and space. The backward pass then uses the stored entries to efficiently backtrack in $O(n)$ time. In total the algorithm uses $O(n^2)$ time and space. The space usage of this algorithm makes it unsuitable in practice for the sizes of strings that we consider.

Hirschberg [14] showed how to improve the space at the cost of increasing the time for the backward pass. The key idea is to not store all values of the DPM, but instead use a divide and conquer approach in the backward pass to

**Table 1.** Comparison of different algorithms for local dynamic programming. FullMatrix is Wagner and Fischer's algorithm [21], Hirschberg is the linear space algorithm by Hirschberg [14], FastLSA$_k$ is the algorithm by Driga et al. [9,10] with parameter $k$, CO is the cache-oblivious algorithm by Chowdhury et al. [4,6], and FastCO$_k$ is our new algorithm with parameter $k$.

| Algorithm | Forward Pass | Backward Pass | Space | I/O | CO |
|---|---|---|---|---|---|
| FullMatrix | $O\left(n^2\right)$ | $O(n)$ | $O\left(n^2\right)$ | $O\left(\frac{n^2}{B}+\frac{n}{B}+1\right)$ | Yes |
| Hirschberg | $O\left(n^2\right)$ | $O\left(n^2\right)$ | $O(n)$ | $O\left(\frac{n^2}{B}+\frac{n}{B}+1\right)$ | Yes |
| FastLSA$_k$ | $O\left(n^2\right)$ | $O\left(\frac{n^2}{k}+n\right)$ | $O(nk+D)$ | $O\left(\frac{n^2}{B}+\frac{n}{B}+1\right)$ | No |
| CO | $O\left(n^2\right)$ | $O\left(n^2\right)$ | $O(n)$ | $O\left(\frac{n^2}{BM}+\frac{n}{B}+1\right)$ | Yes |
| FastCO$_k$ | $O\left(n^2\right)$ | $O\left(\frac{n^2}{k}+n\right)$ | $O(nk)$ | $O\left(\frac{n^2k}{BM}+\frac{n}{B}+1\right)$ | Yes |

reconstruct the path. At each step in the backward pass the algorithm splits the DPM into two halves. In each half the algorithm recursively finds an optimal path. The algorithm then combines paths for each half into an optimal path for the entire DPM. The total time for the backward pass increases to $O(n^2)$. Hence, in total the algorithm uses $O(n^2)$ time and $O(n)$ space. Myers and Miller [18] popularized Hirschberg's algorithm for sequence alignment in bioinformatics, and it has since been widely used in practice.

More recently, Driga et al. [9,10] proposed an advanced recursive algorithm, called FastLSA. The overall idea is to divide the DPM into $k^2$ submatrices, where $k$ is a tunable parameter defined by the user. The forward pass computes and stores the entries of the input boundaries of the submatrices, i.e., the row and column immediately above and to the left of the submatrix. This uses $O(n^2)$ time as the previous algorithms, but now additional $O(nk)$ space is used to store the input boundaries. The backward pass uses the stored input boundaries to speed up the computation of the optimal path by processing only the submatrices that intersect the optimal path. The submatrices are processed recursively until their size is below another user defined threshold $D$. Submatrices of size less than $D$ are processed using Wagner and Fischer's algorithm. The parameter $D$ is chosen such that space for the full matrix algorithm is sufficiently small to fit in a fast cache. With the additional stored input boundaries the time for the backward pass is improved to $O(\frac{n^2}{k}+n)$. In total the algorithm uses $O(n^2+\frac{n^2}{k}+n) = O(n^2)$ time and $O(nk+D)$ space. In addition to the basic recursive idea, the algorithm implements several advanced optimizations to improve the practical running time of the backward pass. For instance, the sizes of the submatrices in recursive calls are reduced according to the entry point of the optimal path in the submatrix and the allocation, deallocation, and caching of the additional space is handled in a non-trivial way. The resulting full algorithm is substantially more complicated than Hirschberg's algorithm.

In practice, Driga et al. [9,10] only consider strings of lengths $\leq 2000$ and in this case they showed that their solution outperforms both Wagner and Firscher's algorithm and Hirschberg's algorithm. For large strings we found that the original implementation was not competitive with the other algorithms. However, by optimizing and simplifying the implementation in the spirit of our new algorithm (see Sec. 2), we were able to obtain a fast and competitive algorithm suitable for large strings.

In terms of I/O complexity all of the above algorithms use $O(\frac{n^2}{B} + \frac{n}{B} + 1)$ I/Os. Furthermore, the algorithm by Driga et al. [9,10] is cache-aware since it needs to know the parameters of the memory hierarchy in order to optimally select the threshold $D$.

Chowdhury et al. [4,6] gave a cache-oblivious algorithm that significantly improves this I/O bound. The key idea is to split the DPM into 4 submatrices and apply a simple divide and conquer approach in both passes. The forward pass computes and stores the input boundaries of the 4 submatrices similar to the algorithm by Driga et al. [9,10], however, the computation is now done recursively on each submatrix. This uses $O(n^2)$ time and $O(n)$ space. The backward pass recursively processes the submatrices that intersect the optimal path. This also uses $O(n^2)$ time and $O(n)$ space. Chowdhury et al. [4,6] showed that the total number of I/Os incurred by the algorithm is $O(\frac{n^2}{MB} + \frac{n}{B} + 1)$. Compared to the previous results, this improves the number of I/Os in the leading quadratic term by a factor $M$. Furthermore, they also showed that this bound is optimal in the sense that any implementation of the local dynamic programming algorithm must use at least this many I/Os. In practice, the reduced number of I/Os significantly improve upon the performance of Hirschberg's algorithm on large strings. To the best of our knowledge, this algorithm is the fastest known practical solution on large strings. Furthermore, the full algorithm is nearly as simple as Hirschberg's algorithm.

The above bounds represent the best known worst-case complexities for general local dynamic programming string comparison. If we restrict the problem in terms of alphabet size or cost function or if we use the properties of a specific local dynamic programming string comparison problem better bounds are known, see e.g., [2,3,7,8,13,15–17,19] and also the survey [20].

### 1.3 Our Results

We present a simple new algorithm with the following complexity.

**Theorem 1.** *Let $X$ and $Y$ be strings of length $n$. Given any integer parameter $k$, $2 \leq k \leq n$, we can solve any local dynamic programming string comparison problem for $X$ and $Y$ using $O(n^2)$ time for the forward pass, $O(\frac{n^2}{k})$ time for the backward pass, and $O(nk)$ space. Furthermore, the algorithm uses $O(\frac{n^2 k}{MB} + \frac{nk}{B} + 1)$ I/Os in a cache-oblivious model.*

Theorem 1 generalizes the previous bounds. In particular, with $k = O(1)$ we match the bounds of the cache-oblivious algorithm by Chowdhury et al. [4,6]. Furthermore, we obtain the same time-space trade-off for the backward pass as the algorithm by Driga et al. [9,10] by choosing $k$ accordingly.

We have implemented our algorithm and our optimized and simplified version of the algorithm by Driga et al. [9,10] with $k = 8, 16, 32$ and compared it with the previous algorithms on strings of length up to $2^{21} = 2097152$ on 3 different hardware architectures. In all our experiments, these algorithms significantly improve the current state-of-the-art cache-oblivious algorithm by Chowdhury et al. [4,6]. Our algorithms are faster even when $k = 8$ and the performance further improves until $k = 32$. Hence, our results show that a small constant factor additional space can have a significant impact in practice. Furthermore, we found that our new simple and cache-oblivious algorithm is competitive with our optimized, cache-aware, and tuned implementation of the more complicated algorithm by Driga et al. [9,10]. On one of the tested architectures our new algorithm was even substantially faster than the algorithm by Driga et al. [9,10].

Algorithmically, our new algorithm is a relatively simple combination of the division of the DPM into $k^2$ submatrices from Driga et al. [9,10] and the recursive and cache-oblivious approach from Chowdhury et al. [4,6]. A similar approach has been studied for solving the problem efficiently on multicore machines [5]. Our results show that this idea can also improve performance on individual cores.

### 1.4 Basic Definitions

For simplicity, we explain our algorithms in terms of the longest common subsequence problem. All of our bounds and techniques generalize immediately to any local dynamic programming string comparison problem.

Let $X$ be a string of length $|X| = n$ of characters from an alphabet $\Sigma$. We denote the character at position $i$ in $X$ by $X[i]$ and the substrings from position $i$ to $j$ by $X[i,j]$. The substrings $X[1,j]$ and $X[i,n]$ are the *prefixes* and *suffixes* of $X$, respectively. A *subsequence* of $X$ is any string $Z$ obtained by deleting characters in $X$. Given two strings $X$ and $Y$ a *common subsequence* is a subsequence of both $X$ and $Y$. A *longest common subsequence (LCS)* of $X$ and $Y$ is a common subsequence of $X$ and $Y$ of maximal length. The *longest common subsequence problem* is to compute an LCS of $X$ and $Y$.

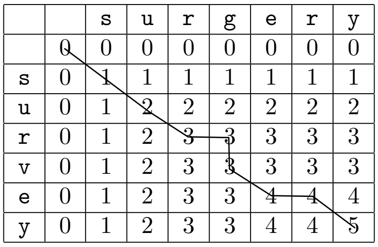
Let $X$ and $Y$ be strings of length $n$. The standard dynamic programming solution fills in an $n+1 \times n+1$ DPM $C$ according to the following recurrence.

$$C[i,j] = \begin{cases} 0 & \text{if } j = 0 \vee i = 0, \\ C[i-1,j-1]+1 & \text{if } i,j > 0 \wedge X[i] = Y[j], \\ \max \begin{cases} C[i,j-1] \\ C[i-1,j] \end{cases} & \text{if } i,j > 0 \wedge X[i] \neq Y[j] \end{cases} \tag{1}$$

The entry $C[i,j]$ is the length of the LCS between prefixes $X[1,i]$ and $Y[1,j]$ and hence the length of LCS of $X$ and $Y$ is $C[n,n]$. Note that each entry $C[i,j]$ depends only on the values in $C[i-1,j]$, $C[i,j-1]$, $C[i-1,j-1]$ and the characters of $X[i]$ and $Y[j]$. Hence, we can fill in the entries in a top-down left-to-right order. The *LCS path* is the path in $C$ obtained by backtracking from $C[n,n]$ to $C[0,0]$. Each diagonal edge in the LCS path corresponds to a character of the LCS. See Fig. 1 for an example.

|  |  | s | u | r | g | e | r | y |
|---|---|---|---|---|---|---|---|---|
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| s | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| u | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| r | 0 | 1 | 2 | 3 | 3 | 3 | 3 | 3 |
| v | 0 | 1 | 2 | 3 | 3 | 3 | 3 | 3 |
| e | 0 | 1 | 2 | 3 | 3 | 4 | 4 | 4 |
| y | 0 | 1 | 2 | 3 | 3 | 4 | 4 | 5 |

(a)



(b)

**Fig. 1.** Computing the LCS of `survey` and `surgery`. (a) The dynamic programming matrix. (b) The LCS path. Each diagonal edge corresponds to a character of the LCS. The resulting LCS is `surey`.

## 2 A New Algorithm for LCS

We now present our new algorithm for LCS. We describe our algorithm in the same basic framework as Chowdhury et al. [4,6].

Let $X$ and $Y$ be strings of length $n$ and let $k$ be the parameter for the algorithm. For simplicity, we assume that $n$ and $k$ are powers of 2. Our algorithm repeatedly uses a simple recursive procedure for computing the output boundary of the submatrix given the input boundary. We explain this procedure first and then give the full algorithm.

### 2.1 Computing Boundaries

Given the input boundary of a DPM for two strings of length $n$ we can compute the output boundary by computing the entries in the DPM in the standard top-down left-to-right order. This uses $O(n^2)$ time and $O(n)$ space, since we compute each entry in constant time and we only need to store the last two rows of the DPM during the algorithm. However, the number of I/Os incurred is $O(\frac{n^2}{B})$. The following simple recursive algorithm, similar to the one presented in Chowdhury et al. [4,6], improves this bound to $O(\frac{n^2}{MB})$ I/Os. If $n = 1$ we compute the output boundary directly using recurrence 1. Otherwise, we split the matrix into $k^2$ submatrices of size $\frac{n}{k} \times \frac{n}{k}$. We recursively compute the output boundaries for each submatrix by processing them in a top-down left-to-right order. At each recursive call to process a submatrix, the input boundary consists of the output boundary of the submatrix immediately above, to the left, and above-left. Hence with this ordering, the input boundary is available for each recursive call.

The algorithm uses $O(n^2)$ time and $O(n)$ space as before, i.e., we only need to store the boundaries of the submatrices that are currently being processed. Let $I_1(n, k)$ denote the number of I/Os incurred by the algorithm on strings of length $n$ with parameter $k$. If $n$ is sufficiently small such that the recursive

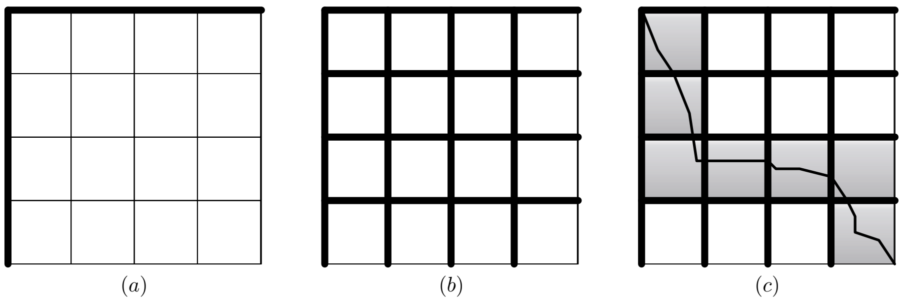


**Fig. 2.** States of the algorithm for $k = 4$. (a) A partition of the DPM into $4 \times 4$ submatrices. Thick lines indicate the initially stored boundaries. (b) After the forward pass. (c) After the backward pass. Only the shaded submatrices intersecting the LCS path are processed.

computation is done within internal memory, the algorithm only incurs I/Os to read and write the input strings and to read and write the boundaries. The length of the input strings and boundaries is $O(n + nk) = O(nk)$ and hence the number of I/Os is $O(\frac{nk}{B} + 1)$. Otherwise, the algorithm additionally creates $k^2$ subproblems of strings of length $\frac{n}{k}$ each. Thus, the total number of I/Os is given by the following recurrence.

$$I_1(n,k) = \begin{cases} O(\frac{nk}{B} + 1) & \text{if } n \leq \alpha_1 M, \\ k^2 I_1(\frac{n}{k}, k) + O(\frac{nk}{B} + 1) & \text{otherwise.} \end{cases} \tag{2}$$

Here, $\alpha_1$ is a suitable constant such that all computations of strings of length $\alpha_1 M$ are done entirely within memory. It follows that the total number of I/Os is $I_1(n,k) = O(\frac{n^2 k}{MB} + \frac{nk}{B} + 1)$.

## 2.2 Computing the LCS

We now present the full algorithm to compute the LCS in $C$. The algorithm is recursive and works as follows. If $n = 1$ we simply compute an optimal path directly using recurrence (1). Otherwise, we proceed in the following steps (see Fig 2).

*Step 1: Forward Pass* Partition $C$ into $k^2$ square submatrices of size $\frac{n}{k} \times \frac{n}{k}$. Compute and store the input boundaries of each submatrix in a top-down left-to-right order. We compute the boundaries using the algorithm from Sect. 2.1.

*Step 2: Backward pass* Compute an optimal LCS path through the submatrices from the bottom-right to the top-left. At each step we recursively find an optimal path through a submatrix $C'$ given the input boundary (computed in step 1) and a point on the optimal path on the output boundary. Depending on the exit point on the input boundary of the computed optimal LCS path through $C'$ we continue in the submatrix above, to the left, or above-left of $C'$ using the exit point as the point on the output boundary in the next step.

*Step 3: Output LCS* Finally, concatenate the path through the submatrices to form an optimal LCS path in $C$ and output the corresponding LCS.

## 2.3 Analysis

First consider the time complexity of the algorithm. Step 1 (the forward pass) uses $O(n^2)$ time. In step 2 (the backward pass), we only process the submatrices that are intersected by the optimal path. Since any path from $(n, n)$ to $(0, 0)$ can intersect at most $2k-1$ submatrices, step 2 uses $O((2k-1)\cdot\frac{n^2}{k^2}+n) = O(\frac{n^2}{k}+n)$ time. Finally, step 3 concatenates the pieces of the path and outputs the LCS in $O(n)$ time. In total, the algorithm uses $O(n^2)$ time.

Next consider the space used by the algorithm. Let $S(n, k)$ denote the space for a subproblem of size $n$ with parameter $k$. The stored input boundaries use $O(nk)$ space and the recursive call uses $S(n/k, k)$ space. Hence, the total space $S(n, k)$ is given by the recurrence

$$S(n,k) = \begin{cases} O(1) & \text{if } n = O(1), \\ S(n/k,k) + O(nk) & \text{otherwise.} \end{cases}$$

It follows that the space used by the algorithm is $S(n, k) = O(nk)$.

Next consider the I/O complexity. Let $I_2(n, k)$ denote I/O complexity of the algorithm on strings of length $n$ with parameter $k$. If $n$ is sufficiently small such that the recursive computation is done within internal memory, the algorithm incurs $O(\frac{nk}{B}+1)$ I/Os by similar arguments as in the analysis above. Otherwise, the algorithm additionally does $k^2-1$ boundary computations in step 1 on subproblems of size $\frac{n}{k}$ and recursively creates $2k-1$ subproblems of strings of length $\frac{n}{k}$. Hence, the total number of I/Os is given by

$$I_2(n,k) = \begin{cases} O(\frac{nk}{B}+1) & \text{if } n \le \alpha_2 M, \\ (k^2-1)I_1\left(\frac{n}{k},k\right) + (2k-1)I_2\left(\frac{n}{k},k\right) + O\left(\frac{nk}{B}+1\right) & \text{otherwise.} \end{cases} \tag{3}$$

Here, $\alpha_2$ is a suitable constant such that computation is done entirely in memory. It follows that $I_2(n, k) = O(\frac{n^2k}{MB}+\frac{nk}{B}+1)$.

In summary, our algorithm for LCS uses $O(n^2)$ time for the forward pass, $O(\frac{n^2}{k}+n)$ time for the backward pass, $O(nk)$ space, and $O(\frac{n^2k}{MB}+\frac{nk}{B}+1)$ I/Os. Since the algorithm only uses the local dependencies of LCS these bounds hold for any local dynamic programming string comparison problem. Hence, this completes the proof of Theorem 1.

## 3 Experimental Results

### 3.1 Setup

We have compared the following algorithms.

FullMatrix Wagner and Fischer's [21] original algorithm in our own implementation.

Hirschberg Hirschberg's [14] linear space divide and conquer algorithm in our own implementation.

CO The cache-oblivious algorithm by Chowdhury et al. [4,6]. We have tested the original implementation of the algorithm.

FastLSA$_k$ The FastLSA algorithm by Driga et al. [9,10] with parameter $k$. We used an optimized version of the original implementation of the algorithm. The optimization improves and simplifies parameter passing in the recursion, and the allocation and deallocation of the auxiliary arrays. In our experiments, we report the results for $k = 8, 16, 32$, since larger values of $k$ did not further improve the performance of the algorithm. Furthermore, we have tuned the threshold parameter $D$ for each of the tested hardware architectures.

FastCO$_k$ An implementation of our new algorithm with parameter $k$. As with FastLSA, we report the results for $k = 8, 16, 32$. In our experiments we found that the choice of $k$ did not affect the forward pass. For simplicity, we therefore fixed $k = 2$ for the forward and only varied $k$ in the backward pass.

We compared the algorithms on the following 3 architectures.

Intel i7 2.66GHz. 32KB L1, 256KB L2, 8MB L3 cache. 4GB memory
AMD X2 - 2.5GHz. 64KB L1, 512KB L2 cache. 4GB memory
Intel M - 1.6GHz. 2MB L2 cache. 1GB memory

All algorithms were implemented in C/C++ and compiled using the `gcc` 3.4 compiler. We tested the performance of the algorithms on strings of lengths $n = 2^i$, for $i = 16, 17, 18, 19, 20, 21$, i.e., the largest strings are slighty larger than 2 million. The strings are DNA strings taken from the standardized text collection of Pizza&Chili Corpus [1]. We have experimented with other types of strings but found only very small differences in performance. This is likely due to the small difference between worst-case and best-case performance. For brevity, we therefore focus DNA strings from the standardized text collection in our experiments. Additionally, we have used Cachegrind[2] to simulate the algorithms on a standard memory hierarchy with 64KB L1 and 512KB L2 caches.

### 3.2 Results

The results of the running time and the Cachegrind experiments are listed in Tables 2 and 3. Results for FullMatrix are not reported since they were either

[1] `pizzachili.dcc.uchile.cl` or `pizzachili.di.unipi.it`
[2] `valgrind.org/info/tools.html#cachegrind`

**Table 2.** Performance results for HIRSCHBERG (HB), CO, FASTLSA$_k$ (FLSA$_k$), FASTCO$_k$ (FCO$_k$) on Intel i7, AMD X2, and Intel M. The fastest running times for each row is in boldface.

Intel i7

| $n$ | HB | CO | FLSA$_8$ | FLSA$_{16}$ | FLSA$_{32}$ | FCO$_8$ | FCO$_{16}$ | FCO$_{32}$ |
|---|---|---|---|---|---|---|---|---|
| $2^{16}$ | $0.016h$ | $0.012h$ | $0.009h$ | $0.009h$ | **0.008h** | $0.009h$ | $0.009h$ | **0.008*h*** |
| $2^{17}$ | $0.063h$ | $0.049h$ | $0.0387h$ | $0.036h$ | **0.033h** | $0.036h$ | $0.035h$ | $0.034h$ |
| $2^{18}$ | $0.251h$ | $0.194h$ | $0.150h$ | $0.144h$ | **0.132h** | $0.143h$ | $0.136h$ | $0.134h$ |
| $2^{19}$ | $1.003h$ | $0.775h$ | $0.584h$ | $0.559h$ | **0.529h** | $0.565h$ | $0.539h$ | $0.530h$ |
| $2^{20}$ | $4.059h$ | $3.129h$ | $2.320h$ | $2.290h$ | $2.127h$ | $2.238h$ | $2.258h$ | **2.100h** |
| $2^{21}$ | $16.105h$ | $12.297h$ | $9.544h$ | $9.022h$ | $8.741h$ | $9.036h$ | $8.611h$ | **8.355h** |

AMD X2

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $2^{16}$ | $0.017h$ | $0.009h$ | $0.010h$ | $0.010h$ | $0.010h$ | **0.007h** | **0.007h** | **0.007h** |
| $2^{17}$ | $0.069h$ | $0.037h$ | $0.041h$ | $0.039h$ | $0.038h$ | $0.028h$ | $0.027h$ | **0.026h** |
| $2^{18}$ | $0.278h$ | $0.149h$ | $0.169h$ | $0.159h$ | $0.156h$ | $0.114h$ | $0.108h$ | **0.104h** |
| $2^{19}$ | $1.123h$ | $0.597h$ | $0.685h$ | $0.640h$ | $0.624h$ | $0.455h$ | $0.430h$ | **0.418h** |
| $2^{20}$ | $4.474h$ | $2.389h$ | $2.752h$ | $2.574h$ | $2.498h$ | $1.846h$ | $1.721h$ | **1.671h** |
| $2^{21}$ | $17.949h$ | $9.442h$ | $11.007h$ | $10.337h$ | $9.950h$ | $7.278h$ | $6.873h$ | **6.685h** |

Intel M

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $2^{16}$ | $0.027h$ | $0.021h$ | $0.0140h$ | $0.013h$ | - | $0.015h$ | **0.012h** | - |
| $2^{17}$ | $0.108h$ | $0.083h$ | $0.0571h$ | $0.053h$ | - | $0.061h$ | **0.050h** | - |
| $2^{18}$ | $0.438h$ | $0.334h$ | $0.227h$ | $0.218h$ | - | $0.234h$ | **0.200h** | - |
| $2^{19}$ | $1.800h$ | $1.337h$ | $0.945h$ | $0.889h$ | - | $0.928h$ | **0.852h** | - |
| $2^{20}$ | $7.170h$ | $5.325h$ | $3.814h$ | $3.575h$ | - | $3.697h$ | **3.481h** | - |
| $2^{21}$ | $28.999h$ | $20.601h$ | $15.283h$ | $14.994h$ | - | $14.730h$ | **14.529h** | - |

infeasible or far from competitive in all our experiments. Furthermore, for Intel M we only report result for FASTLSA and FASTCO with parameter $k$ up to 16 due to the small memory of this machine.

From Table 2 we see that FASTLSA$_{32}$ or FASTCO$_{32}$ significantly outperform the current state-of-the-art cache-oblivious algorithm CO. Compared with HIRSCHBERG, FASTLSA$_{32}$ and FASTCO$_{32}$ are a about a factor 1.8 to 2.6 faster. Surprisingly, our simple cache-oblivious FASTCO$_k$ is competitive with our optimized and cache-aware implementation of the more complicated FASTLSA$_k$. For the AMD X2 architecture, FASTCO is even significantly faster than FASTLSA. We outperform previous results even when $k = 8$ and our performance further improves until $k = 32$.

Our Cachegrind experiments listed in Table 3 show that FASTCO$_k$ executes a similar number of instructions as FASTLSA$_k$ and far less than both HIRSCHBERG and CO. Furthermore, FASTCO$_k$ incurs at least a factor 2000 less cache misses in both L1 and L2 cache compared to FASTLSA$_k$. Note that this corresponds well with the difference in the theoretical I/O complexity between the algorithms. The fewest number of cache misses are incurred by CO closely followed by FASTCO$_k$.

**Table 3.** Cachegrind results for HIRSCHBERG (HB), CO, FASTLSA$_k$ (FLSA$_k$), FASTCO$_k$ (FCO$_k$) on a 64K L1 and 512K L2 cache hierarchy. Lowest instruction/miss count shown in boldface. We could only test inputs up to $n = 2^{19}$ due to the overhead of the Cachegrind simulation.

Instructions Executed $\times 10^9$

| $n$ | HB | CO | FCO$_8$ | FCO$_{16}$ | FCO$_{32}$ | FLSA$_8$ | FLSA$_{16}$ | FLSA$_{32}$ |
|---|---|---|---|---|---|---|---|---|
| $2^{16}$ | 177.30 | 123.84 | 77.122 | 72.823 | **70.670** | 84.463 | 79.043 | 76.584 |
| $2^{17}$ | 708.87 | 494.68 | 308.06 | 290.91 | **282.53** | 337.68 | 315.95 | 306.07 |
| $2^{18}$ | 2,835.4 | 1,978.2 | 1,221.2 | 1,163.4 | **1,129.3** | 1,352.9 | 1,263.7 | 1,223.8 |
| $2^{19}$ | 11,339 | 7,907.8 | 4,914.2 | 4,646.4 | **4,514.6** | 5,416.1 | 5,057 | 4,894.5 |

L1 cache misses $\times 10^6$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $2^{16}$ | 1,090 | **0.855** | 1.682 | 1.980 | 2.070 | 1,293 | 1,162 | 1,154 |
| $2^{17}$ | 4,639 | **1.952** | 3.823 | 4.566 | 7.49 | 5,156 | 4,834 | 4,626 |
| $2^{18}$ | 18,866 | **5.916** | 11.23 | 12.29 | 17.41 | 20,640 | 19,4654 | 18,789 |
| $2^{19}$ | 76,654 | **19.85** | 37.27 | 39.59 | 46.55 | 83,201 | 77,630 | 75,355 |

L2 cache misses $\times 10^6$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $2^{16}$ | 604.6 | **0.345** | 1.038 | 1.373 | 1.711 | 1,151 | 1,146 | 1,150 |
| $2^{17}$ | 3,207 | **0.594** | 1.949 | 2.49 | 5.2 | 4,575 | 4,579 | 4,581 |
| $2^{18}$ | 16,517 | **1.312** | 4.385 | 5.067 | 9.373 | 18,741 | 18,303 | 18,290 |
| $2^{19}$ | 71,629 | **3.416** | 11.689 | 15.12 | 18.792 | 82,131 | 75,219 | 73,117 |

The difference between the number of cache misses incurred by FASTCO$_k$ and FASTLSA$_k$ is much larger than the difference in their running time. The main reason for this is because the number of cache misses incurred relative to the total number of instructions executed is low (around 4% for FASTLSA$_k$). Ultimately, the simple FASTCO$_k$ simultaneously achieves good cache performance and a low instruction count making it competitive with current state-of-the-art algorithms.

**Acknowledgments.** We would like to thank the authors of Driga et al. [9, 10] and Chowdhury et al. [4, 6] for providing us with the source code of their algorithms.

## References

1. Aggarwal, A., Vitter, J.S.: The Input/Output complexity of sorting and related problems. Commun. ACM 31(9), 1116–1127 (1988)
2. Bille, P.: Faster approximate string matching for short patterns. Theory Comput. Syst. (2011) (to appear)
3. Bille, P., Farach-Colton, M.: Fast and compact regular expression matching. Theoret. Comput. Sci. 409(3), 486–496 (2008)
4. Chowdhury, R.A., Ramachandran, V.: Cache-oblivious dynamic programming. In: Proc. 17th Symp. on Discrete Algorithms, pp. 591–600 (2006)

5. Chowdhury, R.A., Ramachandran, V.: Cache-efficient dynamic programming algorithms for multicores. In: Proc. 20th Symp. on Parallelism in Algorithms and Architectures, pp. 207–216 (2008), http://doi.acm.org/10.1145/1378533.1378574
6. Chowdhury, R.A., Le, H.S., Ramachandran, V.: Cache-oblivious dynamic programming for bioinformatics. Trans. Comput. Biol. and Bioinformatics 7, 495–510 (2010)
7. Cole, R., Hariharan, R.: Approximate string matching: A simpler faster algorithm. SIAM J. Comput. 31(6), 1761–1782 (2002)
8. Crochemore, M., Landau, G.M., Ziv-Ukelson, M.: A subquadratic sequence alignment algorithm for unrestricted scoring matrices. SIAM J. Comput. 32(6), 1654–1673 (2003)
9. Driga, A., Lu, P., Schaeffer, J., Szafron, D., Charter, K., Parsons, I.: FastLSA: A fast, linear-space, parallel and sequential algorithm for sequence alignment. In: Proc. Intl. Conf. on Parallel Processing, pp. 48–57 (2005)
10. Driga, A., Lu, P., Schaeffer, J., Szafron, D., Charter, K., Parsons, I.: FastLSA: A fast, linear-space, parallel and sequential algorithm for sequence alignment. Algorithmica 45, 337–375 (2006)
11. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. 40th Symp. Foundations of Computer Science, pp. 285–297 (1999)
12. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology, Cambridge (1997)
13. Hermelin, D., Landau, G.M., Landau, S., Weimann, O.: A unified algorithm for accelerating edit-distance computation via text-compression. In: Proc. 26th Symp. Theoretical Aspects of Computer Science. Leibniz International Proceedings in Informatics (LIPIcs), vol. 3, pp. 529–540 (2009)
14. Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. Commun. ACM 18(6), 341–343 (1975)
15. Hunt, J.W., Szymanski, T.G.: A fast algorithm for computing longest common subsequences. Commun. ACM 20, 350–353 (1977)
16. Landau, G.M., Vishkin, U.: Fast parallel and serial approximate string matching. J. Algorithms 10, 157–169 (1989)
17. Masek, W., Paterson, M.: A faster algorithm for computing string edit distances. J. Comput. System Sci. 20, 18–31 (1980)
18. Myers, E.W., Miller, W.: Optimal alignments in linear space. Comput. Appl. Biosci. 4(1), 11–17 (1988)
19. Myers, G.: A fast bit-vector algorithm for approximate string matching based on dynamic programming. J. ACM 46(3), 395–415 (1999)
20. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. 33(1), 31–88 (2001)
21. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM 21, 168–173 (1974)

# An Efficient Implicit OBDD-Based Algorithm for Maximal Matchings*

Beate Bollig[1] and Tobias Pröger[2]

[1] TU Dortmund, LS2 Informatik, Germany
[2] ETH Zürich, Institut für Theoretische Informatik, Switzerland

**Abstract.** The maximal matching problem, i.e., the computation of a matching that is not a proper subset of another matching, is a fundamental optimization problem and algorithms for maximal matchings have been used as submodules for problems like maximal node-disjoint paths or maximum flow. Since in some applications graphs become larger and larger, a research branch has emerged which is concerned with the design and analysis of implicit algorithms for classical graph problems. Input graphs are given as characteristic Boolean functions of their edge sets and problems have to be solved by functional operations. As OBDDs, which are closely related to deterministic finite automata, are a well-known data structure for Boolean functions, OBDD-based algorithms are used as a heuristic approach to handle very large graphs. Here, an implicit OBDD-based maximal matching algorithm is presented that uses only a polylogarithmic number of functional operations with respect to the number of vertices of the input graph.

## 1 Introduction

Since some modern applications require huge graphs, explicit representations by adjacency matrices or adjacency lists may cause conflicts with memory limitations and even polynomial time algorithms are sometimes not fast enough. As time and space resources do not suffice to consider individual vertices and edges, one way out seems to be to deal with sets of vertices and edges represented by their characteristic functions. Ordered binary decision diagrams, denoted OBDDs, are well suited for the representation and manipulation of Boolean functions [5]. They are closely related to deterministic finite automata for Boolean languages $L$, where $L \subseteq \{0,1\}^n$ (see, e.g., Section 3.2 in [15]). OBDDs are able to take advantage over the presence of regular substructures which leads sometimes to sublinear graph representations. Therefore, a research branch has emerged which is concerned with the design and analysis of so-called implicit or symbolic algorithms for classical graph problems on OBDD-represented graph instances (see, e.g., [1–3], [6, 7], [9], [12, 13], and [16]). Implicit algorithms have to solve problems on a given graph instance by efficient functional operations offered by the OBDD data structure.

* The first author has been supported by DFG project BO 2755/1-1.

The maximal matching problem, i.e., the computation of a matching that is not a proper subset of another matching, is a fundamental optimization problem and algorithms for maximal matchings have been used in some maximal node-disjoint paths or maximum flow algorithms (see, e.g., [8]). The design of efficient implicit algorithms requires new paradigms and techniques but it has turned out that some methods known from the design of parallel algorithms are useful, e.g., the technique of iterative squaring is similar to the path-doubling strategy. Using an efficient degree reduction procedure, the first optimal parallel algorithm for maximal matchings has been presented by Kelsen [11]. It runs in time $\mathcal{O}(\log^3 |V|)$ using $\mathcal{O}((|E| + |V|)/\log^3 |V|)$ processors on bipartite graphs $G = (V, E)$ and is optimal in the sense that the time processor product is equal to that of the best sequential algorithm. The main result of our paper is the following one.

**Theorem 1.** *A maximal bipartite matching in an implicitly defined graph $G = (V, E)$ can be implicitly computed by $\mathcal{O}(\log^4 |V|)$ functional operations on Boolean functions over a logarithmic number of Boolean variables. For general graphs $\mathcal{O}(\log^5 |V|)$ functional operations are sufficient.*

For this result we make use of the algorithm presented in [11] but for the implicit setting also new ideas are necessary. Note, that our aim is not to achieve new algorithmic techniques for explicit graph representations but to demonstrate the similarity of paradigms in the design of parallel and implicit algorithms that can also be used as building blocks for the solution of other combinatorial problems on one hand and on the other hand to develop efficient algorithms for large structured graphs. The similarity between implicit and parallel algorithms has also been demonstrated by the following result. A problem can be solved in the implicit setting with a polylogarithmic number of functional operations on a logarithmic number of Boolean variables (with respect to the number of vertices of the input graph) iff the problem is in NC, the complexity class that contains all problems computable in polylogarithmic time with polynomially many processors [13, 14]. Nevertheless, this structural result does not lead directly to efficient implicit algorithms.

In order to reduce the number of functional operations, iterative squaring is used in our algorithm. One may argue against the use of iterative squaring because despite the improvement in the number of functional operations intermediate results of exponential size (with respect to the input length) can be generated. Nevertheless, Sawitzki has demonstrated that iterative squaring can also be useful in applications [12]. The maximum flow problem in 0-1 networks has been one of the first classical graph problems for which an implicit OBDD-based algorithm has been presented and Hachtel and Somenzi were able to compute a maximum flow for a graph with more than $10^{27}$ vertices and $10^{36}$ edges in less than one CPU minute [9]. To improve this algorithm Sawitzki has used iterative squaring for the computation of augmenting paths by $\mathcal{O}(\log^2 |V|)$ functional operations. If the maximum flow value is constant with respect to the network size, the algorithm performs altogether a polylogarithmic number of operations. Both max flow algorithms belong to the class of so-called layered-network methods but Sawitzki's algorithm prevents breadth-first searches by using iterative squaring and as a result overcomes the dependence on the depths of the layered networks. In order to confirm the practical relevance of his algorithm he has implemented both maximum flow algorithms and has shown that his algorithm outperforms the algorithm of Hachtel and Somenzi for very structured graphs.

The rest of the paper is organized as follows. In Section 2 we define some notation and review some basics concerning OBDDs and functional operations, implicit graph algorithms and matchings. Section 3 contains the main result, an implicit algorithm for the maximal matching problem that uses only a polylogarithmic number of functional operations with respect to the number of vertices of the input graph. Finally, we finish the paper with some concluding remarks.

In order to investigate the algorithm's behavior on large and structured networks, it has been analyzed on grid graphs and it has been shown that the overall running time and the space requirement is also polylogarithmic (for these results see the full version of the paper [4]).

## 2 Preliminaries

We briefly recall the main notions we are dealing with in the paper.

### 2.1 OBDDs and Functional Operations

OBDDs are a very popular dynamic data structure in areas working with Boolean functions, like circuit verification or model checking. (For a history of results on binary decision diagrams see, e.g., [15]).

**Definition 2.** *Let $X_n = \{x_1, \ldots, x_n\}$ be a set of Boolean variables. A variable ordering $\pi$ on $X_n$ is a permutation on $\{1, \ldots, n\}$ leading to the ordered list $x_{\pi(1)}, \ldots, x_{\pi(n)}$ of the variables. A $\pi$-*OBDD* on $X_n$ is a directed acyclic graph $G = (V, E)$ whose sinks are labeled by the Boolean constants $0$ and $1$ and whose non-sink (or decision) nodes are labeled by Boolean variables from $X_n$. Each decision node has two outgoing edges, one labeled by $0$ and the other by $1$. The edges between decision nodes have to respect the variable ordering $\pi$, i.e., if an edge leads from an $x_i$-node to an $x_j$-node, then $\pi^{-1}(i) < \pi^{-1}(j)$ ($x_i$ precedes $x_j$ in $x_{\pi(1)}, \ldots, x_{\pi(n)}$). Each node $v$ represents a Boolean function $f_v \in B_n$, i.e., $f_v : \{0,1\}^n \rightarrow \{0,1\}$, defined in the following way. In order to evaluate $f_v(b)$, $b \in \{0,1\}^n$, start at $v$. After reaching an $x_i$-node choose the outgoing edge with label $b_i$ until a sink is reached. The label of this sink defines $f_v(b)$. The size of a $\pi$-*OBDD* $G$, denoted by $|G|$, is equal to the number of its nodes. A $\pi$-*OBDD* of minimal size for a given function $f$ and a fixed variable ordering $\pi$ is unique up to isomorphism. The $\pi$-*OBDD* size of a function $f$, denoted by $\pi$-*OBDD*$(f)$, is the size of the minimal $\pi$-*OBDD* representing $f$. The *OBDD* size of $f$ is the minimum of all $\pi$-*OBDD*$(f)$.*

Sometimes it is useful to have the notion of OBDDs where there are only edges between nodes labeled by neighboring variables, i.e., if an edge leads from an $x_i$-node to an $x_j$-node, then $\pi^{-1}(i) = \pi^{-1}(j) - 1$.

**Definition 3.** *An* OBDD *on $X_n$ is complete if all paths from the source to one of the sinks have length $n$. The width of a complete* OBDD *is the maximal number of nodes labeled by the same variable.*

A variable ordering is called a natural variable ordering if $\pi$ is the identity $1, 2, \ldots, n$. Complete OBDDs with respect to natural variable orderings differ from deterministic finite automata only in the minor aspect that tests may not be omitted even if the corresponding subfunction is the constant 0.

Now, we briefly describe a list of important operations on OBDDs (for a detailed discussion and the corresponding time and space requirements see, e.g., Section 3.3 in [15] and the full version of the paper [4]). Let $f$ and $g$ be Boolean functions in $B_n$ on the variable set $X_n = \{x_1, \ldots, x_n\}$ and $G_f$ and $G_g$ be $\pi$-OBDDs for the representations of $f$ and $g$, respectively.

– Negation: Given $G_f$, compute a $\pi$-OBDD for the function $\overline{f} \in B_n$.
– Replacement by constant: Given $G_f$, an index $i \in \{1, \ldots, n\}$, and a Boolean constant $c_i \in \{0, 1\}$, compute a $\pi$-OBDD for the subfunction $f_{|x_i=c_i}$.
– Equality test: Given $G_f$ and $G_g$, decide, whether $f$ and $g$ are equal.
– Satisfiability: Given $G_f$, decide, whether $f$ is not the constant function 0.
– Synthesis: Given $G_f$ and $G_g$ and a binary Boolean operation $\otimes \in B_2$, compute a $\pi$-OBDD $G_h$ for the function $h \in B_n$ defined as $h := f \otimes g$.
– Quantification: Given $G_f$, an index $i \in \{1, \ldots, n\}$, and a quantifier $Q \in \{\exists, \forall\}$, compute a $\pi$-OBDD $G_h$ for the function $h \in B_n$ defined as $h := (Qx_i)f$, where $(\exists x_i)f := f_{|x_i=0} \vee f_{|x_i=1}$ and $(\forall x_i)f := f_{|x_i=0} \wedge f_{|x_i=1}$. In the rest of the paper quantifications over $k$ Boolean variables $(Qx_1, \ldots, x_k)f$ are denoted by $(Qx)f$, where $x = (x_1, \ldots, x_k)$.

Sometimes it is useful to reverse the edges of a given graph. Therefore, we define the following operation (see, e.g., [13]).

**Definition 4.** *Let $\rho$ be a permutation on $\{1, \ldots, k\}$ and $f \in B_{kn}$ be defined on Boolean variable vectors $x^{(1)}, \ldots, x^{(k)}$ of length $n$. The argument reordering $\mathcal{R}_\rho(f) \in B_{kn}$ with respect to $\rho$ is $\mathcal{R}_\rho(f)(x^{(1)}, \ldots, x^{(k)}) = f(x^{(\rho(1))}, \ldots, x^{(\rho(k))})$.*

## 2.2 OBDD-Based Graph Algorithms and Matching Problems

Let $G = (V, E)$ be a graph with $N$ vertices $v_0, \ldots v_{N-1}$ and $|z|_2 := \sum_{i=0}^{n-1} z_i 2^i$, where $z = (z_{n-1}, \ldots, z_0) \in \{0, 1\}^n$ and $n = \lceil \log N \rceil$. Now, $E$ can be represented by an OBDD for its characteristic function, where $x, y \in \{0, 1\}^n$ and

$$\chi_E(x, y) = 1 \Leftrightarrow (|x|_2, |y|_2 < N) \wedge (v_{|x|_2}, v_{|y|_2}) \in E.$$

(For the ease of notation we omit the index 2 in the rest of the paper and we assume that $N$ is a power of 2 since it has no bearing on the essence of our results.) Undirected edges are represented by symmetric directed ones. Furthermore, we do not distinguish between vertices of the input graph and their Boolean encoding since the meaning is clear from the context.

For implicit computations some Boolean functions are helpful. The equality function $EQ$ computes 1 for two inputs $x$ and $y$ iff $|x| = |y|$. $NEQ$ is the negated equality function. Since sometimes a vertex (or an edge) has to be chosen out of a given set of vertices (or edges), several priority functions $\Pi_\prec$ have been defined in the implicit setting (see, e.g., [9, 12]). We define $\Pi_\prec(x, y, z) = 1$ iff $y \prec_x z$, where $\prec_x$ is a total order on the vertex set $V$ and $x, y, z$ are vertices in $V$. In the following we only use a very simple priority function independent of the choice of $x$, where $\Pi_\prec(x, y, z) = 1$ iff $|y| < |z|$. It is easy to see that $EQ$, $NEQ$, and $\Pi_\prec$ can be represented by OBDDs of linear size with respect to variable orderings, where the variables with the same significance are tested one after another.

A graph $G = (V, E)$ is bipartite, if $V$ can be partitioned into two disjoint nonempty sets $U$ and $W$, such that for all edges $(u, w) \in E$ it holds $u \in U$ and $w \in W$ or vice versa. The distance between two edges on a (directed) path is the number of edges between them. The distance between two vertices on a (directed) path is the number of vertices between them plus 1. The degree of a vertex $v$ in $G$ is the number of edges in $E$ incident to $v$. A matching in an undirected graph $G = (V, E)$ is a subset $M \subseteq E$ such that no two edges of $M$ are adjacent. $M$ is a maximum matching if there exists no matching $M' \subseteq E$ such that $|M'| > |M|$, where $|S|$ denotes the cardinality of a set $S$. A matching $M$ is maximal if $M$ is not a proper subset of another matching. Given a matching $M$ a vertex $v$ is matched if $(v, w) \in M$ for some $w \in V$ and free otherwise.

In the implicit setting the maximum (maximal) matching problem is the following one. Given an OBDD for the characteristic function of the edge set of an undirected input graph $G$, the output is an OBDD that represents the characteristic function of a maximum (maximal) matching in $G$.

## 3 The Maximal Matching Algorithm

In this section we prove Theorem 1 and present an implicit algorithm for the maximal bipartite matching problem. The algorithm can easily be extended for general graphs. The idea is to start with Kelsen's parallel algorithm for the computation of maximal matchings on explicitly defined graphs [11]. In the parallel setting more or less only a high-level description of the algorithm is given. Moreover, we have to add more ideas because we cannot access efficiently single vertices or edges in the implicit setting.

The algorithm `findMaximalBipartiteMatching` is simple. Step-by-step a current matching is enlarged by computing a matching in the subgraph of $G = (V, E)$ that consists only of the edges that are not incident to the current matching. The key idea is an algorithm `match` that computes a matching $M'$, $M' \subseteq E$, adjacent to at least a fraction of 1/6 of the edges in the input graph for `match`. After removing these edges from the input graph the procedure is repeated. Therefore, after $\mathcal{O}(\log |V|)$ iterations the remaining subgraph is empty and the current matching is obviously a maximal matching in $G$.

The algorithm `match` makes use of another algorithm `halve` that halves approximately the degree of each vertex in a bipartite graph. The idea is to compute

**Algorithm 1.** `findMaximalBipartiteMatching`

**Input:** $\chi_E(x, y)$

```
(1) ▷ Initialize. Start with the empty matching.
    $M(x, y) \leftarrow 0$
(2) while $\chi_E(x, y) \neq 0$ do
(3)     ▷ Compute a matching $M'$.
        $M'(x, y) \leftarrow \mathtt{match}(\chi_E(x, y))$
(4)     ▷ Delete the edges incident to a matched vertex in $M'$.
        $\mathtt{INCNODE}(x) \leftarrow (\exists y)(M'(x, y))$
        $\chi_E(x, y) \leftarrow \chi_E(x, y) \wedge \overline{\mathtt{INCNODE}(x)} \wedge \overline{\mathtt{INCNODE}(y)}$
(5)     ▷ Add the edges from $M'$ to $M$.
        $M(x, y) \leftarrow M(x, y) \vee M'(x, y)$
(6) return $M(x, y)$
```

an Euler partition of the input graph such that the graph is decomposed into edge-disjoint paths. Each vertex of odd (even) degree is the endpoint of exactly 1 (0) open path. By two-coloring the edges on each path in the Euler partition and deleting all edges of one color, the degree of each vertex in the input graph is approximately halved. Here, we use the fact that bipartite graphs have no cycles of odd length. Therefore, for each path, where a vertex $v$ is not an endpoint, and for each cycle, the number of edges incident to $v$ colored by one of the two colors is equal to the number of edges colored by the other one. In fact in the algorithm `halve` we only use the color red and delete all red edges after the coloring. A precondition of the parallel algorithm `halve` is that for each vertex its incident edges have been paired [11]. Here, we present a new algorithm called `calculatePairing` which computes implicitly a pairing of the edges with $\mathcal{O}(\log^2 |V|)$ functional operations. This algorithm together with the degree reduction procedure in `halve` can possibly be used as building blocks for the solution of other combinatorial problems in the implicit setting. We assume that there is for each vertex an ordering on its incident edges given by a priority function (see Section 2). In the first step of the algorithm `calculatePairing` for each vertex the neighborhood of its incident edges is determined. Almost all edges have two neighbors with respect to one of its endpoints (all but the first and the last one). In order to compute a symmetric pairing every other edge incident to the same vertex is colored red. This is realized by an indicator function called `RED`. Afterwards, two neighboring edges $(x, y), (x, z)$ are paired iff $(x, y)$ is red, i.e., $\mathtt{RED}(x, y) = 1$, and $(x, y)$ has a higher priority than $(x, z)$, i.e., $\Pi_{\prec}(x, y, z) = 1$. Therefore, each edge has at most one chosen neighbor with respect to one of its endpoints and at most one of the edges incident to the same vertex is not paired. The output of `calculatePairing` is a function which depends on three vertex arguments $x, y$, and $z$ and whose function value is 1 iff $y$ and $z$ are two vertices adjacent to $x$ which have been paired. Therefore, the function is symmetric in the second and third argument. For the computation of the pairing we determine for each edge its position with respect to all edges incident to the same vertex according to a priority function. This procedure is similar to the well-known list ranking algorithm: given a linked list for each member of the list the number of its position in the list has to be calculated (for a nice introduction into design and analysis of parallel algorithms see, e.g., [10]). In our case we are only interested in whether the number of the position of an edge according to a priority function is odd or even.

**Algorithm 2.** `calculatePairing`

```
Input: $\chi_E(x,y)$

(1) ▷ Determine the neighborhood of the edges.
    $\mathtt{ORDER}(x,y,z) \leftarrow \chi_E(x,y) \wedge \chi_E(x,z) \wedge \Pi_{\prec}(x,y,z) \wedge \overline{(\exists \xi)(\chi_E(x,\xi) \wedge \Pi_{\prec}(x,y,\xi) \wedge \Pi_{\prec}(x,\xi,z))}$
(2) ▷ Compute the distance between edges incident to the same vertex
    using iterative squaring.
    $\mathtt{DIST}_0(x,y,z) \leftarrow \mathtt{ORDER}(x,y,z)$
    for $i = 1,2,...,\log|V|$ do
        $\mathtt{DIST}_i(x,y,z) \leftarrow (\exists \xi)(\mathtt{DIST}_{i-1}(x,y,\xi) \wedge \mathtt{DIST}_{i-1}(x,\xi,z))$
(3) ▷ Color for each vertex its incident edges alternately.
    $\mathtt{RED}(x,y) \leftarrow \chi_E(x,y) \wedge \overline{(\exists \xi)(\chi_E(x,\xi) \wedge \Pi_{\prec}(x,\xi,y))}$
    for $i = 1,2,...,\log|V|$ do
        $\mathtt{RED}(x,y) \leftarrow \mathtt{RED}(x,y) \vee (\exists \xi)(\mathtt{RED}(x,\xi) \wedge \mathtt{DIST}_i(x,\xi,y))$
(4) ▷ Select only edge pairs $((x,y),(x,z))$, where the first edge is red.
    return $(\mathtt{ORDER}(x,y,z) \wedge \mathtt{RED}(x,y)) \vee (\mathtt{ORDER}(x,z,y) \wedge \mathtt{RED}(x,z))$
```

**Lemma 5.** *The algorithm* `calculatePairing` *computes for all vertices a pairing of its incident edges respectively with $\mathcal{O}(\log^2|V|)$ functional operations.*

*Proof.* The correctness of `calculatePairing` follows from the following observations: the function $\mathtt{ORDER}(x,y,z)$ computes the output 1 iff $y$ and $z$ are adjacent to the vertex $x$, the edge $(x,y)$ is smaller than the edge $(x,z)$ according to the chosen priority function, and there is no edge between $(x,y)$ and $(x,z)$ (with respect to the priority function). In step 2 the function $\mathtt{DIST}_i(x,y,z)$ computes 1 iff the distance, i.e., the number of edges between the edge $(x,y)$ and $(x,z)$ with respect to the priority function, is $2^i-1$. Afterwards for each vertex the first of its incident edges according to the priority function is colored red and then all edges which have an odd distance to the first one are also colored red. Now, the output of `calculatePairing` is a function on three vertex arguments $x,y$ and $z$, where the value is 1 iff the edges $(x,y)$ and $(x,z)$ are neighbored and the first one with respect to the priority function is red.

The most time consuming steps during `calculatePairing` are (2) and (3), where the position of the edges and the coloring of the incident edges are calculated. Traversing the incident edges of a vertex needs $\mathcal{O}(\log|V|)$ iterations each using $\mathcal{O}(\log|V|)$ operations for the quantification over $\mathcal{O}(\log|V|)$ variables. □

**Algorithm 3.** `halve`

**Input:** $\chi_E(x, y)$

```
(1) ▷ Compute the successor relation.
    $\mathtt{PAIRING}(x,y,z) \leftarrow \mathtt{calculatePairing}(\chi_E(x,y))$
    $\mathtt{SUCC}(x,y,z) \leftarrow \mathtt{PAIRING}(y,x,z)$
(2) ▷ Compute distance and reachability relations on the directed edges
    defined by the successor relation.
    $i \leftarrow 0$
    $\mathtt{DIST}_0(v,w,x,y) \leftarrow EQ(w,x) \wedge \mathtt{SUCC}(v,w,y)$
    $\mathtt{REACHABLE}(v,w,x,y) \leftarrow (EQ(v,x) \wedge EQ(w,y)) \vee (EQ(w,x) \wedge \mathtt{SUCC}(v,w,y))$
    **repeat**
        $i \leftarrow i+1$
        $\mathtt{REACHABLE}'(v,w,x,y) \leftarrow \mathtt{REACHABLE}(v,w,x,y)$
        $\mathtt{REACHABLE}(v,w,x,y) \leftarrow \mathtt{REACHABLE}(v,w,x,y) \vee$
            $(\exists \xi,\theta)(\mathtt{REACHABLE}(v,w,\xi,\theta) \wedge \mathtt{REACHABLE}(\xi,\theta,x,y))$
        $\mathtt{DIST}_i(v,w,x,y) \leftarrow (\exists \xi,\theta)(\mathtt{DIST}_{i-1}(v,w,\xi,\theta) \wedge \mathtt{DIST}_{i-1}(\xi,\theta,x,y))$
    **until** $\mathtt{REACHABLE}'(v,w,x,y) = \mathtt{REACHABLE}(v,w,x,y)$
(3) ▷ On each path, color an appropriate edge red.
    $\mathtt{RED}(x,y) \leftarrow \chi_E(x,y) \wedge (\forall \xi,\theta)(\overline{\mathtt{REACHABLE}(\xi,\theta,x,y)} \vee EQ(\xi,x) \vee \Pi_{\prec}(x,x,\xi)) \wedge$
          $(\forall \theta,\xi)(\overline{\mathtt{REACHABLE}(\theta,\xi,y,x)} \vee EQ(\theta,x) \vee \Pi_{\prec}(x,x,\theta)) \wedge$
          $(\forall \xi)((\overline{\mathtt{REACHABLE}(x,y,\xi,x)} \wedge \overline{\mathtt{REACHABLE}(\xi,x,x,y)}) \vee \Pi_{\prec}(x,y,\xi))$
    $\mathtt{RED}(x,y) \leftarrow \mathtt{RED}(x,y) \vee \mathtt{RED}(y,x)$
(4) ▷ Color the edges alternately.
    **for** $j = 1,2,...,i$ **do**
        $\mathtt{RED}(x,y) \leftarrow \mathtt{RED}(x,y) \vee (\exists \xi,\theta)(\mathtt{RED}(\xi,\theta) \wedge \mathtt{DIST}_j(\xi,\theta,x,y))$
(5) ▷ Delete the red edges.
    **return** $\chi_E(x,y) \wedge \overline{\mathtt{RED}(x,y)} \wedge \overline{\mathtt{RED}(y,x)}$
```

The pairing computed by `calculatePairing` is symmetric and it is used by the algorithm `halve` to define (directed) paths in the (undirected) input graph. An edge $(y, z)$ is a successor edge of an edge $(x, y)$ and $\mathtt{SUCC}(x, y, z) = 1$ iff the edges $(y, z)$ and $(y, x)$ are paired according to `calculatePairing`. Using this successor relation `SUCC` the undirected input graph is decomposed into directed edge-disjoint paths. Since the pairing is symmetric, $(y, x)$ is also a successor of $(z, y)$. Therefore, for each directed path from a vertex $u'$ to a vertex $u''$ defined by the successor relation `SUCC`, there exists also a directed path from $u''$ to $u'$. This property is important in order to guarantee that a coloring of the directed edges can be used for an appropriate coloring of the undirected edges in the input graph. For each directed path in the decomposition every other edge is colored red and a directed edge $(u, v)$ is red iff the directed edge $(v, u)$ is red. Therefore, for each pair of (undirected) edges computed by `calculatePairing` exactly one edge is red and by deleting the red edges the degree of each vertex is approximately halved. A crucial step, which is new in the implicit setting, is the choice of the first edges that are colored red on a directed path, because all directed paths are investigated simultaneously, i.e., for each directed path from a vertex $s$ to a vertex $t$ the directed path from $t$ to $s$ is considered at the same time. We have to avoid the situation that two edges $(x, y)$ and $(x, z)$, where $(x, y)$ is a directed edge on a directed path from a vertex $s$ to a vertex $t$ and $(x, z)$ a directed edge on the reversed path from $t$ to $s$, are colored red at the same time, because otherwise all edges from $s$ to $t$ and from $t$ to $s$ would be red after the coloring procedure. Therefore, we ensure that in the beginning either directed edges on the path from $s$ to $t$ or on the corresponding path from $t$ to $s$ are colored. For this reason the edge relation `Reachable` is used, where $\texttt{REACHABLE}(v, w, x, y)$ is 1 iff there exists a directed path from the edge $(v, w)$ to the edge $(x, y)$ defined by the successor relation `SUCC`. Due to the symmetry of `SUCC` the relation `REACHABLE` is also symmetric in the following way: iff $\texttt{REACHABLE}(v, w, x, y) = 1$ then $\texttt{REACHABLE}(y, x, w, v) = 1$. Therefore, using `REACHABLE` it is also possible to determine the predecessors of a directed edge. Now, the first red edges on a directed path are the edges with the highest priority: for each directed path the smallest vertex $v$ on the path according to a priority function together with a successor $u$ is chosen if there exists no predecessor of $v$ which has a higher priority than $u$ according to the chosen priority function. This procedure ensures that either for a directed path starting from a vertex $s$ and ending in a vertex $t$ an edge $(v, u)$ is chosen or for the reversed directed path from $t$ to $s$. Afterwards an edge $(u, v)$ is colored red iff the edge $(v, u)$ is red. Next, each edge on a directed path for which the distance to one of the first red edges on the path is odd is also colored red and all red edges are deleted from the input graph. Note, that it is possible for a directed path that more than one edge is chosen in the beginning but these edges have the same starting point, therefore the distance between these edges is even because the input graph is bipartite such that no problem occurs.

**Lemma 6.** *Let $d(v)$ and $d'(v)$ denote the degree of a vertex $v$ in the graph given by $\chi_E(x, y)$ before and after running procedure* `halve` *on $\chi_E(x, y)$. Then $d'(v) \in \{\lfloor d(v)/2 \rfloor, \lceil d(v)/2 \rceil\}$. The algorithm* `halve` *uses $\mathcal{O}(\log^2 |V|)$ functional operations.*

*Proof.* For the number of functional operations step (2) and step (4) are the most expensive ones. The graph is traversed via iterative squaring in the second step. Since the length of a path is $\mathcal{O}(|E|)$, the number of iterations is $\mathcal{O}(\log |E|) = \mathcal{O}(\log |V|)$, and the quantification over the Boolean variables that encode an edge can also be done using $\mathcal{O}(\log |V|)$ operations ($\mathcal{O}(1)$ functional operations for the quantification of each variable). The number of functional operations in step (4) can be calculated in a similar way.

For the correctness of the algorithm `halve` step (3) is the most interesting one. The directed paths according to the successor relation `SUCC` are edge-disjoint but not vertex disjoint. In step (3) for each directed path according to the successor relation at least one edge is carefully chosen and colored red. The first condition ensures that only edges that belong to the input graph can be chosen. The second and third requirements guarantee that a first red edge is incident to the vertex $x$ with the highest priority on the path. The fourth condition ensures that two arbitrary edges incident to $x$ that are on the same directed path and have an even distance are not both colored red. This condition is sufficient, we do not have to choose only the neighbor of $x$ which has the highest priority because we color the edges afterwards alternately and edges whose distance is odd get the same color anyway. Together with our considerations above we are done. $\square$

The idea for the correctness of `match` is that a (directed) subgraph $P(x, y)$ of the input $\chi_E(x, y)$ is computed for which each vertex has indegree and outdegree at most 1. It consists only of vertex-disjoint open simple paths and trivial cycles. Therefore, the subgraph can be seen as the union of two matchings. By choosing the matching which is the larger one we are done. For this reason we color each path in $P(x, y)$ alternately and we remove the edges that are not red. Since the paths are vertex-disjoint the coloring is easier than the coloring of the edges in the algorithm `halve`. In fact we color the vertices and not the edges. We choose for each directed simple path the first vertex and color afterwards all vertices for which the distance to the first one is even. Then we choose the (directed) edges $(x, y)$ iff the vertex $x$ is red. Finally, we traverse the computed directed subgraph into the corresponding undirected one.

The directed subgraph $P$ is computed in the following way. In each iteration the vertices with degree 1 are determined. For each vertex $x$ adjacent to vertices with degree 1 in the remaining graph one of these vertices $y$ is chosen according to a priority function and $(x, y)$ is added to $P$. Afterwards all edges incident to vertices with degree 1 are deleted and the degree of all vertices is (approximately) halved. Note, that during the computation an edge $(x, y)$ in $P$ can be eliminated later on if $x$ gets another partner that has a higher priority than $y$. At any time each vertex $x$ has at most one partner.

It can be shown that at the beginning of step (8) in `match` at least 1/3 of the input edges are incident to edges defined via $P(x, y)$. The intuition is the following one. An edge $(u, v)$ is not incident to the computed matching iff the edge $(u, v)$ is deleted during the algorithm `halve` and the last edges $(u, u')$ and $(v, v')$ incident to $u$ and $v$ during the **while** loop are eliminated in step (5) of `match` because $u'$ and $v'$ are at the same time adjacent to vertices $x$ and $y$ which have degree 1 and are chosen as partners in the respective iteration of the **while** loop. As a consequence we can conclude that the degree of $u'$ and $v'$ is (approximately) at least twice the degree of $u$ and $v$ in the input graph because Lemma 6 ensures that the degree of each node is (almost) regularly halved in each iteration. Therefore the output of the algorithm `match` is a matching incident to at least 1/6 of the input edges.

**Lemma 7.** *The algorithm* `match` *implicitly computes a matching in an implicitly defined input graph* $G = (V, E)$ *incident to at least* $1/6$ *of the edges in* $E$. *It needs* $\mathcal{O}(\log^3 |V|)$ *functional operations.*

*Proof.* There are $\mathcal{O}(\log |V|)$ iterations of the **while** loop, each of them costs $\mathcal{O}(\log^2 |V|)$ functional operations. The algorithm `halve` is the dominating step during the **while** loop of `match`. Therefore, $\mathcal{O}(\log^3 |V|)$ functional operations are sufficient. The correctness follows from our considerations above. (See also [11].) $\square$

**Algorithm 4.** `match`

```
Input: $\chi_E(x,y)$

(1) ▷ Initialize.
    $\chi_{E'}(x,y) \leftarrow \chi_E(x,y)$; $P(x,y) \leftarrow 0$
(2) while $\chi_{E'}(x,y) \neq 0$ do
(3)     ▷ Determine the vertices of degree at least 2.
        $\textsc{TwoOrMoreNeighbors}(x) \leftarrow (\exists y,z)(NEQ(y,z) \wedge \chi_{E'}(x,y) \wedge \chi_{E'}(x,z))$
(4)     ▷ Set $P(x,y)=1$ iff $y$ has degree 1 and is the partner of $x$.
        $Q(x,y) \leftarrow \chi_{E'}(x,y) \wedge \overline{\textsc{TwoOrMoreNeighbors}(y)}$
        $Q'(x,y) \leftarrow Q(x,y) \wedge \overline{(\exists z)(Q(x,z) \wedge \Pi_{\prec}(x,z,y))}$
        $P(x,y) \leftarrow (P(x,y) \wedge \overline{(\exists z)(Q'(x,z))}) \vee Q'(x,y)$
(5)     ▷ Delete edges incident to vertices of degree 1.
        $\chi_{E'}(x,y) \leftarrow \chi_{E'}(x,y) \wedge \textsc{TwoOrMoreNeighbors}(x) \wedge \textsc{TwoOrMoreNeighbors}(y)$
(6)     ▷ Halve (approximately) the degree of each vertex.
        $\chi_{E'}(x,y) \leftarrow$ halve($\chi_{E'}(x,y)$)
(7) ▷ Add trivial cycles to the computed matching.
    $M_1(x,y) \leftarrow P(x,y) \wedge P(y,x)$
(8) ▷ Color the vertices in the graph given by $P(x,y)$ alternately and
    choose an edge $(x,y)$ iff $x$ is red.
    $\mathtt{RED}(x) \leftarrow (\forall \xi)\overline{(P(\xi,x))}$; $\mathtt{DIST}_0(x,y) \leftarrow P(x,y)$
    for $i = 1,2,...,\log|V|$ do
        $\mathtt{DIST}_i(x,y) \leftarrow (\exists \xi)(\mathtt{DIST}_{i-1}(x,\xi) \wedge \mathtt{DIST}_{i-1}(\xi,y))$
        $\mathtt{RED}(x) \leftarrow \mathtt{RED}(x) \vee (\exists \xi)(\mathtt{RED}(\xi) \wedge \mathtt{DIST}_i(\xi,x))$
    $M_2(x,y) \leftarrow P(x,y) \wedge \mathtt{RED}(x)$
(9) return $M_1(x,y) \vee M_2(x,y) \vee M_2(y,x)$
```

Summarizing, we have shown that `findMaximalBipartiteMatching` uses $\mathcal{O}(\log^4 |V|)$ functional operations for the computation of a maximal matching in an implicitly defined input graph $G = (V, E)$. Adapting the ideas for the decomposition of general graphs into a logarithmic number of bipartite subgraphs [11], our algorithm can be similarly generalized with an additional factor of a logarithmic number of functional operations.

## 4 Concluding Remarks

Our maximal matching algorithm seems to be simple enough to be useful in practical applications. We have shown that maximal matchings can be computed with a polylogarithmic number of functional operations in the implicit setting. Moreover, in [4] we have proved that there exists a graph class for which even the overall running time of our maximal matching algorithm is $\mathcal{O}(\log^3 |V| \log\log |V|)$ and the space usage is $\mathcal{O}(\log^2 |V|)$, where $V$ is the set of vertices of the input graph. One direction for future work is to implement the algorithm and to perform empirical experiments to determine its practical value. It would be interesting to investigate how the performance of the maximal matching algorithm depends on the chosen priority function. Here, we have used a very simple one. The maximal number of Boolean variables on which a function in the maximal matching algorithm depends dominates the overall worst-case bounds for the running time and the space usage. Therefore, another open question is whether we can reduce this number without increasing significantly the number of functional operations. Experimental evaluation of different maximal matching algorithms might be revealing.

The authors would like to thank the anonymous referees for carefully reading the manuscript and making helpful suggestions.

## References

1. Bollig, B.: Exponential space complexity for OBDD-based reachability analysis. Information Processing Letters 110, 924–927 (2010)
2. Bollig, B.: Exponential Space Complexity for Symbolic Maximum Flow Algorithms in 0-1 Networks. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 186–197. Springer, Heidelberg (2010)
3. Bollig, B.: On Symbolic OBDD-Based Algorithms for the Minimum Spanning Tree Problem. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part II. LNCS, vol. 6509, pp. 16–30. Springer, Heidelberg (2010)
4. Bollig, B., Pröger, T.: An efficient implicit OBDD-based algorithm for maximal matchings (2011), http://ls2-www.cs.tu-dortmund.de/~bollig/maxMatching.pdf
5. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers 35(8), 677–691 (1986)
6. Gentilini, R., Piazza, C., Policriti, A.: Computing strongly connected components in a linear number of symbolic steps. In: Proc. of SODA, pp. 573–582 (2003)
7. Gentilini, R., Piazza, C., Policriti, A.: Symbolic graphs: linear solutions to connectivity related problems. Algorithmica 50(1), 120–158 (2008)
8. Goldberg, A.V., Plotkin, S.K., Vaidya, P.M.: Sublinear time parallel algorithms for matching and related problems. Journal of Algorithms 14(2), 180–213 (1993)
9. Hachtel, G.D., Somenzi, F.: A symbolic algorithm for maximum flow in $0-1$ networks. Formal Methods in System Design 10, 207–219 (1997)
10. Jájá, J.: An introduction to parallel algorithms. Addison-Wesley Publishing Company (1992)
11. Kelsen, P.: An optimal parallel algorithm for maximal matching. Information Processing Letters 52(4), 223–228 (1994)
12. Sawitzki, D.: Implicit Flow Maximization by Iterative Squaring. In: Van Emde Boas, P., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2004. LNCS, vol. 2932, pp. 301–313. Springer, Heidelberg (2004)
13. Sawitzki, D.: The Complexity of Problems on Implicitly Represented Inputs. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 471–482. Springer, Heidelberg (2006)
14. Sawitzki, D.: Implicit simulation of FNC algorithms. Tech. Rep. TR07-028, ECCC Report (2007)
15. Wegener, I.: Branching programs and binary decision diagrams: theory and applications. SIAM (2000)
16. Woelfel, P.: Symbolic topological sorting with OBDDs. J. Discrete Algorithms 4(1), 51–71 (2006)

# Strong Termination for Gap-Order Constraint Abstractions of Counter Systems

Laura Bozzelli

Technical University of Madrid (UPM), 28660 Boadilla del Monte, Madrid, Spain
laura.bozzelli@fi.upm.es

**Abstract.** We address termination analysis for the class of *gap-order constraint systems* (GCS), an (infinitely-branching) abstract model of counter machines recently introduced in [8], in which constraints (over $\mathbb{Z}$) between the variables of the source state and the target state of a transition are *gap-order constraints* (GC) [18]. GCS extend monotonicity constraint systems [4], integral relation automata [9], and constraint automata in [12]. Since GCS are infinitely-branching, termination does not imply *strong termination*, i.e. the existence of an upper bound on the lengths of the runs from a given state. We show the following: (1) checking strong termination for GCS is decidable and PSPACE-complete, and (2) for each control location of the given GCS, one can build a GC representation of the set of variable valuations from which strong termination does *not* hold.

## 1 Introduction

**Abstractions of Counter Systems.** One standard approach in formal analysis is the abstraction based one: the analysis is performed on an *abstraction* of the given system, specified in some weak (non-complete) computational formalism for which checking the properties of interest is decidable. The relation between the abstraction and the concrete system is usually specified as a semantic over-approximation. This ensures that the approach is conservative, by giving a decision procedure that (for correct systems) is sound but in general incomplete. With regard to the class of counter systems, a widely investigated complete computational model, interesting abstractions have been studied, for which meaningful classes of verification problems have been shown to be decidable. Many of these abstractions are in fact restrictions: examples include Petri nets [16], reversal-bounded counter machines [14], and flat counter systems [10]. Genuine abstractions are obtained by approximating counting operations by non-functional fragments of Presburger constraints between the variables of the current state and the variables of the next state. As a consequence of this abstraction, the set of successors of a state is potentially infinite. Examples include the class of Monotonicity Constraint Systems (MCS) [4] and its variants, like constraint automata in [12], and integral relation automata (IRA) [9], for which the (monotonicity) constraints (MC) are boolean combinations of inequalities of the form $u < v$ or $u \leq v$, where $u$ and $v$ range over variables or integer constants. MCS and their subclasses (namely, *size-change systems*) have found important applications for automated termination proofs of functional programs (see e.g. [4]). Richer classes of non-functional fragments of Presburger constraints have been investigated, e.g. difference bound constraints [11], and their extension, namely octagon relations [6], where it is shown that the transitive closure of a single constraint is Presburger definable.

Recently, an (infinitely-branching) abstract model of counter systems, namely *gap-order constraint systems* (GCS), has been introduced [8], where the constraints (over $\mathbb{Z}$) between the variables of the source state and the target state of a transition are (transitional) *gap-order constraints* (GC) [18]. These constraints are positive boolean combinations of inequalities of the form $u - v \geq k$, where $u, v$ range over variables and integer constants and $k$ is a natural number. Thus, GC can express simple relations on variables such as lower and upper bounds on the values of individual variables; and equality, and gaps (minimal differences) between values of pairs of variables. GC have been introduced in the field of constraint query languages (constraint Datalog) for deductive databases [18], and also have found applications in the analysis of safety properties for parameterized systems [1,2] and for determining state invariants in counter systems [13]. As pointed out in [2], using GC for expressing the enabling conditions of transitions allows to handle a large class of protocols, where the behavior depends on the relative ordering of values among variables, rather than the actual values of these variables. GCS *strictly* extend MCS. This because GC extend MC and, differently from MC, are closed under existential quantification (but not under negation). Hence, GC are closed under composition (which captures the reachability relation for a fixed path in the control graph). Note that if we extend the constraint language of GCS by allowing either negation, or constraints of the form $u - v \geq -k$, with $k \in \mathbb{N}$, then the resulting class of systems can trivially emulate Minsky counter machines, leading to undecidable basic decision problems.

**Our contribution.** We address termination analysis of GCS. Since GCS are infinitely-branching, termination (i.e., the non-existence of states from which there is an infinite run) does not imply the existence of an upper bound on the lengths of the runs from a given state. The fulfillment of this last condition, we call *strong termination*, can be a necessary requirement in some contexts, such as running-time analysis [3] for infinitely-branching formalisms. Checking usual termination for GCS is known to be decidable and PSPACE-complete [8]. In this paper, by a non-trivial extension of the approach used in [8], we establish the following results:

– (1) For each control location of the given GCS, it is possible to compute a GC representation of the set of variable valuations from which strong termination does *not* hold, and (2) checking strong termination and strong termination for a designated state in GCS are decidable and PSPACE-complete.

Our approach is as follows. First, we consider a subclass of GCS, called *simple* GCS: we establish our first result for simple GCS, and provide a polynomial-time checkable condition for verifying strong termination in simple GCS, which is independent on the size of the lower bounds $k$ in GC. Second, for a given unrestricted GCS $\mathcal{S}$, we show that it is possible to construct a finite family $\mathcal{F}$ of *simple* GCS such that the union of the sets of strongly-terminating states of the single components in $\mathcal{F}$ correspond to the set of strongly-terminating states of $\mathcal{S}$. Then, we show that it is possible to compute separately and in exponential time suitable abstractions of the simple GCS in $\mathcal{F}$ (we are not able to give an upper bound on the size of $\mathcal{F}$), which preserve the fulfillment of the above polynomial-time checkable condition for simple GCS. This leads to exponential-time procedures for solving strong termination and strong termination for a designated state in GCS. Finally, we show that in fact, the considered problems are PSPACE-complete.

A potential application of our results is to use them as basic tool in running-time analysis (based on GCS abstraction) of infinitely-branching computational systems. Note that concurrent open systems are usually infinitely-branching because of the ongoing interaction with an unpredictable environment, and GCS can be used to abstractly model such an interaction.

**Related work.** Strong termination has been addressed in [5]. There, it is shown that for the subclass of MCS where integer constants are disallowed except for 0, checking strong termination is PSPACE-complete. Note that our results extend the above result in two directions: (1) we consider a strict extension of MCS, namely GCS, and (ii) our symbolic algorithm builds a GC representation of the set of non-strongly-terminating states, a very substantial information compared to the algorithm in [5] (see also [4]). For example, by using such a decidable finite representation one can check whether two GCS have the same set of strongly-terminating states. Due to space reasons, many proofs are omitted and can be found in [7].

## 2 Preliminaries

Let $\mathbb{Z}$ (resp., $\mathbb{N}$) be the set of integers (resp., natural numbers). We fix a finite set $Var = \{x_1, \ldots, x_r\}$ of variables, a finite set of constants $Const \subseteq \mathbb{Z}$ such that $0 \in Const$, and a fresh copy of $Var$, $Var' = \{x'_1, \ldots, x'_r\}$. For an arbitrary finite set of variables $V$, an (integer) *valuation* over $V$ is a mapping of the form $\nu : V \to \mathbb{Z}$, assigning to each variable in $V$ an integer value. For $V' \subseteq V$, $\nu_{V'}$ denotes the restriction of $\nu$ to $V'$. For a valuation $\nu$, by convention, we define $\nu(c) = c$ for all $c \in \mathbb{Z}$.

**Definition 1.** [18] A *gap-order constraint* (GC) over $V$ and $Const$ is a conjunction $\xi$ of inequalities of the form $u - v \geq k$, where $u, v \in V \cup Const$ and $k \in \mathbb{N}$. W.l.o.g. we assume that for all $u, v \in V \cup Const$, there is at most one conjunct in $\xi$ of the form $u - v \geq k$ for some $k$. A valuation $\nu : V \to \mathbb{Z}$ *satisfies* $\xi$ if for each conjunct $u - v \geq k$ of $\xi$, $\nu(u) - \nu(v) \geq k$. We denote by $Sat(\xi)$ the set of such valuations.

**Definition 2.** [9,8] A (*gap-order*) *monotonicity graph* (MG) over $V$ and $Const$ is a directed weighted graph $G$ with set of vertices $V \cup Const$ and edges $u \xrightarrow{k} v$

labeled by natural numbers $k$, and s.t.: if $u \xrightarrow{k} v$ and $u \xrightarrow{k'} v$ are edges of $G$, then $k = k'$. The set $Sat(G)$ of *solutions* of $G$ is the set of valuations $\nu$ over $V$ s.t. for each $u \xrightarrow{k} v$ in $G$, $\nu(u) - \nu(v) \geq k$. GC and MG are equivalent formalisms since there is a trivial linear-time computable bijection assigning to each GC $\xi$ an MG $G(\xi)$ such that $Sat(G(\xi)) = Sat(\xi)$.

The notation $G \models u < v$ means that there is an edge in $G$ from $v$ to $u$ with weight $k > 0$. Moreover, $G \models u \leq v$ means that there is an edge of $G$ from $v$ to $u$, and $G \models u = v$ means $G \models u \leq v$ and $G \models v \leq u$. Also, we write $G \models u_1 \lhd_1 \ldots \lhd_{n-1} u_n$ to mean that $G \models u_i \lhd_i u_{i+1}$ for each $1 \leq i < n$, where $\lhd_i \in \{<, \leq, =\}$. A *transitional* GC (resp., *transitional* MG) is a GC (resp., MG) over $Var \cup Var'$ and $Const$. For valuations $\nu, \nu' : Var \to \mathbb{Z}$, we denote by $\nu \oplus \nu'$ the valuation over $Var \cup Var'$ defined as follows: $(\nu \oplus \nu')(x_i) = \nu(x_i)$ and $(\nu \oplus \nu')(x'_i) = \nu'(x_i)$ for $i = 1, \ldots, r$.

**Definition 3.** [8] A *gap-order constraint system* (GCS) over $Var$ and $Const$ is a finite directed labeled graph $\mathcal{S}$ such that each edge is labeled by a *transitional* GC. We denote by $Q(\mathcal{S})$ the set of vertices in $\mathcal{S}$, called *control points*, and by $E(\mathcal{S})$ the set of edges.

The semantics of a GCS $\mathcal{S}$ is given by an infinite directed graph $[\![\mathcal{S}]\!]$ defined as:

– the vertices of $[\![\mathcal{S}]\!]$, called *states* of $\mathcal{S}$, are the pairs of the form $(q, \nu)$, where $q$ is a control point of $\mathcal{S}$ and $\nu : Var \to \mathbb{Z}$ is a valuation over $Var$;
– there is an edge in $[\![\mathcal{S}]\!]$ from $(q, \nu)$ to $(q', \nu')$ iff there is a (labeled) edge in $\mathcal{S}$ of the form $q \xrightarrow{\xi} q'$ such that $\nu \oplus \nu' \in Sat(\xi)$. We say that the edge of $[\![\mathcal{S}]\!]$ from $(q, \nu)$ to $(q', \nu')$ is an *instance* of the edge $q \xrightarrow{\xi} q'$ of $\mathcal{S}$.

For a finite path $\wp$ of a GCS $\mathcal{S}$, $s(\wp)$ and $t(\wp)$ denote the source and target control points of $\wp$. For a finite path $\wp$ and a path $\wp'$ such that $t(\wp) = s(\wp')$, the composition of $\wp$ and $\wp'$, written $\wp\wp'$, is defined as usual. A path of $[\![\mathcal{S}]\!]$ is called a *run* of $\mathcal{S}$. The length $|\wp|$ (resp., $|\pi|$) of a path $\wp$ (resp., run $\pi$) of $\mathcal{S}$ is defined in the standard way. A *non-null* path of $\mathcal{S}$ is a path of $\mathcal{S}$ of non-null length. Let $\wp = q_0 \xrightarrow{\xi_0} q_1 \xrightarrow{\xi_1} q_2, \ldots$ be a path of $\mathcal{S}$. A run $\pi$ of $\mathcal{S}$ is an *instance* of $\wp$ if $\pi$ is of the form $\pi = (q_0, \nu_0) \to (q_1, \nu_1) \to (q_2, \nu_2), \ldots$ and for each $i$, $(q_i, \nu_i) \to (q_{i+1}, \nu_{i+1})$ is an instance of $q_i \xrightarrow{\xi_i} q_{i+1}$. A state $s$ of $\mathcal{S}$ is *terminating* if there is *no* infinite run of $\mathcal{S}$ starting from $s$. A state $s$ of $\mathcal{S}$ is *unbounded* if the set of lengths of the *finite* runs of $\mathcal{S}$ starting from $s$ is unbounded (equivalently, infinite). A state $s$ of $\mathcal{S}$ is *strongly terminating* if it is *not* unbounded. Since $[\![\mathcal{S}]\!]$ is infinitely-branching, termination and strong termination are distinct concepts. In particular, strong termination implies termination, but the vice-versa in general does not hold.

*Example 1.* Consider the GCS $\mathcal{S}$ consisting of two self-loops $q \xrightarrow{\xi} q$ and $q \xrightarrow{\xi'} q$, where: $\xi = [(x'_1 < x_1) \wedge (x_1 \geq 0) \wedge (x_2 \geq 0)]$ and $\xi' = [(x'_1 = x_1) \wedge (x'_2 < x_2) \wedge (x_1 \geq 0) \wedge (x_2 \geq 0)]$. Each state of $\mathcal{S}$ is terminating since along any run, the pair $(x_1, x_2)$ decreases strictly w.r.t. the lexicographic order (over $\mathbb{N} \times \mathbb{N}$). On the other hand, one can easily check that each state $s = (q, \nu)$ with $\nu(x_1) > 0$ and $\nu(x_2) \geq 0$ is unbounded.

Since we use MG representations to manipulate GC, we assume that the edge-labels in GCS are transitional MG. A set $U$ of states of a GCS $\mathcal{S}$ is MG *representable* if there is a family $\{\mathcal{G}_q\}_{q \in Q(\mathcal{S})}$ of finite sets of MG over $Var$ and $Const$ such that $\bigcup_{G \in \mathcal{G}_q} Sat(G) = \{\nu \mid (q, \nu) \in U\}$ for each $q \in Q(\mathcal{S})$.

**Investigated problems.** For a GCS $\mathcal{S}$, we denote by $Inf_{\mathcal{S}}$ the set of non-terminating states $s$ of $\mathcal{S}$ and by $Unb_{\mathcal{S}}$ the set of unbounded states of $\mathcal{S}$. Note that $Inf_{\mathcal{S}} \subseteq Unb_{\mathcal{S}}$. Moreover, for $q \in Q(\mathcal{S})$, we denote by $Inf^q_{\mathcal{S}}$ (resp., $Unb^q_{\mathcal{S}}$) the set of states in $Inf_{\mathcal{S}}$ (resp., $Unb_{\mathcal{S}}$) of the form $(q, \nu)$ for some valuation $\nu$. The termination problem, i.e. checking whether $Inf_{\mathcal{S}} = \emptyset$ for a given GCS $\mathcal{S}$, is known to be decidable and PSPACE-complete [8]. In this paper, we address strong termination:

– *Strong Termination Problem*: given a GCS $\mathcal{S}$, is the set $Unb_{\mathcal{S}}$ empty?
– *Strong Termination Problem w.r.t. a designated state*: given a GCS $\mathcal{S}$ and a state $s$ of $\mathcal{S}$, does $s \notin Unb_{\mathcal{S}}$ hold?

## 2.1 Properties of Monotonicity Graphs

We recall some basic properties of MG [9]. Furthermore, we recall a sound and complete (w.r.t. satisfiability) approximation scheme of MG [8] such that basic operations on MG preserve soundness and completeness of this approximation.

A MG $G$ is *satisfiable* if $Sat(G) \neq \emptyset$. Let $G$ be a MG over $V$ and $Const$. For $V' \subseteq V$, *the restriction of $G$ to $V'$*, written $G_{V'}$, is the MG given by the subgraph of $G$ whose set of vertices is $V' \cup Const$. For all vertices $u, v$ of $G$, we denote by $p_G(u, v)$ the least upper bound (possibly $\infty$) of the weight sums on all paths in $G$ from $u$ to $v$ (we set $p_G(u, v) = -\infty$ if there is no such a path). The MG $G$ is *normalized* iff: (1) for all vertices $u, v$ of $G$, if $p_G(u, v) > -\infty$, then $p_G(u, v) \neq \infty$ and $u \xrightarrow{p_G(u,v)} v$ is an edge of $G$, and (2) for all constants $c_1, c_2 \in Const$, $p_G(c_1, c_2) \leq c_1 - c_2$. Intuitively, a normalized MG is a MG closed under logical consequence.

**Proposition 4.** [9] *Let $G$ be a* MG *over $V$ and $Const$. Then:*

1. *If $G$ is* normalized*, then $G$ is satisfiable. Moreover, for all $V' \subseteq V$, every solution of $G_{V'}$ can be extended to a whole solution of $G$.*
2. *One can check in polynomial time if $G$ is satisfiable. Moreover, if $G$ is satisfiable, then one can build in polynomial time an equivalent* normalized MG *$\overline{G}$ (i.e., $Sat(\overline{G}) = Sat(G)$), called the* closure *of $G$.*

According to Proposition 4, for a satisfiable MG $G$, we denote by $\overline{G}$ the closure of $G$. Moreover, for all unsatisfiable MG $G$ over $V$ and $Const$, we use an unique closure corresponding to some MG $G_{nil}$ over $V$ and $Const$ such that $(G_{nil})_{\emptyset}$ is unsatisfiable. The following known result [9] essentially asserts that MG (or, equivalently, GC) are closed under intersection and existential quantification.

**Proposition 5.** [9] *Let $G$ be a* MG *on $V$ and $Const$ and $G'$ be a* MG *on $V'$ and $Const$.*

1. ***Intersection:*** *one can build in polynomial time a* MG *over $V \cup V'$ and $Const$, written $G \bigotimes G'$, s.t. for $\nu : V \cup V' \to \mathbb{Z}$, $\nu \in Sat(G \bigotimes G')$* iff *$\nu|_V \in Sat(G)$ and $\nu|_{V'} \in Sat(G')$. Hence, for $V = V'$, $Sat(G \bigotimes G') = Sat(G) \cap Sat(G')$.*
2. ***Composition:*** *assume that $G$ and $G'$ are transitional* MG*. Then, one can build in polynomial time a transitional* MG*, written $G \bullet G'$, s.t. for all $\nu, \nu' : Var \to \mathbb{Z}$, $\nu \oplus \nu' \in Sat(G \bullet G')$ iff $\nu \oplus \nu'' \in Sat(G)$ and $\nu'' \oplus \nu' \in Sat(G')$ for some $\nu'' : Var \to \mathbb{Z}$. Moreover, the composition operator $\bullet$ is associative.*

**Approximation scheme:** let $K$ stand for $max(\{|c_1 - c_2| + 1 \mid c_1, c_2 \in Const\})$. Note that $K > 0$. For each $h \in \mathbb{N}$, let $\lfloor h \rfloor_K = min(\{h, K\})$.

**Definition 6 ($K$-bounded MG).** [8] A MG is *$K$-bounded* iff for each of its edges $u \xrightarrow{k} v$, $k \leq K$. For a MG $G$ on $V$ and $Const$, $\lfloor G \rfloor_K$ denotes the $K$-bounded MG over $V$ and $Const$ obtained from $G$ by replacing each edge $u \xrightarrow{k} v$ of $G$ with the edge $u \xrightarrow{\lfloor k \rfloor_K} v$.

**Proposition 7.** [8] *Let $G$ be a* MG *over $V$ and $Const$. Then, $G$ is satisfiable iff $\lfloor G \rfloor_K$ is satisfiable. Moreover, $\lfloor \overline{G} \rfloor_K = \lfloor\ \overline{\lfloor G \rfloor_K}\ \rfloor_K$. Furthermore, for transitional* MG *$G_1$ and $G_2$, $\lfloor G_1 \bullet G_2 \rfloor_K = \lfloor \lfloor G_1 \rfloor_K \bullet \lfloor G_2 \rfloor_K \rfloor_K$.*

## 2.2 Results on the Reachability Relation in GCS

We recall some constructive results on the reachability relation in GCS [8].

**Definition 8.** *A transitional* MG *$G$ is said to be* complete *if:*

- *for all vertices $u$ and $v$, $G \models u \leq v \Rightarrow G \models u \lhd v$ for some $\lhd \in \{<, =\}$;*
- *for all $u, v \in Var \cup Const$, either $G \models u \leq v$ or $G \models v \leq u$;*
- *for all $u, v \in Var' \cup Const$, either $G \models u \leq v$ or $G \models v \leq u$.*

Intuitively, complete transitional MG induce a total ordering on the set of vertices in $Var \cup Const$ (resp., $Var' \cup Const$). A GCS $\mathcal{S}$ is *complete* iff each MG in $\mathcal{S}$ is complete. Fix a *complete* GCS $\mathcal{S}$. For a finite path $\wp$ of $\mathcal{S}$, the *reachability relation w.r.t.* $\wp$, denoted by $\leadsto_\wp$, is the binary relation on the set of valuations over $Var$ defined as: for all $\nu, \nu' : Var \to \mathbb{Z}$, $\nu \leadsto_\wp \nu'$ iff there is a run of $\mathcal{S}$ from $(s(\wp), \nu)$ to $(t(\wp), \nu')$ which is an instance of the path $\wp$. For a transitional MG $G$, *$G$ characterizes the reachability relation* $\leadsto_\wp$ iff $Sat(G) = \{\nu \oplus \nu' \mid \nu \leadsto_\wp \nu'\}$. We associate to each non-null finite path $\wp$ of $\mathcal{S}$ a transitional MG $G_\wp$ and a transitional $K$-bounded MG $G^{bd}_\wp$, defined as:

- $\wp = q \xrightarrow{G} q'$: $G_\wp = \overline{G}$ and $G^{bd}_\wp = \lfloor\ \overline{G}\ \rfloor_K$;
- $\wp = \wp' \wp''$, $|\wp'| > 0$, and $\wp'' = q \xrightarrow{G} q'$: $G_\wp = G_{\wp'} \bullet G$ and $G^{bd}_\wp = \lfloor G^{bd}_{\wp'} \bullet \lfloor G \rfloor_K \rfloor_K$.

The following results have been shown in [8]. Theorem 9 can be easily deduced, while Theorem 10 is a refinement of a result in [9] establishing that for a GCS $\mathcal{S}$, the reflexive transitive closure of the transition relation of $[\![\mathcal{S}]\!]$ is effectively $\mathsf{GC}$ definable (a similar result can be found in [18], where it is shown that for Datalog queries with $\mathsf{GC}$, there is a closed form evaluation). Note that in Theorem 10, we are not able to give an upper bound on the cardinality of the set $\mathcal{P}_{\mathcal{S}}$.

**Theorem 9.** [8] *For a non-null finite path $\wp$ of $\mathcal{S}$, $G_\wp = \overline{G_\wp}$, $G_\wp^{bd} = \lfloor G_\wp \rfloor_K$ and is complete, and $G_\wp$ is complete and characterizes the reachability relation $\rightsquigarrow_\wp$. Moreover, the set $\{(\lfloor G_\wp \rfloor_K, s(\wp), t(\wp)) \mid \wp$ is a non-null finite path and $G_\wp$ is satisfiable$\}$. has size bounded by $O(|Q(\mathcal{S})|^2 \cdot (K+2)^{(2|Var|+|Const|)^2})$ and can be computed in time $O(|E(\mathcal{S})| \cdot |Q(\mathcal{S})|^2 \cdot (K+2)^{(2|Var|+|Const|)^2})$.*

**Theorem 10.** [8] *One can compute a* finite *set $\mathcal{P}_{\mathcal{S}}$ of non-null finite paths of $\mathcal{S}$ such that: for each non-null finite path $\wp'$ of $\mathcal{S}$ from $q$ to $q'$, there is a path $\wp \in \mathcal{P}_{\mathcal{S}}$ from $q$ to $q'$ so that $\lfloor G_{\wp'} \rfloor_K = \lfloor G_\wp \rfloor_K$, and $\rightsquigarrow_{\wp'}$ implies $\rightsquigarrow_\wp$ (i.e., $\rightsquigarrow_{\wp'}$ is contained in $\rightsquigarrow_\wp$).*

## 3 Strong Termination for Simple GCS

In this section, we solve strong termination for a restricted class of $\mathsf{GCS}$ introduced in [8]. A $\mathsf{MG}$ $G$ is *weakly normalized* if for all vertices $u, v$, $p_G(u,v) \geq 0$ (resp., $p_G(u,v) > 0$) implies $G \models v \leq u$ (resp., $G \models v < u$). Note that $G$ is weakly normalized iff $\lfloor G \rfloor_K$ is weakly normalized.
A transitional $\mathsf{MG}$ $G$ is *(weakly) idempotent* iff $\lfloor G \bullet G \rfloor_K = \lfloor G \rfloor_K$.

**Definition 11 (Simple GCS).** [8] *A simple* $\mathsf{GCS}$ *is a* $\mathsf{GCS}$ *consisting of just two edges of the form $q_0 \xrightarrow{G_0} q$ and $q \xrightarrow{G} q$ such that $q_0 \neq q$. Moreover, we require that $G_0 \bullet G$ is satisfiable, $G_0$ and $G$ are complete and* weakly *normalized, and $G$ is idempotent.*

To present our results on simple $\mathsf{GCS}$, we need additional definitions.

**Definition 12 (lower and upper variables).** [8] We denote by $MAX$ (resp., $MIN$) the maximum (resp., minimum) of $Const$. For a transitional $\mathsf{MG}$ $G$ and $y \in Var \cup Var'$, $y$ is a *lower* (resp., *upper*) *variable* of $G$ if $G \models y < MIN$ (resp., $G \models MAX < y$). Moreover, $y$ is a *bounded variable* of $G$ if $G \models MIN \leq y$ and $G \models y \leq MAX$.

**Definition 13.** A transitional $\mathsf{MG}$ is *balanced* iff for all $u, v \in Var \cup Const$ and $\lhd \in \{<, =\}$, $G \models u \lhd v$ iff $G \models u' \lhd v'$ (where for $u \in Var \cup Const$, we write $u'$ to denote the corresponding variable in $Var'$ if $u \in Var$, and $u$ itself otherwise).

Intuitively, a transitional $\mathsf{MG}$ is balanced if the partial orders on $Var \cup Const$ and $Var' \cup Const$ induced by the $\mathsf{MG}$ are the same. Fix a simple $\mathsf{GCS}$ $\mathcal{S}$ with edges $q_0 \xrightarrow{G_0} q$ and $q \xrightarrow{G} q$. Since $G$ is idempotent, by the associativity of $\bullet$ and Proposition 7, we obtain that for each $k \geq 1$, $\lfloor G_0 \bullet \underbrace{G \bullet \ldots \bullet G}_{k \text{ times}} \rfloor_K = \lfloor G_0 \bullet G \rfloor_K$.

Hence, $G_0 \bullet \underbrace{G \bullet \ldots \bullet G}_{k \text{ times}}$ and $\underbrace{G \bullet \ldots \bullet G}_{k \text{ times}}$ are satisfiable for each $k \geq 1$. Since $G$ is complete and $G \bullet G$ is satisfiable, by Proposition 5(2) it follows that $G$ is *balanced* as well. Moreover, since $G$ is satisfiable and complete, a variable $y \in Var \cup Var'$ is or a lower variable, or an upper variable, or a bounded variable of $G$, where the "or" is exclusive. We denote by $\text{L}_1, \ldots, \text{L}_N$ (resp., $\text{U}_1, \ldots, \text{U}_M$) the lower (resp., the upper) variables of $G$ in $Var$, and by $\text{B}_1, \ldots, \text{B}_H$ the bounded variables of $G$ in $Var$. Hence, we can assume that

$$G \models \text{L}_1 \lhd_2 \ldots \lhd_N \text{L}_N < \text{B}_1 \lhd'_2 \ldots \lhd'_H \text{B}_H < \text{U}_1 \lhd''_2 \ldots \lhd''_M \text{U}_M$$

where $\lhd_2 \ldots \lhd_N, \lhd'_2 \ldots \lhd'_H, \lhd''_2 \ldots \lhd''_M \in \{<, =\}$. Since $G$ is balanced it follows that the lower variables (resp., upper variables) of $G$ in $Var'$ are $\text{L}'_1, \ldots, \text{L}'_N$ (resp., $\text{U}'_1, \ldots, \text{U}'_M$), and the bounded variables of $G$ in $Var'$ are $\text{B}'_1, \ldots, \text{B}'_H$. Moreover,

$$G \models \text{L}'_1 \lhd_2 \ldots \lhd_N \text{L}'_N < \text{B}'_1 \lhd'_2 \ldots \lhd'_H \text{B}'_H < \text{U}'_1 \lhd''_2 \ldots \lhd''_M \text{U}'_M$$

Now, we recall a polynomial-time checkable condition on simple GCS [8].

**Definition 14 (termination condition).** [8] We say that $G$ satisfies the termination condition iff one of the following holds:

***lower variables:*** either $G \models \text{L}_i < \text{L}'_i$ *for some* $1 \leq i \leq N$,
or $G \models \text{L}_i = \text{L}'_i$ *and* $G \models \text{L}'_j < \text{L}_j$ *for some* $1 \leq i < j \leq N$.
***upper variables:*** either $G \models \text{U}'_i < \text{U}_i$ *for some* $1 \leq i \leq M$,
or $G \models \text{U}_j = \text{U}'_j$ *and* $G \models \text{U}_i < \text{U}'_i$ *for some* $1 \leq i < j \leq M$.

Intuitively, the above condition asserts that either there is a lower (resp., upper) variable of $G_{Var}$ whose value strictly increases (resp., decreases) along each run of $\mathcal{S}$, or there are two lower (resp., upper) variables of $G_{Var}$ such that the absolute value of their difference strictly decreases along each run of $\mathcal{S}$. Let $\mathcal{TC}$ be the class of simple GCS satisfying the termination condition. By Definition 14, we easily obtain the following.

**Proposition 15.** *If $\mathcal{S} \in \mathcal{TC}$, then $Unb^q_{\mathcal{S}} = \emptyset$ (i.e., the set of unbounded states of $\mathcal{S}$ of the form $(q, \nu)$ is empty) and $Inf_{\mathcal{S}} = \emptyset$.*

**Theorem 16.** [8] *Let $\mathcal{S} \notin \mathcal{TC}$. Then, $Inf_{\mathcal{S}}$ is* MG *representable and one can construct a* MG *representation of $Inf_{\mathcal{S}}$.*

Moreover, we can show the following non-trivial result (a proof is in [7]).

**Theorem 17.** *If $\mathcal{S} \notin \mathcal{TC}$, then $Unb_{\mathcal{S}} = Inf_{\mathcal{S}}$ and $Inf^{q_0}_{\mathcal{S}} \neq \emptyset$.*

### 3.1 $\mathcal{S}$ Satisfies the Termination Condition

We define a polynomial-time checkable condition on simple GCS which implies the termination condition. We will show that it characterizes the simple GCS such that $Inf_{\mathcal{S}} = \emptyset$ and $Unb_{\mathcal{S}} \neq \emptyset$.

**Definition 18 (unboundedness condition).** We say that $\mathcal{S}$ *satisfies the unboundedness condition* iff $\mathcal{S} \in \mathcal{TC}$ and **none** of the following properties holds:

***lower variables:*** *there is a lower variable* $\text{L}$ *of* $G_0$ *in* $Var$ *such that*

- either $G_0 \models \text{L} \leq \text{L}'_i$ *and* $G \models \text{L}_i < \text{L}'_i$ *for some* $1 \leq i \leq N$*,*
- or $G_0 \models \text{L} \leq \text{L}'_i$*,* $G \models \text{L}_i = \text{L}'_i$*, and* $G \models \text{L}'_j < \text{L}_j$ *for some* $1 \leq i < j \leq N$*.*

***upper variables:*** *there is an upper variable* $\text{U}$ *of* $G_0$ *in* $Var$ *such that*

- either $G_0 \models \text{U}'_i \leq \text{U}$ *and* $G \models \text{U}'_i < \text{U}_i$ *for some* $1 \leq i \leq M$*,*
- or $G_0 \models \text{U}'_i \leq \text{U}$*,* $G \models \text{U}_i = \text{U}'_i$*, and* $G \models \text{U}'_j > \text{U}_j$ *for some* $1 \leq j < i \leq M$*.*

Intuitively, the unboundedness condition implies the termination condition and asserts that: (i) there is no lower (resp., upper) variable of $G_{Var'}$ (or equivalently of $G_{Var}$) whose value strictly increases (resp., decreases) along each run of $\mathcal{S}$ and at the same time is lower (resp., upper) bounded by a lower (resp., upper) variable of $G_0$ in $Var$, (ii) there is no pair of lower (resp., upper) variables of $G_{Var'}$ such that the absolute value of their difference strictly decreases along each run of $\mathcal{S}$, and at the same time is lower (resp., upper) bounded by a lower (resp., upper) variable of $G_0$ in $Var$.
Let $\mathcal{UC}$ be the class of simple GCS satisfying the unboundedness condition. Note that $\mathcal{UC} \subseteq \mathcal{TC}$. The following result easily follows from Proposition 15 and Definition 18.

**Proposition 19.** *If* $\mathcal{S} \in \mathcal{TC} \setminus \mathcal{UC}$*, then* $Unb_{\mathcal{S}} = Inf_{\mathcal{S}} = \emptyset$*.*

It remains to consider the case when $\mathcal{S} \in \mathcal{UC}$. We define two integers $L$ and $U$ as follows: $L$ is the smallest $1 \leq i \leq N$ such that $G_0 \models \text{L} \leq \text{L}'_i$ for some lower bound variable $\text{L}$ of $G_0$ in $Var$ and $G \models \text{L}_i = \text{L}'_i$ (if such an $i$ does not exist, we set $L = N + 1$). Finally, $U$ is the greatest $1 \leq i \leq M$ such that $G_0 \models \text{U}'_i \leq \text{U}$ for some upper variable $\text{U}$ of $G_0$ in $Var$ and $G \models \text{U}'_i = \text{U}_i$ (if such an $i$ does not exist, we set $U = 0$). Note that $1 \leq L \leq N + 1$ and $0 \leq U \leq M$. The set of *unconstrained variables* in $Var$, written $Unc$, consists of the lower variables $\text{L}_i$ such that $1 \leq i < L$ and the upper variables $\text{U}_j$ such that $U < j \leq M$. We denote by $Unc'$ the corresponding subset in $Var'$. Since $G_0 \bullet G$ is satisfiable, $G_0$ is complete, and $G$ is balanced and complete, it follows that the sets of upper (resp., lower) variables of $G$ and $G_0$ in $Var'$ coincide, and the orderings induced by $G$ and $G_0$ coincide. By Definitions 18 and 14, if $\mathcal{S} \in \mathcal{UC}$, then either (1) there is an upper variable $\text{U}_j$ such that $G_0 \not\models \text{U}'_j \leq \text{U}$ for each upper variable $\text{U}$ of $G_0$ in $Var$, or (2) there is a lower variable $\text{L}_j$ such that $G_0 \not\models \text{L} \leq \text{L}'_j$ for each lower variable $\text{L}$ of $G_0$ in $Var$. Thus, by the above considerations, it follows that $Unc \neq \emptyset$ if $\mathcal{S} \in \mathcal{UC}$. The following lemma directly follows from definition of $Unc$.

**Lemma 20.** *For a valuation* $\nu_0 : Var \to \mathbb{Z}$*, the set of valuations* $\{\nu_{(Var \setminus Unc)} \mid (q, \nu)$ *is reachable from* $(q_0, \nu_0)$ *in* $[\![\mathcal{S}]\!]\}$ *is finite.*

Let $v_L = \text{L}_L$ if $L < N + 1$, and $v_L = MIN$ otherwise. Moreover, let $v_U = \text{U}_U$ if $U > 0$, and $v_U = MAX$ otherwise. For a valuation $\nu$ over $Var$ such that $\nu \in Sat(G_{Var})$, we denote by $N_\nu$ the natural number defined as follows:

$$\begin{aligned} N_\nu = min(&\{\nu(x) - \nu(v) \mid x \in Unc,\ v \in Unc \cup \{v_L, v_U\} \text{ and } G \models v < x\} \cup \\ &\{\nu(v) - \nu(x) \mid x \in Unc,\ v \in Unc \cup \{v_L, v_U\} \text{ and } G \models x < v\}). \end{aligned}$$

where the minimum of the empty set is 0. Let $\Delta \in \mathbb{N}$ be the maximum of the set of edge weights of $G$. Now, we give two technical lemmata whose proofs are in [7]. Lemma 21 ensures the following crucial property: let $\mathcal{S} \in \mathcal{UC}$ and $\pi = (q,\nu_0)\ldots(q,\nu_m)$ be a run of $\mathcal{S}$ of non-null length such that $\nu_0$ and $\nu_m$ agree on $Var \setminus Unc$. Then, for all $k \geq 1$ and valuations $\nu'_0 \in Sat(G_{Var})$ such that $\nu'_0$ and $\nu_0$ agree on $Var \setminus Unc$, if $N_{\nu'_0}$ *is sufficiently large*, then there is also a run of length greater than $k$ from $(q,\nu'_0)$ (intuitively, obtained by pumping the pseudo-cycle $\pi$).

**Lemma 21 (Pumping lemma for unboundedness).**
*Let $\mathcal{S} \in \mathcal{UC}$, $(q,\nu_0),\ldots,(q,\nu_m)$ be a run of $\mathcal{S}$ of non-null length such that $(\nu_m)_{Var\setminus Unc} \in Sat(G_{Var\setminus Unc})$, and $\nu'_0 : Var \to \mathbb{Z}$ such that $\nu'_0 \in Sat(G_{Var})$ and $\nu_0$ and $\nu'_0$ agree on $Var \setminus Unc$. If $\lceil \frac{N_{\nu'_0}}{m\cdot(|Var|+1)} \rceil > \Delta$, then there is a run $(q,\nu'_0),\ldots,(q,\nu'_m)$ s.t.: $\nu'_m \in Sat(G_{Var})$, $N_{\nu'_m} \geq \lceil \frac{N_{\nu'_0}}{m\cdot(|Var|+1)} \rceil$, and for each $0 \leq i \leq m$, $\nu'_i$ and $\nu_i$ agree on $Var \setminus Unc$.*

**Lemma 22.** *Assume that $\mathcal{S} \in \mathcal{UC}$. Let $\nu_0, \nu : Var \to \mathbb{Z}$ be s.t. $\nu_0 \oplus \nu \in Sat(G_0 \bullet G)$ and $\nu \in Sat(G_{Var})$. Then, the following set is* infinite

$$\{N_{\nu'} \mid \nu_0 \oplus \nu' \in Sat(G_0 \bullet G),\ \nu' \in Sat(G_{Var}),\ \textit{and}\ \nu'_{Var\setminus Unc} = \nu_{Var\setminus Unc}\}$$

By Lemmata 20, 21, and 22, we deduce the following result.

**Lemma 23.** *Assume that $\mathcal{S} \in \mathcal{UC}$. Then, $(q_0,\nu_0) \in Unb_{\mathcal{S}}$* iff *there is a finite run $\pi$ of $\mathcal{S}$ starting from $(q_0,\nu_0)$ of the form*

$$\pi = (q_0,\nu_0)(q,\nu'_0)(q,\nu)\ldots(q,\nu')\ldots(q,\nu'')$$

*such that $\nu''_{(Var\setminus Unc)} = \nu'_{(Var\setminus Unc)}$, and the subrun $(q,\nu')\ldots(q,\nu'')$ has non-null length.*

*Proof.* For the right implication $\Rightarrow$, assume that $(q_0,\nu_0) \in Unb_{\mathcal{S}}$. Hence, the set of lengths of the finite runs from $(q_0,\nu_0)$ is infinite. Then, by Lemma 20, the result follows.
For the left implication $\Leftarrow$, assume that for a valuation $\nu_0$ over $Var$, there is a finite run $\pi$ of $\mathcal{S}$ starting from $(q_0,\nu_0)$ of the form

$$\pi = (q_0,\nu_0)(q,\nu'_0)(q,\nu)\ldots(q,\nu')\ldots(q,\nu'')$$

such that $\nu''_{(Var\setminus Unc)} = \nu'_{(Var\setminus Unc)}$, and the subrun $(q,\nu')\ldots(q,\nu'')$ has non-null length. Let us consider the prefix of $\pi$ of length 2 given by $(q_0,\nu_0)(q,\nu'_0)(q,\nu)$, and let $S_\nu = \{\overline{\nu} \mid \nu_0 \oplus \overline{\nu} \in Sat(G_0 \bullet G),\ \overline{\nu} \in Sat(G_{Var}),\ \text{and}\ \overline{\nu}_{Var\setminus Unc} = \nu_{Var\setminus Unc}\}$. Since $\nu_0 \oplus \nu \in Sat(G_0 \bullet G)$ and $\nu \in Sat(G_{Var})$, by Lemma 22, the set $S_\nu$ is infinite, and the set $Int(S_\nu) = \{N_{\overline{\nu}} \mid \overline{\nu} \in S_\nu\}$ is infinite as well. Let us consider the suffix of $\pi$, $(q,\nu)\ldots(q,\nu')\ldots(q,\nu'')$, where $\nu''_{(Var\setminus Unc)} = \nu'_{(Var\setminus Unc)}$. Let $\overline{\nu} \in S_\nu$ and $h \geq 1$. Since $\overline{\nu}_{Var\setminus Unc} = \nu_{Var\setminus Unc}$, Lemma 21 (applied repetitively) ensures that there is $n_h \in \mathbb{N}$ such that if $N_{\overline{\nu}} \geq n_h$, then there is a finite run $\pi_0\pi_1\ldots\pi_h$ from $(q,\overline{\nu})$ so that

– $\pi_0 = (q,\overline{\nu}) \dots (q,\nu^1)$ and for $1 \leq i \leq h$, $\pi_i$ is of the form $(q,\nu^i),\dots,(q,\nu^{i+1})$, has non-null length, and $\nu^i_{Var\setminus Unc} = \nu^{i+1}_{Var\setminus Unc} = \nu''_{(Var\setminus Unc)} = \nu'_{(Var\setminus Unc)}$.

Since $\nu_0 \oplus \overline{\nu} \in Sat(G_0 \bullet G)$, the run $\pi_0\pi_1 \dots \pi_h$ can be completed (by adding as prefix a run of length 2 from $(q_0,\nu_0)$ to $(q,\overline{\nu})$) into a run starting from $(q_0,\nu_0)$ (the whole run has length at least $h$). Since the set $Int(S_\nu)$ is infinite, for each $h \geq 1$, we can always choose $\overline{\nu} \in S_\nu$ in such a way that the above condition holds. Hence, $(q_0,\nu_0)$ is unbounded. This concludes the proof of the lemma. □

**Theorem 24.** *Assume that $\mathcal{S} \in \mathcal{UC}$. Then, $Unb_{\mathcal{S}}$ is* MG *representable and one can construct a* MG *representation of $Unb_{\mathcal{S}}$.*

*Proof.* Let $\mathcal{S} \in \mathcal{UC}$. Since, $\mathcal{UC} \subseteq \mathcal{TC}$, by Proposition 15, the set of unbounded states of $\mathcal{S}$ of the form $(q,\nu)$ is empty. Hence, it suffices to show that we can construct a finite set $\mathcal{G}_{q_0}$ of MG over $Var$ and $Const$ such that $\bigcup_{G\in\mathcal{G}_{q_0}} Sat(G) = \{\nu \mid (q_0,\nu) \in Unb_{\mathcal{S}}\}$. By Theorem 10, one can compute a *finite* set $\mathcal{P}$ of non-null finite paths of $\mathcal{S}$ from $q$ to $q$ such that for each non-null finite path $\wp'$ of $\mathcal{S}$ from $q$ to $q$, there is a path $\wp \in \mathcal{P}$ so that $\leadsto_{\wp'}$ implies $\leadsto_{\wp}$. Note that given $\wp \in \mathcal{P}$, the transitional MG $G_\wp$ (which characterizes the reachability relation $\leadsto_\wp$) has the form $\underbrace{G \bullet \dots \bullet G}_{k \text{ times}}$ for some $k \geq 1$. Let $G_=$ be the transitional MG corresponding to the GC given by $\bigwedge\limits_{x\in Var\setminus Unc} x' = x$. Then, $\mathcal{G}_{q_0}$ consists of the computable finite set of MG $G'$ over $Var$ and $Const$ such that $G' = (\overline{G''})_{Var}$, where $G'' = G_0 \bullet G_\wp \bullet (G_{\wp'} \bigotimes G_=)$ for some $\wp,\wp' \in \mathcal{P}$. Correctness of the construction easily follows from Propositions 4 and 5, and Lemma 23. □

Moreover, we show the following non-trivial result (a proof is in [7]).

**Theorem 25.** *If $\mathcal{S} \in \mathcal{UC}$, then $Unb^{q_0}_{\mathcal{S}} \neq \emptyset$.*

By Propositions 15 and 19, and Theorems 16, 17, 24, and 25, we obtain the following result, which provides a straightforward polynomial-time algorithm to check strong termination for simple GCS.

**Corollary 26.** *For a simple* GCS *$\mathcal{S}$, the following holds:*

– *If $\mathcal{S} \notin \mathcal{TC}$, then $Inf_{\mathcal{S}} = Unb_{\mathcal{S}}$ and $Inf^{q_0}_{\mathcal{S}} = Unb^{q_0}_{\mathcal{S}} \neq \emptyset$;*
– *If $\mathcal{S} \in \mathcal{UC}$ (hence, $\mathcal{S} \in \mathcal{TC}$), then $Inf_{\mathcal{S}} = \emptyset$ and $Unb^{q_0}_{\mathcal{S}} \neq \emptyset$;*
– *If $\mathcal{S} \in \mathcal{TC}$ and $\mathcal{S} \notin \mathcal{UC}$, then $Inf_{\mathcal{S}} = Unb_{\mathcal{S}} = \emptyset$.*

*Moreover, one can compute a* MG *representation of the sets $Inf_{\mathcal{S}}$ and $Unb_{\mathcal{S}}$.*

## 4 Strong Termination for Unrestricted GCS

Fix a GCS $\mathcal{S}$. First, we give a characterization of the set of unbounded states of $\mathcal{S}$. For a non-null finite path $\wp$ of $\mathcal{S}$ s.t. $s(\wp) = t(\wp)$ (i.e., $\wp$ is cyclic), $(\wp)^\omega$ denotes the infinite path $\wp\wp\dots$. A infinite path $\wp$ of $\mathcal{S}$ of the form $\wp = \wp'(\wp'')^\omega$

is said to be *ultimately periodic*. A state $s$ of $\mathcal{S}$ is *neatly unbounded* w.r.t. an infinite path $\wp$ of $\mathcal{S}$, if there is a sequence of finite runs $(\pi_n)_{n\in\mathbb{N}}$ of $\mathcal{S}$ from $s$ s.t. $\{|\pi_n| \mid n \in \mathbb{N}\}$ is infinite and for each $n$, $\pi_n$ is an instance of the prefix of $\wp$ of length $|\pi_n|$. By using Theorem 10 and Ramsey's Theorem (in its infinite version) [17], we show the following (a proof is in [7]).

**Theorem 27 (Characterization Theorem).** *Let $\mathcal{S}$ be a complete* GCS *and $\mathcal{P}_\mathcal{S}$ be the set of paths of $\mathcal{S}$ satisfying Theorem 10. Then, a state $s$ of $\mathcal{S}$ is unbounded* iff *$s$ is neatly unbounded w.r.t. a* ultimately periodic *path $\wp_0 \cdot (\wp)^\omega$ such that $\wp_0, \wp \in \mathcal{P}_\mathcal{S}$, $G_{\wp_0} \bullet G_\wp$ is satisfiable, $G_\wp$ is idempotent, and $G_{\wp_0}$ and $G_\wp$ are complete and normalized.*

Let $\mathcal{S}$ be a GCS. We denote by $\lfloor\mathcal{S}\rfloor_K$ the GCS obtained from $\mathcal{S}$ by replacing each edge $q \xrightarrow{G} q'$ of $\mathcal{S}$ with the edge $q \xrightarrow{\lfloor G\rfloor_K} q'$. Note that $\lfloor\mathcal{S}\rfloor_K$ is simple iff $\mathcal{S}$ is simple.

**Theorem 28.** *Let $\mathcal{S}$ be a* GCS*. Then, $Unb_\mathcal{S}$ is* MG *representable and one can construct a* MG *representation of $Unb_\mathcal{S}$. Moreover, given $q \in Q(\mathcal{S})$, checking whether $Unb^q_\mathcal{S} \neq \emptyset$ is in* PSPACE *and can be done in time $O(|E(\mathcal{S})| \cdot |Q(\mathcal{S})|^2 \cdot (K+2)^{(2|Var|+|Const|)^2})$.*

*Proof.* We assume that $\mathcal{S}$ is complete (the general case easily follows, and details are given in [7]). Let $\mathcal{P}_\mathcal{S}$ be the computable finite set of non-null finite paths of $\mathcal{S}$ satisfying Theorem 10, and let $\mathcal{F}$ be the finite set of *simple* GCS constructed as: $\mathcal{S}' \in \mathcal{F}$ iff $\mathcal{S}'$ is a GCS consisting of two edges of the form $(\natural, s(\wp_0)) \xrightarrow{G_{\wp_0}} t(\wp_0)$ and $s(\wp) \xrightarrow{G_\wp} t(\wp)$ such that $\wp_0, \wp \in \mathcal{P}_\mathcal{S}$ and $\mathcal{S}'$ is *simple* as well. By Corollary 26, for each $\mathcal{S}' \in \mathcal{F}$, one can compute a MG representation $\mathcal{G}_{\mathcal{S}',in(\mathcal{S}')}$ of $Unb^{(\natural,in(\mathcal{S}'))}_{\mathcal{S}'}$, where $(\natural, in(\mathcal{S}'))$ is the initial control point of $\mathcal{S}'$. Then, by Theorems 10 and 27, $\{\bigcup_{\{\mathcal{S}'\in\mathcal{F}|in(\mathcal{S}')=q\}} \mathcal{G}_{\mathcal{S}',in(\mathcal{S}')}\}_{q\in Q(\mathcal{S})}$ is a MG representation of $Unb_\mathcal{S}$. Thus, the first part of the theorem holds.

For the second part of the theorem, let $\mathcal{F}_K$ be the set of GCS $\mathcal{S}'$ such that $\mathcal{S}' = \lfloor\mathcal{S}''\rfloor_K$ for some $\mathcal{S}'' \in \mathcal{F}$. By Definitions 14 and 18, for a simple GCS $\mathcal{S}''$, $\mathcal{S}'' \in \mathcal{TC}$ (resp., $\mathcal{S}'' \in \mathcal{UC}$) iff $\lfloor\mathcal{S}''\rfloor_K \in \mathcal{TC}$ (resp., $\lfloor\mathcal{S}''\rfloor_K \in \mathcal{UC}$). Thus, by Corollary 26, we obtain that for $q \in Q(\mathcal{S})$: $Unb^q_\mathcal{S} \neq \emptyset$ iff there is $\mathcal{S}' \in \mathcal{F}_K$ with initial control point $(\natural, q)$ such that either $\mathcal{S}' \notin \mathcal{TC}$ or $\mathcal{S}' \in \mathcal{UC}$. Note that this last condition can be checked in time polynomial in the size of $\mathcal{S}'$. Now, the crucial observation is that $\mathcal{F}_K$ can be computed in exponential time since: (i) by Theorem 10, the set $\{(\lfloor G_\wp\rfloor_K, s(\wp), t(\wp)) \mid \wp \in \mathcal{P}_\mathcal{S}$ and $\lfloor G_\wp\rfloor_K$ is satisfiable$\}$ coincides with the set $\mathcal{G}^K_\mathcal{S} = \{(\lfloor G_\wp\rfloor_K, s(\wp), t(\wp)) \mid \wp$ is a non-null finite path of $\mathcal{S}$ and $\lfloor G_\wp\rfloor_K$ is satisfiable$\}$, (ii) by Theorem 9, the set $\mathcal{G}^K_\mathcal{S}$ can be computed in time $O(|E(\mathcal{S})| \cdot |Q(\mathcal{S})|^2 \cdot (K+2)^{(2|Var|+|Const|)^2})$. Hence, checking whether $Unb^q_\mathcal{S} \neq \emptyset$ can be done in time $O(|E(\mathcal{S})| \cdot |Q(\mathcal{S})|^2 \cdot (K+2)^{(2|Var|+|Const|)^2})$.

It remains to show that checking whether $Unb^q_\mathcal{S} \neq \emptyset$ can be done in polynomial space. We outline a NPSPACE algorithm. Since NPSPACE=PSPACE (by Savitch's theorem), the result follows. At each step, the nondeterministic algorithm *implicitly* guesses two non-null finite paths $\wp_0$ and $\wp$ of $\mathcal{S}$ such that $s(\wp_0) = q$, and compute the $\mathsf{GCS}$ $\mathcal{S}'$ having the edges $(\natural, s(\wp_0)) \overset{\lfloor G_{\wp_0} \rfloor_K}{\rightarrow} t(\wp_0)$ and $s(\wp) \overset{\lfloor G_{\wp} \rfloor_K}{\rightarrow} t(\wp)$. Note that by observation (i) above, $\mathcal{S}'$ is simple iff $\mathcal{S}' \in \mathcal{F}_K$. The algorithm keeps in memory *only* the $\mathsf{MG}$ $\lfloor G_{\wp_0} \rfloor_K$ and $\lfloor G_{\wp} \rfloor_K$ associated with the paths $\wp_0$ and $\wp$ implicitly generated so far, together with their source and target control points. If the current $\mathsf{MG}$ $\mathcal{S}'$ corresponds to a simple $\mathsf{MG}$ (hence, $\mathcal{S}' \in \mathcal{F}_K$) such that either $\mathcal{S}' \notin \mathcal{TC}$ or $\mathcal{S}' \in \mathcal{UC}$, then the algorithm terminates with success. Otherwise, the algorithm chooses two edges from control points $t(\wp_0)$ and $t(\wp)$, say $t(\wp_0) \overset{G_0}{\rightarrow} q_0$ and $t(\wp) \overset{G}{\rightarrow} q$, computes the $\mathsf{MG}$ $\lfloor \lfloor G_{\wp_0} \rfloor_K \bullet \lfloor G_0 \rfloor_K \rfloor_K$ and $\lfloor \lfloor G_{\wp} \rfloor_K \bullet \lfloor G \rfloor_K \rfloor_K$ associated with the currently guessed paths, and re-write the memory by replacing $\lfloor G_{\wp_0} \rfloor_K$ and $\lfloor G_{\wp} \rfloor_K$ with $\lfloor \lfloor G_{\wp_0} \rfloor_K \bullet \lfloor G_0 \rfloor_K \rfloor_K$ and $\lfloor \lfloor G_{\wp} \rfloor_K \bullet \lfloor G \rfloor_K \rfloor_K$, and $t(\wp_0)$ and $t(\wp)$ with $q_0$ and $q$, and the procedure is repeated. □

**Corollary 29.** *The strong termination problem and the strong termination problem w.r.t. a designated state are both* PSPACE*-complete.*

*Proof.* By Theorem 28, strong termination is in PSPACE, and checking whether $Unb^q_{\mathcal{S}} = \emptyset$ for a given $\mathsf{GCS}$ $\mathcal{S}$ and $q \in Q(\mathcal{S})$, is in PSPACE too. By an easy linear-time reduction to this last problem, membership in PSPACE for strong termination w.r.t. a designated state follows as well (for details see [7]). PSPACE-hardness directly follows from PSPACE-hardness of termination for Boolean Programs [15] and the fact that $\mathsf{GCS}$ subsume Boolean Programs (note that for Boolean Programs, which are finitely-branching, strong termination corresponds to termination). □

## References

1. Abdulla, P.A., Delzanno, G.: On the coverability problem for constrained multiset rewriting. In: Proc. 5th AVIS (2006)
2. Abdulla, P.A., Delzanno, G., Rezine, A.: Approximated parameterized verification of infinite-state processes with global conditions. Formal Methods in System Design 34(2), 126–156 (2009)
3. Albert, E., Arenas, P., Genaim, S., Puebla, G.: Automatic Inference of Upper Bounds for Recurrence Relations in Cost Analysis. In: Alpuente, M., Vidal, G. (eds.) SAS 2008. LNCS, vol. 5079, pp. 221–237. Springer, Heidelberg (2008)
4. Ben-Amram, A.: Size-change termination, monotonicity constraints and ranking functions. Logical Methods in Computer Science 6(3) (2010)
5. Ben-Amram, A., Vainer, M.: Complexity Analysis of Size-Change Terminating Programs. In: Second Workshop on Developments in Implicit Computational Complexity (2011)
6. Bozga, M., Gîrlea, C., Iosif, R.: Iterating Octagons. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 337–351. Springer, Heidelberg (2009)
7. Bozzelli, L.: Strong termination for gap-order constraint abstractions of counter systems. Technical report (2011), http://clip.dia.fi.upm.es/~lbozzelli
8. Bozzelli, L., Pinchinat, S.: Verification of gap-order constraint abstractions of counter systems. In: Proc. 13th VMCAI, Springer, Heidelberg (2012)
9. Cerans, K.: Deciding Properties of Integral Relational Automata. In: Shamir, E., Abiteboul, S. (eds.) ICALP 1994. LNCS, vol. 820, pp. 35–46. Springer, Heidelberg (1994)

10. Comon, H., Cortier, V.: Flatness Is Not a Weakness. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 262–276. Springer, Heidelberg (2000)
11. Comon, H., Jurski, Y.: Multiple Counters Automata, Safety Analysis and Presburger Arithmetic. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 268–279. Springer, Heidelberg (1998)
12. Demri, S., D'Souza, D.: An automata-theoretic approach to constraint LTL. Information and Computation 205(3), 380–415 (2007)
13. Fribourg, L., Richardson, J.: Symbolic Verification with Gap-Order Constraints. In: Gallagher, J.P. (ed.) LOPSTR 1996. LNCS, vol. 1207, pp. 20–37. Springer, Heidelberg (1997)
14. Ibarra, O.: Reversal-bounded multicounter machines and their decision problems. Journal of ACM 25(1), 116–133 (1978)
15. Jonson, N.D.: Computability and Complexity from a Programming Perspective. Foundations of Computing Series. MIT Press (1997)
16. Peterson, J.L.: Petri Net Theory and the Modelling of Systems. Prentice-Hall (1981)
17. Ramsey, F.: On a problem of formal logic. Proceedings of the London Mathematical Society 30, 264–286 (1930)
18. Revesz, P.Z.: A Closed-Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. Theoretical Computer Science 116(1-2), 117–149 (1993)

# Covering Space in the Besicovitch Topology*

Julien Cervelle

LACL, Université Paris-Est Créteil
94010 Créteil cedex, France
julien.cervelle@univ-paris-est.fr

**Abstract.** This paper studies how one can spread points in the Besicovitch space in order to keep them far one from another. We first study the general case and then what happens if the chosen points are all regular Toeplitz configurations or all quasiperiodic configurations.

**Keywords:** Hamming distance, Besicovitch distance, dynamical systems, Toeplitz sequences.

## 1 Introduction

In compact spaces, the more you have points in a set, the shorter the distance between the closest points of the set. More precisely, for any $\varepsilon$, there is an integer $N$ such that any set of cardinal at least $N$ contains two points whose relative distance is less than $\varepsilon$. This is easily proved covering the compact space with open balls of diameter $\varepsilon$ and selecting a finite sub-covering and choosing $N$ as its size plus one. If one has $N$ points, using pigeon hole lemma, there are two points in the same ball.

This is not the case in non-compact spaces. For instance, it is clear that $\mathbb{N} \subset \mathbb{R}$ is an infinite set of reals which are all at distance greater or equal to 1 from all other members.

This feature has some direct consequences on code theory. For error-correcting (resp. detecting) codes, the valid representations of information must be at the center of open balls of some fixed radius which do not overlap (resp. which do not contain another valid representation). The radius depends on the number of error to correct (resp. detect). For classical code theory on finite words, the Hamming distance on words is often considered.

In this paper, we study the space $\{0,1\}^{\mathbb{N}}$ of uni-infinite words on $\{0,1\}$ or *configurations* endowed with the Besicovitch topology. Configurations are often called sequences or streams. For more investigations about configurations, see for instance [1]. The Besicovitch distance is used, among others, in the domain of symbolic dynamical systems, and particularly cellular automata. Considering the phase space $\{0,1\}^{\mathbb{N}}$, the classical product topology, called Cantor topology, has counter-intuitive properties and the Besicovitch topology was proposed as

* This work has been supported by the ANR Blanc "Projet EQINOCS" (ANR-11-BS02-004).

a less biased alternative for cellular automata dynamical behavior study (see [4,3,6]). Moreover, looking at its definition, one can note that it is a sort of extension of the Hamming distance to infinite words.

The space $\{0,1\}^{\mathbb{N}}$ is not compact and many of the proofs for result with Cantor topology can't be done in the Besicovitch topology. For instance Hedlund's theorem which states that cellular automata are the continuous shift-invariant map on $\{0,1\}^{\mathbb{N}}$ is based on compactness and open cover extraction (see [7]).

We want to evaluate how much non-compact the Besicovitch space is and, if possible, prove a kind of weak compactness. We do this by studying how points get closer as you add points. Formally if $\mathcal{S}$ is a set with at least $N$ members we want to find the maximum distance between the closest members of $\mathcal{S}$ and see how this behaves when $N$ tends to infinity.

First, we state a negative result giving an uncountable set of configurations such that any two members are at distance 1, the maximum for the Besicovitch distance. Hence, there is no chance to prove a kind of weak compactness for the Besicovitch space. Next, we try to see if the negative result is still true restricting ourselves to some natural subsets of configurations, *Toeplitz configurations* and *quasiperiodic configurations.*

Toeplitz configuration are a dense positively invariant set in $\{0,1\}^{\mathbb{N}}$. It has been studied in [2] and proposed as a good test set since to prove some properties it is enough to prove them only on Toeplitz sequences. They play a role similar to periodic sequences in Cantor topology. Quasiperiodic configurations are a natural candidate between the general case and Toeplitz configurations and play a special role in the field of tilings, often considered as the static version of cellular automata (see for instance [5]).

We first prove that the negative result still holds on quasiperiodic configurations. However, we can prove a non-trivial bound for a natural subset of Toeplitz configurations: *regular* Toeplitz configurations. In order to perform this last study, we first consider finite words on $\{0,1\}$ of a given length since the Besicovitch distance definition is expressed in terms of the Hamming distance between the prefixes. Then we extend the result on the Hamming distance to $\{0,1\}^{\mathbb{N}}$ proving that the distance between closest members of a set tends to one half when the cardinal of the set increases and that this bound is tight.

The paper is organized as follows. In the next section, we give definitions about the Besicovitch distance, quasiperiodic and Toeplitz configurations. In Section 3 and 4 we state the negative results about the Besicovitch distance for the general and the quasiperiodic cases. In Section 5, we study the restriction to regular Toeplitz configurations.

## 2 Definitions and Tools

In this section, we introduce the space we study and a few notions.

*Configurations.* We call *configurations* uni-infinite words on $\{0,1\}$. If $x$ is a configuration, as for words, we note $x_i$ the $i$th letter (the first one has index 0). We note $x_{\rightarrow n}$ the prefix on length $n$ of $x$ i.e. $x_{\rightarrow n} = x_0 \ldots x_{n-1}$. A word $u$ is a *factor* of the configuration (or the word) $x$ if there is an integer $k$ such that for all $i$ where $0 \leqslant i < |u|$, $u_i = x_{k+i}$. We note it $u \sqsubset x$ or $u \sqsubset_n x$ if $u$ is of length $n$.

*The Besicovitch topology.* The Besicovitch topology measures the rate of differences between two configurations. Its formal definition is given by the pseudo-distance $d_B$ defined by

$$d_B(x, y) = \limsup_{n\to\infty} \frac{d_H(x_{\to n}, y_{\to n})}{n}$$

where $d_H$, the Hamming distance, is such that $d_H(x, y) = |\{n,\ x_n \neq y_n\}|$.

It is only a pseudo-distance since two configurations with finitely or logarithmically many differences are at distance zero. Taking the quotient of $\{0, 1\}^{\mathbb{N}}$ w.r.t. the equivalence relation $x \sim y \Leftrightarrow d(x, y) = 0$, we obtain a distance. For more information about the Besicovitch topology, see [4,3].

*Quasiperiodic configurations.* The configuration $x$ is *quasiperiodic* if

$$\forall n, \exists N, \forall u, u \sqsubset_n x \Rightarrow \forall w, w \sqsubset_N x \Rightarrow u \sqsubset w,$$

that is if for all integers $n$ there exists an integer $N$ such that all factor of length $n$ of $x$ can be found in any factor of length $N$.

## 2.1 Toeplitz Configurations and Their Construction

A configuration $x$ is *Toeplitz* if for all positions $i \in \mathbb{N}$, there exists a period $p$ such that $\forall k \in \mathbb{Z}$ such that $pk + i \geqslant 0$, $x_{pk+i} = x_i$.

In order to build Toeplitz configurations, one can use a simple algorithm. It assigns letters to cells of the configuration step by step. Initially, all cells are unassigned. At each step, a cell is assigned a value, and this assignment is repeated periodically along the configuration. With this algorithm, one can build all Toeplitz configurations.

Formally, a Toeplitz configuration $x$ is totally (but not uniquely) defined by a finite or infinite sequence of couple $(v_i, p_i)_{0\leqslant i<L}$ $(L \in \mathbb{N} \cup \{\infty\})$ where the cells of $x$ are filled with values $v_i$ periodically with period $p_i$. At step 0, $x_0$ and all cells of index $kp_0$ for $k \in \mathbb{N}$ are assigned to $v_0$. At step 1, choose the smallest index of an unassigned cell $j_1$ (which must be 1 unless $p_0 = 1$ in which case there is no step 1). All cells of index $j_1 + kp_1$ are assigned to $v_1$. We continue to assign values to cells periodically, always starting from the unassigned cell of smallest index: at step $i$, for $j_i$ the smallest index of an unassigned cell, all cells of index $j_i + kp_i$ are assigned to $v_i$. The beginning of the process is illustrated Figure 1.

This process must verify two conditions:

- the periods $p_i$ must be chosen so that a cell is never assigned twice. This implies a rather complicated relation on periods;
- each cell is eventually assigned at some step.

If $v$ and $p$ are two such sequences, we note $\mathcal{A}(v, p)$ the Toeplitz sequence the algorithm outputs. If the sequence $(v_i, p_i)_{i\in\mathbb{N}}$ is finite, then the configuration is periodic with period given by the least common multiple of the $p_i$.

$$\begin{array}{r@{\,:\,}l}
p_0 = 3 & \mathbf{v_0}\ .\ .\ \mathbf{v_0}\ .\ .\ \mathbf{v_0}\ .\ .\ \mathbf{v_0}\ .\ .\ \mathbf{v_0}\ .\ .\ \mathbf{v_0}\ .\ .\ \mathbf{v_0}\ .\ .\ \mathbf{v_0}\ .\ . \\
p_1 = 6 & v_0\,\mathbf{v_1}\ .\ v_0\ .\ .\ v_0\,\mathbf{v_1}\ .\ v_0\ .\ .\ v_0\,\mathbf{v_1}\ .\ v_0\ .\ .\ v_0\,\mathbf{v_1}\ .\ v_0\ .\ . \\
p_2 = 6 & v_0\,v_1\,\mathbf{v_2}\,v_0\ .\ .\ v_0\,v_1\,\mathbf{v_2}\,v_0\ .\ .\ v_0\,v_1\,\mathbf{v_2}\,v_0\ .\ .\ v_0\,v_1\,\mathbf{v_2}\,v_0\ .\ . \\
p_3 = 12 & v_0\,v_1\,v_2\,v_0\,\mathbf{v_3}\ .\ v_0\,v_1\,v_2\,v_0\ .\ .\ v_0\,v_1\,v_2\,v_0\,\mathbf{v_3}\ .\ v_0\,v_1\,v_2\,v_0\ .\ . \\
p_4 = 6 & v_0\,v_1\,v_2\,v_0\,v_3\,\mathbf{v_4}\,v_0\,v_1\,v_2\,v_0\ .\ \mathbf{v_4}\,v_0\,v_1\,v_2\,v_0\,v_3\,\mathbf{v_4}\,v_0\,v_1\,v_2\,v_0\ .\ \mathbf{v_4} \\
\multicolumn{2}{c}{\vdots} \\
\multicolumn{2}{r}{v_0\,v_1\,v_2\,v_0\,v_3\,v_4\,v_0\,v_1\,v_2\,v_0\,v_5\,v_4\,v_0\,v_1\,v_2\,v_0\,v_3\,v_4\,v_0\,v_1\,v_2\,v_0\,v_6\,v_4}
\end{array}$$

**Fig. 1.** Sample Toeplitz configuration construction

## 3 The Result for the General Case

Our first result for the Besicovitch topology is negative. It states that there exists an uncountable set of configurations such that each member is at distance 1 (the maximum for the Besicovitch distance) from all other members.

**Lemma 1.** *There exists an uncountable set $\mathcal{D}$ of infinite words such that for any two of those words, they differ at infinitely many positions.*

*Proof.* Let $u$ be a sequence of $\{0,1\}^{\mathbb{N}}$. Let $w^u$ be the Toeplitz configurations built on the sequence $(u_i, 2^i)_{i\in\mathbb{N}}$.

Informally, the sequence is built as follows: the half of the configuration is assigned to $u_0$; the half of the remaining cells is assigned to $u_1$; the half of the remaining cells is assigned to $u_2$ and so on. The beginning of the sequence is (letters are raised according to their index for better readability)

$$u_0{}^{u_1}u_0{}^{u_2}u_0{}^{u_1}u_0{}^{u_3}u_0{}^{u_1}u_0{}^{u_2}u_0{}^{u_1}u_0{}^{u_4}u_0{}^{u_1}u_0{}^{u_2}u_0{}^{u_1}u_0{}^{u_3}u_0{}^{u_1}u_0{}^{u_2}u_0{}^{u_1}u_0$$

If $u$ and $v$ are two distinct binary sequences and $l$ is such that $u_l \neq v_l$, then at all positions $i$ such that $i \equiv 2^l - 1 \mod 2^{l+1}$ (hence infinitely many), the sequences $w^u$ and $w^v$ are distinct. The set $\mathcal{D} = \{w^u,\, u \in \{0,1\}^{\mathbb{N}}\}$ proves the lemma. $\square$

The negative result is as follows.

**Proposition 2.** *There exists an uncountable set of configurations $\mathcal{Y}$ such that for any $x$ and $y$ in $\mathcal{Y}$, $d_B(x,y) = 1$.*

*Proof.* In this proof, we note $\mathfrak{2}^i = 2^{2^i}$. Let $w$ be a sequence of $\mathcal{D}$ of Lemma [1]. Define the configuration $s^w$ by $s^w_i = w_j$ for $\mathfrak{2}^j - 2 \leqslant i < \mathfrak{2}^{j+1} - 2$. It is the concatenation of blocs of $\mathfrak{2}^{j+1} - \mathfrak{2}^j$ times the letter $w^j$.

Let $w$ and $w'$ be two distinct sequences of $\mathcal{D}$. There is an infinite set $J$ of positions which $w$ and $w'$ differ at. By definition of $s$, for all $j \in J$, since $\mathfrak{2}^i = o(\mathfrak{2}^{i+1})$, one has

$$\frac{d_H(s^w_{\to \mathfrak{2}^{j+1}-2}, s^{w'}_{\to \mathfrak{2}^{j+1}-2})}{\mathfrak{2}^{j+1} - 2} \geqslant \frac{(\mathfrak{2}^{j+1} - 2) - (\mathfrak{2}^j - 2)}{\mathfrak{2}^{j+1} - 2} \sim 1 \ .$$

Hence, considering that the limit superior is greater than the limit superior of any subsequence, $d_B(s^w, s^{w'}) = 1$. The set $\mathcal{Y}$ is defined by $\mathcal{Y} = \{s^w,\, w \in \mathcal{D}\}$. $\square$

The following corollary can be deduced:

**Corollary 3.** *Let $\mathcal{R}$ be a dense set of configurations according to Besicovitch topology. Then for all $\varepsilon > 0$, there is an infinite set $\mathcal{E}^{\varepsilon}_{\mathcal{R}}$ of configurations in $\mathcal{R}$ such that for any $x$ and $y$ in $\mathcal{E}^{\varepsilon}_{\mathcal{R}}$, $d_B(x,y) > 1-\varepsilon$. If $\mathcal{R}$ is uncountable, $\mathcal{E}^{\varepsilon}_{\mathcal{R}}$ can be chosen uncountable.*

*Proof.* Let $\mathcal{Y}$ be the set of proposition 2. As $\mathcal{R}$ is dense, for all $y \in \mathcal{Y}$, there is a point $r_y \in \mathcal{R}$ such that $d_B(y, r_y) < \frac{\varepsilon}{2}$. Denote $\mathcal{E}^{\varepsilon}_{\mathcal{R}} = \{r_y,\, y \in \mathcal{Y}\}$. Provided $\varepsilon$ is less than 1, $\mathcal{Y}$ and $\mathcal{E}^{\varepsilon}_{\mathcal{R}}$ have same cardinal. Moreover, using triangular inequality, one has that for all $x$ and $y$ in $\mathcal{Y}$, $d_B(r_x, r_y) > d_B(x,y) - d_B(r_x, x) - d_B(r_y, y) > 1 - \varepsilon$. □

## 4 Quasiperiodic Case Study

In this section, we show that the negative result of the previous section still holds for quasiperiodic configurations.

**Theorem 4.** *There exists a non countable set $\mathcal{Q}$ of quasiperiodic configurations such that for any two $x$ and $y$ in $\mathcal{Q}$, $d_B(x,y) = 1$.*

*Proof.* In this proof, we note $\overline{u}$ the word $u$ where all ones are replaced by zeros and zeros by ones. Let $u \in \{0,1\}^{\mathbb{N}}$. Let $q_u$ defined as the limit of the substitution process: $a_0 = 0$; $a_{n+1} = a_n^2\ \overline{a_n}^{\ 2} a_n\ \overline{a_n}^{\ n}$ if $u_n = 0$ and $a_{n+1} = a_n^2\ \overline{a_n}^{\ 2} a_n a_n^n$ if $u_n = 1$.

Let us first prove that any sequence $q_u$ is quasiperiodic. Let $w$ be a pattern of size $\ell$ of $q_u$. Let $n$ be such that $\ell < |a_n|$. By construction, $w$ is a factor of either $a_n a_n$, $a_n\ \overline{a_n}$, $\overline{a_n a_n}$ or $\overline{a_n}\ a_n$. By construction, in each window of size $2|a_{n+1}|$ either $a_{n+1}$ of $\overline{a_{n+1}}$ occurs. Both contains each of the words $a_n a_n$, $a_n\ \overline{a_n}$, $\overline{a_n a_n}$ or $\overline{a_n}\ a_n$ and so each window of size $2|a_{n+1}|$ contains $w$.

Define $\mathcal{Q} = \{q_u, u \in \mathcal{D}\}$, where $\mathcal{D}$ is defined in lemma 1. Let $u, v \in \mathcal{D}$. Let $I$ be the infinite set of positions $i$ such that $u_i \neq v_i$. We have that

$$d_B(q_u, q_v) \geqslant \lim_{n \in I} \frac{d_H(a_n^2\ \overline{a_n}^{\ 2} a_n\ \overline{a_n}^{\ n}, a_n^2\ \overline{a_n}^{\ 2} a_n a_n^n)}{|a_{n+1}|} \geqslant \lim_{n \in I} \frac{n|a_n|}{(5+n)|a_n|} = 1 \ .$$

□

## 5 Regular Toeplitz Case Study

In this section, we deal with regular Toeplitz sequences that will be defined in Section 5.2.

### 5.1 Finite Words and the Hamming Distance

In this section, we study the space $\{0,1\}^{\ell}$ of words on alphabet $\{0,1\}$ of a given length $\ell$, endowed with the Hamming distance $d_H$. For a set of cardinal $N$, we want to find a (tight) bound $M$ such that $\forall \mathcal{S} \subset \{0,1\}^{\ell}$ s.t. $|\mathcal{S}| = N$, $\min\{d_H(x,y) |\, x, y \in \mathcal{S}\} \leqslant M$.

**Some Technical Lemmas.** We give two combinatorial lemmas.

We consider a set $\mathcal{S} = \{w^0, \dots, w^{N-1}\}$ of $N$ words or length $\ell$. For $i \in \{0, \dots, \ell-1\}$, we define $\mathcal{P}_i = \{\{p,q\} | w_i^p = w_i^q\}$. For a couple $\{p,q\}$, $d_h(w^p, w^q) = \ell - m$, where $m = |\{i, \{p,q\} \in \mathcal{P}_i\}|$. The minimum Hamming distance between words of $\mathcal{S}$ is linked to the couple which maximizes $m$.

In order to simplify proofs and results, we use the following function,

$$g(n) = \binom{\lfloor \frac{n}{2} \rfloor}{2} + \binom{\lceil \frac{n}{2} \rceil}{2} \quad \text{for } n \geqslant 4 \ .$$

The first lemma considers a vector $V = (V_0, \dots, V_{N-1})$ and bounds the number of couples $\{p,q\}$ such that $V_p = V_q$.

**Lemma 5.** *Let $V$ be a vector of letters from the alphabet $\{0,1\}$ of size $N \geqslant 4$. We have that $\left|\{\{p,q\},\, V_p = V_q\}\right| \geqslant g(N)$.*

*Proof.* Let $z$ be the number of $i$ such that $V_i = 0$. There are $\binom{z}{2}$ couples $\{p,q\}$ such that $V_p = V_q = 0$ and $\binom{N-z}{2}$ couples $\{p,q\}$ such that $V_p = V_q = 1$. Hence $|\{\{p,q\},\, V_p = V_q\}| = \binom{N-z}{2} + \binom{z}{2}$. The proof is achieved by a straightforward recurrence, proving that for all $n \geqslant 4$ and for all $z$ such that $0 \leqslant z \leqslant \frac{n}{2}$, $\binom{n-z}{2} + \binom{z}{2} \geqslant g(n)$. □

The next lemma is an upper bound for $M$, the number of $\mathcal{P}_i$ which contains the couple that occurs the most often.

**Lemma 6.** *Let $\mathcal{I}$ be a set of finite cardinal $c$. Let $(\mathcal{P}_i)_{0 \leqslant i < \ell}$ be a sequence of subsets of $\mathcal{I}$ $|\mathcal{P}_i| \geqslant k$. One has $\max_{s \in \mathcal{I}} |\{i,\, s \in \mathcal{P}_i\}| \geqslant \frac{\ell k}{c}$.*

*Proof.* Let $M = \max_{s \in \mathcal{I}} |\{i,\, s \in \mathcal{P}_i\}|$. One has

$$\ell k \leqslant \sum_{i=0}^{\ell-1} |\mathcal{P}_i| = \sum_{s \in \mathcal{I}} \sum_{i=0}^{\ell-1} |\mathcal{P}_i \cap \{s\}| = \sum_{s \in \mathcal{I}} |\{i,\, s \in \mathcal{P}_i\}| \leqslant Mc \ . \square$$

**Bound for the Hamming Distance.** Let $f$ be defined by

$$f(n) = \frac{\lceil \frac{n}{2} \rceil}{2 \lceil \frac{n}{2} \rceil - 1} \ .$$

Note that $f(n) = 1 - \frac{g(n)}{\binom{n}{2}}$, $\lim_{n \to \infty} f(n) = \frac{1}{2}$ and $f(2n) = f(2n-1)$.

**Proposition 7.** *Let $\mathcal{S}$ be a set of $N$ words of length $\ell$ from alphabet $\{0,1\}$. Then $\min_{x,y \in \mathcal{S}} d_H(x,y) \leqslant f(N)\ell$.*

The result states that for a set of cardinal $N$ of words of fixed length, one can always find two words whose ratio of differences relatively to their length is less than $f(N)$, which tends to one half when $N$ is large enough.

*Proof.* Let $\mathcal{S} = \{w^0, \ldots, w^{N-1}\}$. For all $i \in \{0, \ldots, \ell-1\}$, let $\mathcal{P}_i = \big\{\{p,q\},\ w_i^p = w_i^q\big\}$ and $M = \max_c |\{i, c \in \mathcal{P}_i\}|$.

Let $w^p$ and $w^q$ be two words. One has $\forall i, \{p,q\} \in \mathcal{P}_i \Rightarrow w_i^p = w_i^q$. Hence $d_H(w^p, w^q) = \ell - |\{i,\ \{p,q\} \in \mathcal{P}_i\}| \geqslant \ell - M$, and equality holds for the couple $\{p,q\}$ such that $|\{i, \{p,q\} \in \mathcal{P}_i\}| = M$. We conclude that $\ell - M$ is the distance between the closest words $w_p$ and $w_q$. By Lemma 5 we have that $|\mathcal{P}_i| \geqslant g(N)$. Using this inequality in Lemma 6 one finds that $M = \max_c |\{i, c \in \mathcal{P}_i\}| \geqslant \frac{\ell g(N)}{\binom{N}{2}} = \ell(1 - f(N))$ and that $d_H(w_p, w_q) = \ell - M \leqslant \ell - \ell(1 - f(N)) = f(N)\ell$. □

**Bound Tightness.** These kind of bounds have already been studied in the field of code theory in [8]. This paper studies equidistant codes and gives a result which states the tightness of bound of Proposition 7. For all $N$, it gives a set $\mathcal{W}_N$ of $N$ words of some fixed length $\ell$ at Hamming distance $f(N)\ell$ one another. Words of $\mathcal{W}_N$ for $N = 4, 6, 8$ are given in Figure 2, where $\mathcal{C}$ is ordered lexicographically.

$$\begin{aligned} w^0 &= \mathtt{111} \\ w^1 &= \mathtt{100} \\ w^2 &= \mathtt{010} \\ w^3 &= \mathtt{001} \end{aligned}$$

*(a)* $2n = 4$

$$\begin{aligned} w^0 &= \mathtt{1111111111} \\ w^1 &= \mathtt{1111000000} \\ w^2 &= \mathtt{1000111000} \\ w^3 &= \mathtt{0100100110} \\ w^4 &= \mathtt{0010010101} \\ w^5 &= \mathtt{0001001011} \end{aligned}$$

*(b)* $2n = 6$

$$\begin{aligned} w^0 &= \mathtt{11111111111111111111111111111111111} \\ w^1 &= \mathtt{11111111111111100000000000000000000} \\ w^2 &= \mathtt{11111000000000011111111110000000000} \\ w^3 &= \mathtt{10000111100000011110000001111110000} \\ w^4 &= \mathtt{01000100011100010001110001110001110} \\ w^5 &= \mathtt{00100010010011001001001101001101101} \\ w^6 &= \mathtt{00010001001010100100101010101011011} \\ w^7 &= \mathtt{00001000100101100010010110010110111} \end{aligned}$$

*(c)* $2n = 8$

```
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111110000000000000000000000000000000000000000000000000000000000000000000000
111111111111111111111000000000000000000000000000000000001111111111111111111111111111111111100000000000000000000000000000000000
111111000000000000000111111111111111000000000000000000001111111111111110000000000000000000011111111111111111111000000000000000
100000111110000000000111110000000000111111111100000000001111100000000001111111111000000000011111111110000000000111111111100000
010000100001111000000100001111000000111100000011111100001000011110000001111000000111111000011110000001111110000111111000011110
001000010001000111000010001000111000100011100011100011100100010001110001000111000111000111010001110001110001110111000111011101
000100001000100100110001000100100110010010011010011011010010001001001100100100110100110110101001001101001101101100110110111011
000010000100010010101000100010010101001001010101010110110001000100101010010010101010101101101001010101010110111010110111011111
000001000010001001011000010001001011000100101100101101110000100010010110001001011001011011100010010110010110111001011011101111
```

*(d)* $2n = 10$

**Fig. 2.** Words for small $n$

## 5.2 Result for Regular Toeplitz Configurations

The obstacle to the extension of the result on finite words to infinite words comes from the limit superior in the definition of $d_B$. If we restrict our consideration to a class of configurations on which the limit of the Hamming distances between prefixes necessarily exists and is therefore the limit superior required in the Besicovitch distance, then the extension is possible.

In this section, we prove that the limit exists for subset of Toeplitz configurations called *regular Toeplitz configurations*. From corollary 3, as Toeplitz configurations are dense, there is no hope to extend it to the whole class.

During the building of a Toeplitz confiugration using the algorithm given in Section 2.1, though all cells are to be assigned, there are no insurance that the proportion of assigned cells tends to 1 while going through steps. We are only sure that each cell is eventually defined at some step. However, as Besicovitch topology relies on proportions, we say that a Toeplitz configuration $x$ is *regular* if there are sequences $p$ and $v$ such that $x = \mathcal{A}(v,p)$ and:

$$\sum_{0\leqslant j<L} \frac{1}{p_j} = 1 \ . \tag{$*$}$$

The set of regular Toeplitz sequences has continuous cardinal since the sequence $p_i = 2^{i+1}$ generates a regular Toeplitz sequence for any $v$.

We prove that when the cardinal of a set of regular Toeplitz configurations increases, the closest members tend to be at distance one half from each-other.

First we have to state that the limit exists.

**Lemma 8.** *If $x$ and $y$ are regular Toeplitz configurations, the sequence $u_n = \frac{d_H(x_{\to n},y_{\to n})}{n}$ is a Cauchy sequence.*

*Proof.* Let $(v_i,p_i)_{0\leqslant i<L}$ be a sequence defining $x$ and $(v'_i,p'_i)_{0\leqslant i<L'}$ be a sequence defining $y$ using the algorithm given in Section 2.1.

Let $\varepsilon > 0$. Using Equation $(*)$ there is an integer $N$ such that

$$\sum_{j=0}^{N} \frac{1}{p_j} \geqslant 1-\frac{\varepsilon}{8} \quad\text{and}\quad \sum_{j=0}^{N} \frac{1}{p'_j} \geqslant 1-\frac{\varepsilon}{8} \ .$$

Then at step $N$ in building $x$ and $y$, a proportion $1-\frac{\varepsilon}{8}$ cells has been assigned. Moreover, all the letters in these assigned cells repeat with a period $P = \operatorname{lcm}\{p_i, p'_i\, i \in \{0,\dots,N\}\}$. Hence, there are two words $M_x$ and $M_y$ of length $P$ on alphabet $\{0,1,\#\}$, where $x$ [resp. $y$] is the repetition of $M_x$ [resp. $M_y$] where # can be replaced by 0 or 1. As a ratio of $1-\frac{\varepsilon}{8}$ positions are already set, $M_x$ and $M_y$ each has at most $P\frac{\varepsilon}{8}$ occurrences of #.

For instance, if $N = 2$, $v_0 = 0$, $v'_0 = 1$, $v_1 = 1$, $v'_1 = 0$, $p_0 = p'_0 = 2$, $p_1 = 4$ and $p'_1 = 6$ then

$$x = 010?010?010?010?010?010?010?010?010?010?010?010?\dots$$
$$y = 101?1?101?1?101?1?101?1?101?1?101?1?101?1?101?1?\dots$$

where cells marked with ? is set by other values of $v$, $v'$, $p$ and $p'$. The repeated words are $M_x = 010\#010\#010\#$ and $M_y = 101\#1\#101\#1\#$.

If $d = d_H(M_x,M_y)$, for $n > P$ we have $d\left\lfloor\frac{n}{P}\right\rfloor \leqslant d_H(x_{\to n},y_{\to n}) \leqslant \left(d+\frac{P\varepsilon}{8}\right)\left\lceil\frac{n}{P}\right\rceil$ and hence $\frac{d\left(\frac{n}{P}-1\right)}{n} \leqslant \frac{d_H(x_{\to n},y_{\to n})}{n} \leqslant \frac{\left(d+\frac{P\varepsilon}{8}\right)\left(\frac{n}{P}+1\right)}{n}$.

We conclude that $\left|u_n - \frac{d}{P}\right| = \left|\frac{d_H(x_{\to n},y_{\to n})}{n} - \frac{d}{P}\right| \leqslant \frac{\varepsilon}{8P} + \frac{d}{n} + \frac{P\varepsilon}{8n} \leqslant \frac{\varepsilon}{4} + \frac{d}{n}$.

For $n \geqslant \max\left(P, \frac{4d}{\varepsilon}\right) = N'$, one has $\left|u_n - \frac{d}{P}\right| \leqslant \frac{\varepsilon}{2}$. Using the triangular inequality, one has, for all $n$ and $m$ greater than $N'$, $|u_n - u_m| \leqslant \left|u_n - \frac{d}{P}\right| + \left|\frac{d}{P} - u_m\right| \leqslant \varepsilon$.

We conclude that $u_n$ is a Cauchy sequence. □

Now we can state the proposition for regular Toeplitz configurations.

**Proposition 9.** *If $S$ is a set of $N$ regular Toeplitz configurations,*

$$\min_{x,y\in\mathcal{S}} d_B(x,y) \leqslant f(N)$$

*where $f$ is the function defined in Section 5.1.*

*Proof.* Using Lemma 8 and Proposition 7, we have:

$$\begin{aligned}\min_{x,y\in\mathcal{S}} d_B(x,y) &= \min_{x,y\in\mathcal{S}} \limsup_{n\to\infty} \frac{d_H(x_{\to n}, y_{\to n})}{n} \\ &= \min_{x,y\in\mathcal{S}} \lim_{n\to\infty} \frac{d_H(x_{\to n}, y_{\to n})}{n} \leqslant f(N) \ .\end{aligned}$$

□

The following corollary applies the above result to an infinite set of Toeplitz configurations.

**Corollary 10.** *If $\mathcal{S}$ is an infinite set of regular Toeplitz configurations, then for all $\varepsilon > 0$, one can find infinitely many pairs $(x,y)$ with $x$ and $y$ in $\mathcal{S}$ such that*

$$d_B(x,y) \leqslant \frac{1}{2} + \varepsilon$$

*Proof.* Let $N$ be such that $f(N) \leqslant \frac{1}{2}+\varepsilon$. Picking $N$ elements of $\mathcal{S}$, one can apply Proposition 9 to get two elements $x$ and $y$ at distance less than $f(N) \leqslant \frac{1}{2}+\varepsilon$. Picking $N$ other elements, one can get two more elements $x$ and $y$ verifying the condition. Repeating this process, one can find infinitely many $x$ and $y$. □

## 5.3 Bound Tightness for Regular Toeplitz Configurations

As the result on regular Toeplitz configurations comes from Proposition 7 on finite words with the Hamming distance, it can be adapted to configurations.

First, we need a simple lemma giving the relation between the Hamming distance and the Besicovitch distance for periodic configurations.

**Lemma 11.** *Let $u$ and $v$ be finite words of same length $\ell$, $x = u^\infty$ and $y = v^\infty$ the periodic configurations whose repeated pattern are $u$ and $v$ respectively. Then $d_B(x,y) = \frac{d_H(u,v)}{\ell}$.*

*Proof.* As $x_{\to n} = u^{\lfloor \frac{n}{\ell} \rfloor} u_{\to n-\lfloor \frac{n}{\ell} \rfloor}$ and $y_{\to n} = v^{\lfloor \frac{n}{\ell} \rfloor} v_{\to n-\lfloor \frac{n}{\ell} \rfloor}$, one has

$$\begin{aligned}d_B(x,y) &= \limsup_{n\to\infty} \frac{d_H(x_{\to n}, y_{\to n})}{n} \\ &= \limsup_{n\to\infty} \frac{1}{n}\left(\left\lfloor \frac{n}{\ell} \right\rfloor d_H(u,v) + d_H(u_{\to n-\lfloor \frac{n}{\ell} \rfloor}, v_{\to n-\lfloor \frac{n}{\ell} \rfloor})\right) \\ &= \limsup_{n\to\infty} \frac{d_H(u,v)}{n}\left\lfloor \frac{n}{\ell} \right\rfloor + \frac{d_H(u_{\to n-\lfloor \frac{n}{\ell} \rfloor}, v_{\to n-\lfloor \frac{n}{\ell} \rfloor})}{n} \ .\end{aligned}$$

The first term of the sum tends to $\frac{d_H(u,v)}{\ell}$ and the second to 0. □

The following result gives bound tightness.

**Proposition 12.** *For all integer $N$, there is a set $\mathcal{X}_N$ of cardinal $N$ of periodic (hence regular Toeplitz) configurations such that*

$$\forall u, v \in \mathcal{X}_N, d_B(u,v) = f(N) \ .$$

*There is an infinite set $\mathcal{X}_\infty$ of periodic configurations such that*

$$\forall u, v \in \mathcal{X}_\infty, d_B(u,v) = \frac{1}{2} \ .$$

*Proof.* Let $\mathcal{W}_N$ be the set introduced at the end of Section 5.1. Let $\mathcal{X}_N$ be the set of periodic configurations whose repeated words are the words of $\mathcal{W}_N$. Using Lemma 11, since $\forall x, y \in \mathcal{W}_N$, $d_H(x,y) = |x| f(N)$, one has $\forall u, v \in \mathcal{C}, d_B(u,v) = f(N)$.

Let $\mathcal{X}_\infty = \{(0^{2^i}1^{2^i})^\infty, \, i \in \mathbb{N}\}$ whose first members are represented Figure 3. Any two members of $\mathcal{X}_\infty$ are at distance $\frac{1}{2}$. □

```
01010101010101010101010101010101 ...
00110011001100110011001100110011 ...
00001111000011110000111100001111 ...
00000000111111110000000011111111 ...
00000000000000001111111111111111 ...
```

**Fig. 3.** first members of $\{(0^{2^i}1^{2^i})^\infty, \, i \in \mathbb{N}\}$

However, it remains open to find a set where configurations are all at distance strictly greater that one half, though for any $\varepsilon > 0$, one can find configurations whose relative distance is less than $\frac{1}{2} + \varepsilon$.

## References

1. Allouche, J.-P., Shallit, J.O.: Automatic Sequences - Theory, Applications, Generalizations. Cambridge University Press (2003)
2. Blanchard, F., Cervelle, J., Formenti, E.: Some results about the chaotic behavior of cellular automata. Theor. Comput. Sci. 349(3), 318–336 (2005)
3. Blanchard, F., Formenti, E., Kůrka, P.: Cellular automata in the Cantor, Besicovitch and Weyl Topological Spaces. Complex Systems 11, 107–123 (1999)
4. Cattaneo, G., Formenti, E., Margara, L., Mazoyer, J.: A Shift-invariant Metric on $S^{\mathbb{Z}}$ Inducing a Non-trivial Topology. In: Privara, I., Ružička, P. (eds.) MFCS 1997. LNCS, vol. 1295, pp. 179–188. Springer, Heidelberg (1997)
5. Durand, B.: Tilings and Quasiperiodicity. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 65–75. Springer, Heidelberg (1997)
6. Formenti, E.: On the sensitivity of cellular automata in Besic ovitch spaces. Theoretical Computer Science 301(1-3), 341–354 (2003)
7. Hedlund, G.A.: Endomorphism and automorphism of the shift dynamical system. Mathematical System Theory 3, 320–375 (1969)
8. Semakov, N.V., Zinov'ev, V.A.: Equidistant q-ary codes with maximal distance and resolvable balanced incomplete block designs. Problemy Peredachi Informatsii 4(2), 3–10 (1968)

# Approximate Regular Expressions and Their Derivatives

Jean-Marc Champarnaud, Hadrien Jeanne, and Ludovic Mignot

LITIS, Université de Rouen, 76801 Saint-Étienne du Rouvray Cedex, France
{jean-marc.champarnaud,hadrien.jeanne,ludovic.mignot}@univ-rouen.fr

**Abstract.** Several studies have been achieved to construct a finite automaton that recognizes the set of words that are at a bounded distance from some word of a given language. In this paper, we introduce a new family of regular operators based on a generalization of the notion of distance and we define a new family of expressions, the *approximate regular expressions*. We compute Brzozowski derivatives and Antimirov derivatives of such operators, which allows us to provide two recognizers for the language denoted by any approximate regular expression.

## 1 Introduction

The aim of this paper is to introduce automaton-theoretic constructions that make a sound foundation for the design of new approximate matching algorithms. Let us recall that approximate matching consists in locating the segments of the text that approximately correspond to the pattern to be matched, *i.e.* segments of the text that do not present too many errors with respect to the pattern. This research topic has numerous applications, in biology or in linguistics for example, and many algorithms have been designed in this framework for more than thirty years especially concerning approximate string matching (see [4,12] for a survey of such algorithms). Two contexts can be distinguished: in the off-line case, that is when a pre-computing of the text is performed, the basic tool is the construction of indexes [7]; otherwise, the basic technique is dynamic programming [10]. In both cases, automata constructions have been used, either to represent an index [16] or to simulate dynamic programming [6].

In most of the existing algorithms, the similarity between two words is measured by a distance and two basic types of distance called Hamming distance and Levenshtein distance (or edit distance) are generally considered. Our constructions provide a first kind of generalization since the similarity between two words is handled by a general word comparison function rather than by a distance (such a function is for instance not necessarily symmetrical).

Several studies address the problem of constructing a finite automaton that recognizes the language of all the words that are at a distance less than or equal to a given positive integer $k$ from a given word. For instance this problem is considered in [5] where Hamming distance is used and in [15] where Levenshtein distance is used. A challenging problem is to tackle the more general case where the pattern is no longer a word but a regular expression [13]. The solution described in [9] first computes $k+1$ clones of some non-deterministic automaton recognizing the language of the regular expression and then interconnects these clones by a set of transitions that depends on the type of distance.

We solve this problem by defining a new family of operators: given an integer $k$, the $\mathbb{F}_k$ operator is such that, for any regular language $L$, the language $\mathbb{F}_k(L)$ is the set of all the words that are at a distance less than or equal to $k$ from some word of $L$. We then consider the family of *approximate regular expressions* obtained from the family of regular expressions by adding the family of $\mathbb{F}_k$ operators to the set of regular operators. We first provide a formula that, given a regular language $L$, computes the quotient of the language $\mathbb{F}_k(L)$ with respect to a symbol. We then extend the computation of Brzozowski derivatives [2] (resp. of Antimirov derivatives [1]) to the family of approximate regular expressions and we show that the set of Brzozowski derivatives (resp. of Antimirov derivatives) of an approximate regular expression is finite . As a consequence, the language denoted by any approximate regular expression is regular . Our main result is the construction of two recognizers for the language denoted by an approximate regular expression: the deterministic automaton of Brzozowski derivatives and the non-deterministic automaton of Antimirov derivatives.

Classical notions of language theory, such as derivative computation, are recalled in Section 2. Section 3 gives a formalization of the notion of word comparison function and provides a definition of the family of approximate regular expressions. Finally, the derivative-based constructions of an automaton from an approximate regular expression are presented in Section 4.

## 2 Preliminaries

A *finite automaton* $A$ is a 5-tuple $(\Sigma, Q, I, F, \delta)$ with $\Sigma$ the *alphabet* (a finite set of symbols), $Q$ a finite set of *states*, $I \subset Q$ the set of *initial states*, $F \subset Q$ the set of *final states* and $\delta \subset Q \times \Sigma \times Q$ the set of *transitions*. The set $\delta$ is equivalent to the function from $Q \times \Sigma$ to $2^Q$ defined by: $q' \in \delta(q,a)$ if and only if $(q,a,q') \in \delta$. The domain of the function $\delta$ is extended to $2^Q \times \Sigma^*$ as follows: $\forall P \subset Q$, $\delta(P,\varepsilon) = P$, $\delta(P,a) = \bigcup_{p \in P} \delta(p,a)$ and $\delta(P, a \cdot w) = \delta(\delta(P,a),w)$. The automaton $A$ *recognizes* the language $L(A) = \{w \in \Sigma^* \mid \delta(I,w) \cap F \neq \emptyset\}$. The automaton $A$ is *deterministic* if $\mathrm{Card}(I) = 1$ and $\forall (q,a) \in Q \times \Sigma$, $\mathrm{Card}(\delta(q,a)) \leq 1$. A *regular expression* $E$ over an alphabet $\Sigma$ is inductively defined by $E = \emptyset$, $E = \varepsilon$, $E = a$, $E = (F+G)$, $E = (F \cdot G)$, $E = (F^*)$ where $a$ is any symbol in $\Sigma$ and $F$ and $G$ are any two regular expressions. The *language $L(E)$ denoted by $E$* is inductively defined by $L(\emptyset) = \emptyset$, $L(a) = \{a\}$, $L(E+F) = L(E) \cup L(F)$, $L(E \cdot F) = L(E) \cdot L(F)$ and $L(F^*) = (L(F))^*$ where $a$ is any symbol in $\Sigma$, $F$ and $G$ are any two regular expressions, and for any $L_1, L_2 \subset \Sigma^*$, $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$, $L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$ and $L_1^* = \{w_1 \cdots w_k \mid k \geq 1 \wedge \forall j \in \{1,\ldots,k\},\ w_j \in L_1\} \cup \{\varepsilon\}$. A language $L$ is *regular* if and only if there exists a regular expression $E$ such that $L(E) = L$. It has been proved by Kleene [8] that a language is regular if and only if it is recognized by a finite automaton.

The *quotient of* $L$ w.r.t. a symbol $a$ is the language $a^{-1}(L) = \{w \in \Sigma^* \mid aw \in L\}$. It can be recursively computed as follows:

$$\begin{array}{rr} a^{-1}(\emptyset) = a^{-1}(\{\varepsilon\}) = a^{-1}(\{b\}) = \emptyset, & a^{-1}(\{a\}) = \{\varepsilon\} \\ a^{-1}(L_1 \cup L_2) = a^{-1}(L_1) \cup a^{-1}(L_2), & a^{-1}(L_1^*) = a^{-1}(L_1) \cdot L_1^* \end{array}$$
$$a^{-1}(L_1 \cdot L_2) = \begin{cases} a^{-1}(L_1) \cdot L_2 \cup a^{-1}(L_2) & \text{if } \varepsilon \in L_1, \\ a^{-1}(L_1) \cdot L_2 & \text{otherwise.} \end{cases}$$

The quotient $w^{-1}(L)$ of $L$ w.r.t. a word $w$ is the set $\{w' \in \Sigma^* \mid w \cdot w' \in L\}$. It can be recursively computed as follows: $\varepsilon^{-1}(L) = L$, $(aw')^{-1}(L) = w'^{-1}(a^{-1}(L))$ with $a \in \Sigma$ and $w' \in \Sigma^*$. The Myhill-Nerode Theorem [11,14] states that a language $L$ is regular if and only if the set of quotients $\{u^{-1}(L) \mid u \in \Sigma^*\}$ is finite. The derivative of an expression $E$ w.r.t. a word $w$ is an expression denoting the quotient of $L(E)$ w.r.t. $w$.

The notion of derivative of an expression has been introduced by Brzozowski [2]. Let $E$ be a regular expression over an alphabet $\Sigma$ and let $a$ and $b$ be two distinct symbols of $\Sigma$. The *derivative of* $E$ w.r.t. $a$ is the expression $\frac{d}{d_a}(E)$ inductively computed as follows :

$$\frac{d}{d_a}(\emptyset) = \frac{d}{d_a}(\varepsilon) = \frac{d}{d_a}(b) = \emptyset, \quad \frac{d}{d_a}(a) = \varepsilon,$$
$$\frac{d}{d_a}(F^*) = \frac{d}{d_a}(F) \cdot F^*, \quad \frac{d}{d_a}(F+G) = \frac{d}{d_a}(F) + \frac{d}{d_a}(G)$$
$$\frac{d}{d_a}(F \cdot G) = \begin{cases} \frac{d}{d_a}(F) \cdot G + \frac{d}{d_a}(G) & \text{if } \varepsilon \in L(F), \\ \frac{d}{d_a}(F) \cdot G & \text{otherwise.} \end{cases}$$

The derivative of $E$ is extended to words of $\Sigma^*$ as follows:

$$\frac{d}{d_\varepsilon}(E) = E, \ \frac{d}{d_{aw}}(E) = \frac{d}{d_w}(\frac{d}{d_a}(E)).$$

The set of derivatives of an expression $E$ is not necessarily finite. It has been proved by Brzozowski [2] that it is sufficient to use the ACI equivalence (that is based on the associativity, the commutativity and the idempotence of the sum) to obtain a finite set of derivatives: the set $\mathcal{D}_E$ of *dissimilar derivatives*. Given a class of ACI-equivalent expressions, a unique representative can be obtained after deleting parenthesis (associativity), ordering terms of each sum (commutativity) and deleting redundant subexpressions (idempotence). Let $E_{\sim_s}$ be the unique representative of the class of the expression $E$. The set of dissimilar derivatives can be computed as follows:

$$\frac{d'}{d'_a}(\emptyset) = \frac{d'}{d'_a}(\varepsilon) = \frac{d'}{d'_a}(b) = \emptyset, \ \frac{d'}{d'_a}(a) = \varepsilon,$$
$$\frac{d'}{d'_a}(E+F) = (\frac{d'}{d'_a}(F) + \frac{d'}{d'_a}(G))_{\sim_s}, \ \frac{d'}{d'_a}(F^*) = \frac{d'}{d'_a}(F) \cdot F^*,$$
$$\frac{d'}{d'_a}(F \cdot G) = \begin{cases} (\frac{d'}{d'_a}(F) \cdot G + \frac{d'}{d'_a}(G))_{\sim_s} & \text{if } \varepsilon \in L(F), \\ (\frac{d'}{d'_a}(F) \cdot G)_{\sim_s} & \text{otherwise.} \end{cases}$$

The *derivative automaton* $B = (\Sigma, Q, \{q_0\}, F, \delta)$ of a regular expression $E$ over an alphabet $\Sigma$ is defined by $Q = \mathcal{D}_E$, $q_0 = E$, $F = \{q \in Q \mid \varepsilon \in L(q)\}$, $\delta = \{(q, a, q') \in Q \times \Sigma \times Q \mid \frac{d'}{d'_a}(q) = q'\}$. The automaton $B$ is deterministic and it recognizes the language $L(E)$. Its size can be exponentially larger than the number of symbols of $E$.

Antimirov's algorithm [1] constructs a non-deterministic automaton from a regular expression $E$. It is based on the *partial derivative* computation. The partial derivative of a simple regular expression $E$ w.r.t. a symbol $a$ is the set $\frac{\partial}{\partial_a}(E)$ of expressions defined as follows:

$$\frac{\partial}{\partial_a}(\emptyset) = \frac{\partial}{\partial_a}(\varepsilon) = \frac{\partial}{\partial_a}(b) = \emptyset,\ \frac{\partial}{\partial_a}(a) = \{\varepsilon\},$$
$$\frac{\partial}{\partial_a}(F+G) = \frac{\partial}{\partial_a}(F) \cup \frac{\partial}{\partial_a}(G),\ \frac{\partial}{\partial_a}(F^*) = \frac{\partial}{\partial_a}(F)\cdot F^*,$$
$$\frac{\partial}{\partial_a}(F\cdot G) = \begin{cases} \frac{\partial}{\partial_a}(F)\cdot G \cup \frac{\partial}{\partial_a}(G) & \text{if } \varepsilon \in L(F),\\ \frac{\partial}{\partial_a}(F)\cdot G & \text{otherwise,}\end{cases}$$

with for any set $\mathcal{E}$ of expressions, $\mathcal{E}\cdot F = \bigcup_{E\in\mathcal{E}} E\cdot F$.
The partial derivative of $E$ is extended to words of $\Sigma^*$ as follows:

$$\frac{\partial}{\partial_\varepsilon}(E) = \{E\},\ \frac{\partial}{\partial_{aw}}(E) = \frac{\partial}{\partial_w}(\frac{\partial}{\partial_a}(E)),$$

with for a set $\mathcal{E}$ of expressions, $\frac{\partial}{\partial_a}(\mathcal{E}) = \bigcup_{E\in\mathcal{E}} \frac{\partial}{\partial_a}(E)$. Every element of a partial derivative is called a *derivated term* of $E$. Antimirov [1] has shown that the set $\mathcal{D}'_E$ of the derivated terms of $E$ is such that $\mathrm{Card}(\mathcal{D}'_E) \le n+1$, where $n$ is the number of symbols of $E$. The *derivated term automaton* $A = (\Sigma, Q, \{q_0\}, T, \delta)$ of a simple regular expression $E$ is defined as follows: $Q = \mathcal{D}'_E$, $q_0 = E$, $F = \{q \in Q \mid \varepsilon \in L(q)\}$, $\delta = \{(q,a,q') \in Q\times\Sigma\times Q \mid q' \in \frac{\partial}{\partial_a}(q)\}$. The automaton $A$ recognizes the language $L(E)$.

## 3 Comparison Functions: Symbols, Sequences and Words

Let $\Sigma$ be an alphabet, $S = \Sigma \cup \{\varepsilon\}$ and $X$ be a subset of $S\times S$. A *cost function* C over $X$ is a function from $X$ to $\mathbb{N}$ satisfying **Condition 1:** for all $\alpha$ in $S$, $\mathrm{C}(\alpha,\alpha) = 0$. For any pair $(\alpha,\beta)$ in $S\times S$ such that $\mathrm{C}(\alpha,\beta)$ is not defined, let us set $\mathrm{C}(\alpha,\beta) = \perp$. Consequently, a cost function can be viewed as a function from $S\times S$ to $\mathbb{N}\cup\{\perp\}$ satisfying Condition 1. Since we use $\perp$ to deal with undefined computation, we set for all $x$ in $\mathbb{N}\cup\{\perp\}$, $\perp + x = x + \perp = x - \perp = \perp - x = \perp$ and for all integers $x, y$ in $\mathbb{N}$, $x - y = \perp$ when $y > x$. A cost function can be represented by a directed and labelled graph $\mathrm{C} = \{S, V\}$ where $V$ is a subset of $S\times(\mathbb{N}\cup\{\perp\})\times S$ such that for all $(\alpha,\beta)$ in $S\times S$, $\mathrm{C}(\alpha,\beta) = k \Leftrightarrow (\alpha,k,\beta) \in V$. Transitions labelled by $\perp$ can be omitted in the graphical representation, as well as the implicit transitions $(\alpha,0,\alpha)$ (See Example 1).

*Example 1.* Let $\Sigma = \{a,b,c\}$. Let C be the cost function defined as follows:

$$\mathrm{C}(x,y) = \begin{cases} 0 & \text{if } x = y,\\ 4 & \text{if } x = a \wedge y = c,\\ 3 & \text{if } x = c \wedge y = a,\\ 1 & \text{if } x \in \{a,c\} \wedge y = b,\\ \perp & \text{otherwise.}\end{cases}$$

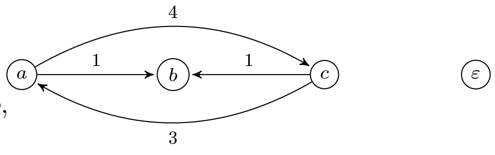


**Fig. 1.** The cost function C

The cost function C can be represented by the graph in Figure 1.

Given a positive integer $k$ we now consider the set $S^k$ of all the *s*equences $s = (s_1, \ldots, s_k)$ of size $k$ made of elements of $S$. A *sequence comparison function* is a function $\mathcal{F}$ from $\bigcup_{k\in\mathbb{N}} S^k \times S^k$ to $\mathbb{N} \cup \{\bot\}$. Given a pair $(s, s')$ of sequences with the same size, $\mathcal{F}(s, s')$ either is an integer or is undefined. In the following we will consider sequence comparison functions $\mathcal{F}$ satisfying **Condition 2:** $\mathcal{F}$ is defined from a given cost function C over $S \times S$, and **Condition 3:** $\mathcal{F}$ is a *symbol-wise* comparison function, that is, for any two sequences $s = (s_1, \ldots, s_n)$ and $s' = (s'_1, \ldots, s'_n)$, it holds:

$$\mathcal{F}(s, s') = \mathcal{F}((s_1), (s'_1)) + \mathcal{F}((s_2, \ldots, s_n), (s'_2, \ldots, s'_n)) = \textstyle\sum_{k\in\{1,\ldots,n\}} \mathcal{F}((s_k), (s'_k)).$$

We consider that those functions satisfy Condition 1: for all $\alpha$ in $S$, $\mathcal{F}((\alpha), (\alpha)) = 0$. Consequently, for any pair of sequences $s = (s_1, \ldots, s_k)$ and $s' = (s'_1, \ldots, s'_k)$ such that $k \neq 1$, **Condition 4** is satisfied: if there exists an integer $k'$ in $\{1, \ldots, k\}$ such that $s_{k'} = s'_{k'} = \varepsilon$, then:

$$\mathcal{F}(s, s') = \mathcal{F}((s_1, \ldots, s_{k'-1}, s_{k'+1}, \ldots, s_k), (s'_1, \ldots, s'_{k'-1}, s'_{k'+1}, \ldots, s'_k)).$$

As a consequence of Condition 3, a symbol-wise sequence comparison function is defined by the images of the pairs of sequences of size 1. Notice that a sequence comparison function is not necessarily symbol-wise, *e.g.* for a given cost function F, $\mathcal{F}((s_1, \ldots, s_n), (s'_1, \ldots, s'_n)) = \sum_{k\in\{1,\ldots,n\}} \mathrm{F}(s_k, s'_k)^k$.
Two of the most well-known symbol-wise sequence comparison functions are the Hamming one ($\mathcal{H}$) and the Levenshtein one ($\mathcal{L}$) respectively defined for any pair of sequences $s = (s_1, \ldots, s_n)$ and $s' = (s'_1, \ldots, s'_n)$ in $S^n \times S^n$ and any integer $n > 0$ by:

$$\mathcal{H}(s, s') = \textstyle\sum_{k\in\{1,\ldots,n\}} \mathrm{H}(s_k, s'_k), \quad \mathcal{L}(s, s') = \textstyle\sum_{k\in\{1,\ldots,n\}} \mathrm{L}(s_k, s'_k),$$

with H and L the two cost functions defined for all $a, b$ in $\Sigma \cup \{\varepsilon\}$ by:

$$\mathrm{H}(a, b) = \begin{cases} \bot & \text{if } (a = \varepsilon \vee b = \varepsilon) \wedge (a, b) \neq (\varepsilon, \varepsilon), \\ 1 & \text{if } a \neq b, \\ 0 & \text{otherwise,} \end{cases} \qquad \text{and } \mathrm{L}(a, b) = \begin{cases} 1 & \text{if } a \neq b, \\ 0 & \text{otherwise.} \end{cases}$$

Let us now explain how a word function comparison can be deduced from a sequence comparison function. Let $w$ be a word in $\Sigma^*$ and $|w|$ be its *length.* The sequence $s = (s_1, \ldots, s_n)$ in $S^n$ is said to be a *split-up* of $w$ if $s_1 \cdots s_n = w$. The set of all the split-ups of size $k$ of a word $w$ is denoted by $\mathrm{Split}_k(w)$ and the set of all the split-ups of $w$ is denoted by $\mathrm{Split}(w)$ .

Let $\mathcal{F}$ be a sequence comparison function, $(u, v)$ be a pair of words of $\Sigma^*$, and $k$ be a positive integer. We consider the following sets:

$$Y(u, v) = \{\mathcal{F}(u', v') \mid \exists k \in \mathbb{N}, k \geq 1 \wedge (u', v') \in \mathrm{Split}_k(u) \times \mathrm{Split}_k(v)\} \cap \mathbb{N},$$
$$Y_m(u, v) = \{\mathcal{F}(u', v') \mid \exists k \in \mathbb{N}, 1 \leq k \leq m \wedge (u', v') \in \mathrm{Split}_k(u) \times \mathrm{Split}_k(v)\} \cap \mathbb{N}.$$

**Definition 2.** *Let $\mathcal{F}$ be a sequence comparison function. The* word comparison function *associated with $\mathcal{F}$ is the function $\mathbb{F}$ from $\Sigma^* \times \Sigma^*$ to $\mathbb{N} \cup \{\bot\}$ defined by:*

$$\mathbb{F}(u, v) = \min\{Y(u, v)\} \quad \textit{if } Y(u, v) \neq \emptyset, \qquad \mathbb{F}(u, v) = \bot \ \textit{otherwise.}$$

In the case of a sequence comparison function based on a cost function, the whole set $\mathbb{N}$ needs not to be considered. Indeed, according to Condition 4, if $u \neq \varepsilon$ or $v \neq \varepsilon$, then $Y(u, v) = Y_{|u|+|v|}(u, v)$ and we can write:

$$\mathbb{F}(u,v)=\begin{cases}0 & \text{if } u=v=\varepsilon,\\ \min\{Y_{|u|+|v|}\} & \text{if } (u,v)\neq(\varepsilon,\varepsilon)\wedge Y_{|u|+|v|}\neq\emptyset,\\ \perp & \text{otherwise.}\end{cases}$$

The *Hamming distance* $\mathbb{H}$ and the *Levenshtein distance* $\mathbb{L}$ are the word comparison functions respectively associated to the sequence comparison functions $\mathcal{H}$ and $\mathcal{L}$. Both of them satisfy the properties of word distances[1]. Notice that in the following we will handle word comparison functions that are not necessarily distance ones (*e.g.* Example 1).

*Example 3.* Let C be the cost function defined in Example 1. Let $s=(s_1)$ and $s'=(s'_1)$ be two sequences of size 1. We define four symbol-wise sequence comparison functions by setting the images of the pairs of sequences of size 1 from the cost function C.

$$\rightarrow^{\mathrm{C}}(s,s')=\mathrm{C}(s_1,s'_1),\quad \leftrightarrow^{\mathrm{C}}(s,s')=\min\{\mathrm{C}(s_1,s'_1),\mathrm{C}(s'_1,s_1)\},$$
$$\leftarrow^{\mathrm{C}}(s,s')=\mathrm{C}(s'_1,s_1),\quad \rightrightarrows^{\mathrm{C}}(s,s')=\min_{x\in\Sigma\cup\{\varepsilon\}}\{\mathrm{C}(s_1,x)+\mathrm{C}(s'_1,x)\}.$$

Let us consider the two split-ups $s=(a,c,a)$ and $s'=(c,a,c)$. According to Figure 2, it holds:

$$\rightarrow^{\mathrm{C}}(s,s')=11,$$
$$\leftarrow^{\mathrm{C}}(s,s')=10,$$
$$\leftrightarrow^{\mathrm{C}}(s,s')=9,$$
$$\rightrightarrows^{\mathrm{C}}(s,s')=6.$$

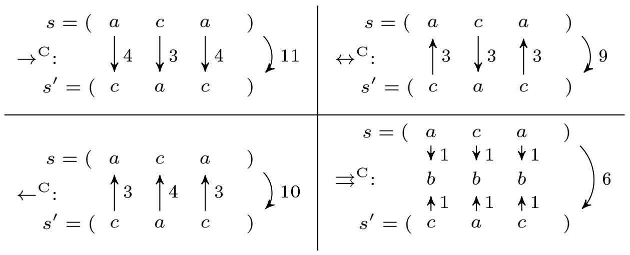


**Fig. 2.** Examples of sequence comparisons

Any word comparison function can be used as an operator for languages, leading to an extension of the expressivity of expressions for any bounded length.

**Definition 4.** *Let $L$ be a language over an alphabet $\Sigma$, $\mathbb{F}$ a word comparison function and $k$ an integer in $\mathbb{N}\cup\{\perp\}$. Then:*

$$\mathbb{F}_k(L)=\begin{cases}\{w\in\Sigma^*\mid \exists u\in L,\mathbb{F}(w,u)\in\{0,\ldots,k\}\} & \textit{if } k\in\mathbb{N},\\ \emptyset & \textit{otherwise.}\end{cases}$$

Let us notice that $\mathbb{F}_k(\mathbb{F}_{k'}(L))$ is not necessarily equal to $\mathbb{F}_{k+k'}(L)$. Indeed, let us consider the three languages $L_1=\mathbb{F}_1(\{a\})$, $L_2=\mathbb{F}_1(\mathbb{F}_1(\{a\}))$ and $L_3=\mathbb{F}_2(\{a\})$ over the alphabet $\Sigma=\{a,b\}$ with $\mathbb{F}$ the word comparison function associated with the symbol-wise sequence comparison function $\mathcal{F}$ defined for any symbol $\alpha,\beta$ by $\mathcal{F}((\alpha),(\beta))=0$ if $\alpha=\beta$, $\mathcal{F}((\alpha),(\beta))=2$ otherwise. Then $L_1=L_2=\{a\}$ whereas $L_3=\{\varepsilon,a,b,aa,ab,ba\}$.

[1] A *word distance* $\mathbb{D}$ is a word comparison function over words in $\Sigma^*$ satisfying the three following properties for all $x,y,z\in\Sigma^*$: **(1)** $\mathbb{D}(x,y)=0\Rightarrow x=y$, **(2)** $\mathbb{D}(x,y)=\mathbb{D}(y,x)$, **(3)** $\mathbb{D}(x,y)+\mathbb{D}(y,z)\geq\mathbb{D}(x,z)$.

An *approximate regular expression* (**ARE**) $E$ over an alphabet $\Sigma$ is inductively defined by $E = \emptyset$, $E = \varepsilon$, $E = a$, $E = F + G$, $E = (F \cdot G)$, $E = (F^*)$, $E = \mathbb{F}_k(F)$ where $a$ is any symbol in $\Sigma$, $F$ and $G$ are any two AREs, $\mathbb{F}$ is any symbol-wise word comparison function and $k$ is any integer in $\mathbb{N} \cup \{\bot\}$. The *language denoted* by an ARE $E$ is the language $L(E)$ inductively defined by $L(\emptyset) = \emptyset$, $L(a) = \{a\}$, $L(F + G) = L(F) \cup L(G)$, $L(F \cdot G) = L(F) \cdot L(G)$, $L(F^*) = L(F)^*$ and $L(\mathbb{F}_k(F)) = \mathbb{F}_k(L(F))$ where $a$ is any symbol in $\Sigma$, $F$ and $G$ are any two AREs, $\mathbb{F}$ is any symbol-wise word comparison function and $k$ is any integer in $\mathbb{N} \cup \{\bot\}$. In order to prove that the language denoted by an ARE $E$ is regular, we show how to compute a finite automaton recognizing $L(E)$.

## 4 Word Comparison Functions, Quotients and Derivatives

In this section, we present two constructions of an automaton from an ARE using Brzozowski's derivatives and Antimirov's ones, respectively leading to a deterministic automaton and a non-deterministic one. We first show how to compute the quotient of a given language $\mathbb{F}_k(L)$ w.r.t. a symbol $a$, where $\mathbb{F}$ is a given word comparison function, $k$ an integer and $L$ a regular language.

### 4.1 Quotient of an $\mathbb{F}_k(L)$ Language

Let $\mathbb{F}$ be a word comparison function associated with a symbol-wise sequence comparison function $\mathcal{F}$ defined over an alphabet $\Sigma$. Let $k$ be a positive integer, $a$ be a symbol in $\Sigma$, $u = aw$ be a word of $\Sigma^+$, and $L'$ be a regular language of $\Sigma^*$. According to Definition 4, the word $u$ is in $L = \mathbb{F}_k(L')$ if and only if there exists a word $v \in L'$ such that $\mathbb{F}(u, v) \leq k$. According to Definition 2, this is equivalent to the existence of an alignment $(u', v') \in \text{Split}_n(u) \times \text{Split}_n(v)$, where $n$ is a positive integer, between $u$ and $v$, the cost $\mathcal{F}(u', v')$ of which is not greater than $k$. Let $u' = (u'_1, \ldots, v'_n)$ and $v' = (v'_1, \ldots, v'_n)$. **(a)** If $n = 1$, $\mathbb{F}(u, v) = \mathcal{F}((a), (v'_1))$ and since $u = aw$, $a \in L \Leftrightarrow w \in \mathbb{F}_{k-\mathcal{F}((a),(v'_1))}{v'_1}^{-1}(L')$. **(b)** Otherwise, let us set $u'' = (u'_2, \ldots, u'_n)$ and $v'' = (v'_2, \ldots, v'_n)$. Moreover, let us set $t = u$ if $u'_1 = \varepsilon$ and $t = u'_2 \cdots u'_n$ otherwise; let us similarly set $z = v$ if $v'_1 = \varepsilon$ and $z = v'_2 \cdots v'_n$ otherwise. Obviously, the word $z$ belongs to ${v'_1}^{-1}(L')$. Since $\mathbb{F}$ is a symbol-wise word comparison function, there exists an alignment $(u', v')$ between $u$ and $v$ satisfying $\mathcal{F}(u', v') \leq k$ if and only if there exists an alignment $(u'', v'')$ between $t$ and $z$ satisfying $\mathcal{F}(u'', v'') \leq k - \mathcal{F}((u'_1), (v'_1))$. According to Definition 2, this is equivalent to the existence of a word $z \in {v'_1}^{-1}(L')$ such that $\mathbb{F}(t, z) \leq k - \mathcal{F}((u'_1), (v'_1))$. According to Definition 4, it is equivalent to say that the word $t$ is in $\mathbb{F}_{k-\mathcal{F}((u'_1),(v'_1))}({v'_1}^{-1}(L'))$. Depending on the value of $(u'_1, v'_1)$ we can distinguish the following cases:

**Case 1** $(u'_1, v'_1) = (a, b)$, with $b \in \Sigma$: $u = aw \in L \Leftrightarrow w \in \mathbb{F}_{k-\mathcal{F}(a,b)}(b^{-1}L')$,
**Case 2** $(u'_1, v'_1) = (a, \varepsilon)$ with $a \in \Sigma$: $u = aw \in L \Leftrightarrow w \in \mathbb{F}_{k-\mathcal{F}(a,\varepsilon)}(L')$,
**Case 3** $(u'_1, v'_1) = (\varepsilon, b)$, with $b \in \Sigma$: $u = aw \in L \Leftrightarrow w \in a^{-1}(\mathbb{F}_{k-\mathcal{F}(\varepsilon,b)}(b^{-1}L'))$.
Since $w \in a^{-1}\mathbb{F}_k(L') \Leftrightarrow aw \in \mathbb{F}_k(L')$, the three previous cases provide a recursive expression of the quotient of the language $\mathbb{F}_k(L')$ w.r.t. a symbol $a \in \Sigma$.

Unfortunately, its computation may imply a recursive loop, due to Case 3, when $\mathcal{F}((\varepsilon),(b)) = 0$. It is possible to get rid of this loop by precomputing the set of all the quotients of $L'$ w.r.t. words $w$ such that $\mathbb{F}(\varepsilon, w) = 0$. In this purpose, let us set $\mathcal{W}_\mathcal{F} = (\bigcup_{b\in\Sigma,\mathcal{F}((\varepsilon),(b))=0}\{b\})^*$ and $X(L') = \{L'\} \cup \bigcup_{w\in\mathcal{W}_\mathcal{F}}\{w^{-1}(L')\}$. Let us notice that if $L'$ is a regular language, the set of its residuals is finite; as a consequence, so is $X(L')$.

**Lemma 5.** *Let $L = \mathbb{F}_k(L')$ be a language over an alphabet $\Sigma$ where $L'$ is a regular language, $\mathbb{F}$ is a symbol-wise word comparison function associated with a sequence comparison function $\mathcal{F}$ and $a$ be a symbol in $\Sigma$. The* quotient *of $L$ w.r.t. $a$ is the language $a^{-1}(L)$ computed as follows:*

$$a^{-1}(L)=\begin{cases} \bigcup_{L''\in X(L'),b\in\Sigma}(\mathbb{F}_{k-\mathcal{F}((a),(b))}(b^{-1}(L'')))\cup\bigcup_{L''\in X(L')}\mathbb{F}_{k-\mathcal{F}((a),(\varepsilon))}(L'')\\ \cup\ a^{-1}(\bigcup_{L''\in X(L'),b\in\Sigma,\mathcal{F}((\varepsilon),(b))\neq 0}(\mathbb{F}_{k-\mathcal{F}((\varepsilon),(b))}(b^{-1}(L''))))\end{cases}$$

*where $X(L') = \{L'\} \cup \bigcup_{w\in\mathcal{W}_\mathcal{F}} w^{-1}(L')$ with $\mathcal{W}_\mathcal{F} = (\bigcup_{b\in\Sigma,\mathcal{F}((\varepsilon),(b))=0}\{b\})^*$.*

*Proof.* For any symbol $\alpha, \beta$ in $\Sigma \cup \{\varepsilon\}$, let us set $k_{\alpha,\beta} = k - \mathcal{F}((\alpha),(\beta))$.

$$u \in a^{-1}(L) \Leftrightarrow au \in L \Leftrightarrow \exists w \in L', \mathbb{F}(au, w) \in \{0,\ldots,k\}$$

$$\Leftrightarrow \begin{cases} \exists b \in \Sigma, \exists w_1bw_2 \in L', \mathbb{F}(\varepsilon, w_1) = 0 \wedge \mathbb{F}(u, w_2) \le k_{a,b}\\ \vee\ \exists w_1w_2 \in L', \mathbb{F}(\varepsilon, w_1) = 0 \wedge \mathbb{F}(u, w_2) \le k_{a,\varepsilon}\\ \vee\ \exists b \in \Sigma, \exists w_1bw_2 \in L', \mathbb{F}(\varepsilon, w_1) = 0 \wedge \mathcal{F}((\varepsilon),(b)) \neq 0 \wedge \mathbb{F}(au, w_2) \le k_{\varepsilon,b}\end{cases}$$

$$\Leftrightarrow \begin{cases} \exists b \in \Sigma, \exists w_1 \in \mathcal{W}_\mathcal{F}, \exists w_2 \in (w_1b)^{-1}(L'), \mathbb{F}(u, w_2) \le k_{a,b}\\ \vee\ \exists w_1 \in \mathcal{W}_\mathcal{F}, \exists w_2 \in (w_1)^{-1}(L'), \mathbb{F}(u, w_2) \le k_{a,\varepsilon}\\ \vee\ \exists b \in \Sigma, \exists w_1 \in \mathcal{W}_\mathcal{F}, \exists w_2 \in (w_1b)^{-1}L', \mathcal{F}((\varepsilon),(b)) \neq 0 \wedge \mathbb{F}(au, w_2) \le k_{\varepsilon,b}\end{cases}$$

$$\Leftrightarrow \begin{cases} \exists b \in \Sigma, \exists w_2 \in b^{-1}(\bigcup_{L''\in X(L')} L''), \mathbb{F}(u, w_2) \le k_{a,b}\\ \vee\ \exists w_2 \in \bigcup_{L''\in X(L')} L'', \mathbb{F}(u, w_2) \le k_{a,\varepsilon}\\ \vee\ \exists b \in \Sigma, \exists w_2 \in b^{-1}(\bigcup_{L''\in X(L')} L''), \mathcal{F}((\varepsilon),(b)) \neq 0 \wedge \mathbb{F}(au, w_2) \le k_{\varepsilon,b}\end{cases}$$

$$\Leftrightarrow \begin{cases} \exists b \in \Sigma, u \in \mathbb{F}_{k_{a,b}} \bigcup_{L''\in X(L')} b^{-1}(L'')\\ \vee\ u \in \bigcup_{L''\in X(L')} \mathbb{F}_{k_{a,\varepsilon}}(L'')\\ \vee\ \exists b \in \Sigma, au \in \mathbb{F}_{k_{\varepsilon,b}}(\bigcup_{L''\in X(L')} b^{-1}(L''))\end{cases}$$

$$\Leftrightarrow \begin{cases} u \in \bigcup_{L''\in X(L'),b\in\Sigma} \mathbb{F}_{k-\mathcal{F}((a),(b))} b^{-1}(L'')\\ \vee\ u \in \bigcup_{L''\in X(L')} \mathbb{F}_{k-\mathcal{F}((a),(\varepsilon))}(L'')\\ \vee\ u \in a^{-1}(\bigcup_{L''\in X(L'),b\in\Sigma,\mathcal{F}((\varepsilon),(b))\neq 0} \mathbb{F}_{k-\mathcal{F}((\varepsilon),(b))} b^{-1}(L''))\end{cases}$$

□

## 4.2 Brzozowski Derivatives

An extension of Brzozowski derivatives can be directly deduced from the computation of the quotient presented in Lemma 5.

**Definition 6.** *Let $E = \mathbb{F}_k(E')$ be an ARE over an alphabet $\Sigma$ where $\mathbb{F}$ is associated with $\mathcal{F}$ and $a$ be a symbol in $\Sigma$. The* dissimilar derivative *of $E$ w.r.t. $a$ is the expression $\frac{d'}{d'_a}(E)$ defined by:*

$$\frac{d'}{d'_a}(E) = \left(\begin{array}{l} \sum_{F\in X(E'),b\in\Sigma}(\mathbb{F}_{k-\mathcal{F}((a),(b))}(\frac{d'}{d'_b}(F)))\\ +\ \sum_{F\in X(E')} \mathbb{F}_{k-\mathcal{F}((a),(\varepsilon))}(F)\\ +\ \frac{d'}{d'_a}(\sum_{F\in X(E'),b\in\Sigma,\mathcal{F}((\varepsilon),(b))\neq 0}(\mathbb{F}_{k-\mathcal{F}((\varepsilon),(b))}(\frac{d'}{d'_b}(F))))\end{array}\right)_{\sim_s},$$

*where* $X(E') = \{E'\} \cup \bigcup_{w \in \mathcal{W}_{\mathcal{F}}} \frac{d'}{d'_w}(E')$ *with* $\mathcal{W}_{\mathcal{F}} = (\bigcup_{b \in \Sigma, \mathcal{F}((\varepsilon),(b))=0} \{b\})^*$.

Let us show that the set of dissimilar derivatives of $E$ is finite (Lemma 7), that the computed expression satisfies the relation between derivation and quotient (Lemma 8) and how to determine whether the empty word belongs to the language denoted by an ARE (Lemma 9).

**Lemma 7.** *Let* $E = \mathbb{F}_k(E')$ *be an ARE over an alphabet* $\Sigma$ *and* $\mathcal{D}_E$ *be the set of dissimilar derivatives of* $E$*. Then* $\mathcal{D}_E$ *is a finite set of AREs. Moreover, its computation halts.*

*Proof.* Consider that $\mathbb{F}$ is associated with $\mathcal{F}$. Let us show by induction over the structure of $E'$ and by recurrence over $k$ that $\mathcal{D}_E$ is a finite set of AREs.

By induction, the set $\mathcal{D}_{E'}$ is a finite set of AREs. Consequently, since $X(E')$ is a subset of $\mathcal{D}_{E'}$, **(Fact 1)** $X(E')$ is a finite set of derivatives of $E'$.

In order to show that $\mathcal{D}_E$ is a finite set, let us show that any derivative $G$ of $E$ satisfies the property $\mathrm{P}(E', k)$: $G$ is a finite sum of expressions of type $\mathbb{F}_{k'}(G')$ with $k' \leq k$ and $G'$ a derivative of $E'$.

According to **Fact 1**, any subexpression $\mathbb{F}_{k-\mathcal{F}((a),(\varepsilon))}(F)$ with $F \in X(E')$ satisfies $\mathrm{P}(E', k)$. Since $X(E')$ is a subset of $\mathcal{D}_{E'}$, $\frac{d'}{d'_b}(F)$ is a derivative of $E'$ for any $b$ in $\Sigma$. Consequently, $\sum_{F \in X(E'), b \in \Sigma} (\mathbb{F}_{k-\mathcal{F}((a),(b))}(\frac{d'}{d'_b}(F)))$ also satisfies $\mathrm{P}(E', k)$. Finally, by recurrence hypothesis, for $k' < k$, any derivative of an expression $\mathbb{F}_{k'}(G')$ satisfies $\mathrm{P}(G', k')$. Consequently, any derivative of $\mathbb{F}_{k-\mathcal{F}((\varepsilon),(b))}(\frac{d'}{d'_b}(F))$ satisfies $\mathrm{P}(\frac{d'}{d'_b}(F), k - \mathcal{F}((\varepsilon),(b)))$ if $\mathcal{F}((\varepsilon),(b)) \neq 0$. Since $F$ is a derivative of $E'$, so is $\frac{d'}{d'_b}(F)$, and since $k - \mathcal{F}((\varepsilon),(b)) < k$, any derivative of $\mathbb{F}_{k-\mathcal{F}((\varepsilon),(b))}(\frac{d'}{d'_b}(F))$ satisfies $\mathrm{P}(E', k)$. As a consequence, **(Fact 2)** any derivative of $E$ w.r.t. a symbol $a$ satisfies $\mathrm{P}(E', k)$.

Let us show now that if an expression $H$ satisfies $\mathrm{P}(E', k)$, then any symbol derivative of $H$ also satisfies $\mathrm{P}(E', k)$. Since $H$ is a sum of expressions of type $\mathbb{F}_{k'}(G')$ where $k' \leq k$ and $G'$ is a derivative of $E'$, any symbol derivative $H'$ of $H$ is the sum of the derivatives of the expressions $H$ is the sum of. According to **Fact 2**, any symbol derivative of an expression $\mathbb{F}_{k'}(G')$ satisfies $\mathrm{P}(G', k')$. Since $G'$ is a derivative of $E'$ and $k' \leq k$, any expression satisfying $\mathrm{P}(G', k')$ also satisfies $\mathrm{P}(E', k)$. As a consequence, any derivative of $E$ w.r.t. a word $w$ in $\Sigma^*$ satisfies $\mathrm{P}(E', k)$.

As a conclusion, since any derivative of $E$ is a sum of expressions all belonging to the finite set $\{\mathbb{F}_{k'}(G) \mid k' \leq k \wedge G \in \mathcal{D}_{E'}\}$, using the ACI-equivalence, $\mathcal{D}_E$ is a finite set of AREs. Moreover, by induction over $E'$ and by recurrence over $k$, since any derivative of an expression $F$ in $X(E')$ belongs to the finite set of derivatives of $E'$ the computation of which halts, and since $\mathcal{F}((\varepsilon),(b)) \neq 0$ implies that $k - \mathcal{F}((\varepsilon),(b)) < k$, the computation of $\mathcal{D}_E$ halts. $\square$

**Lemma 8.** *Let* $E = \mathbb{F}_k(E')$ *be an ARE over an alphabet* $\Sigma$ *and* $a$ *be a symbol in* $\Sigma$*. Then* $L(\frac{d'}{d'_a}(E)) = a^{-1}(L(E))$*.*

**Lemma 9.** *Let $E = \mathbb{F}_k(E')$ be an ARE over an alphabet $\Sigma$ and $a$ be a symbol in $\Sigma$. Let $\mathcal{W}_{\mathcal{F}} = (\bigcup_{b\in\Sigma,\mathcal{F}((\varepsilon),(b))=0}\{b\})^*$ and $X(E') = \{E'\} \cup \bigcup_{w\in\mathcal{W}_{\mathcal{F}}} \frac{d'}{d'_w}(E')$. Then the two following propositions are equivalent:*

-$\varepsilon \in L(E)$
-$k \neq \bot \wedge \varepsilon \in \bigcup_{F\in X(E')} L(F) \cup L(\sum_{F\in X(E'),b\in\Sigma,\mathcal{F}((\varepsilon),(b))\neq 0}(\mathbb{F}_{k-\mathcal{F}((\varepsilon),(b))}(\frac{d'}{d'_b}(F))))$
*Furthermore, this equivalence defines a membership test that halts.*

Lemma 7 ensures that the derivative automaton $D$ of an ARE $E$, computed from the set $\mathcal{D}_E$ of dissimilar derivatives of $E$ following the classical way, is a finite recognizer. Lemma 9 ensures that the set of final states can be computed, since the number of derivatives is finite. Finally, Lemma 8 ensures that the DFA $D$ recognizes $L(E)$.

*Example 10.* Let $F = b^*(a+b)c^*$ and $E = \mathbb{H}_1(F)$ be an ARE over $\Sigma = \{a,b,c\}$ where $\mathbb{H}$ is the Hamming distance. Derivatives of $E$ are the following expressions:

$$\begin{array}{ll}
\frac{d'}{d'_a}(E) = \mathbb{H}_0(F) + \mathbb{H}_1(c^*) + \mathbb{H}_0(c^*) = E_1 & \frac{d'}{d'_a}(E_1) = \mathbb{H}_0(c^*) = E_4 \\
\frac{d'}{d'_b}(E) = E + \mathbb{H}_1(c^*) + \mathbb{H}_0(c^*) = E_2 & \frac{d'}{d'_b}(E_1) = \mathbb{H}_0(F) + \mathbb{H}_0(c^*) = E_3 \\
\frac{d'}{d'_c}(E) = \mathbb{H}_0(F) + \mathbb{H}_0(c^*) = E_3 & \frac{d'}{d'_c}(E_1) = \mathbb{H}_1(c^*) + \mathbb{H}_0(c^*) = E_5 \\
\frac{d'}{d'_a}(E_2) = \mathbb{H}_0(F) + \mathbb{H}_1(c^*) + \mathbb{H}_0(c^*) = E_1 & \frac{d'}{d'_a}(E_3) = \mathbb{H}_0(c^*) = E_4 \\
\frac{d'}{d'_b}(E_2) = E + \mathbb{H}_1(c^*) + \mathbb{H}_0(c^*) = E_2 & \frac{d'}{d'_b}(E_3) = \mathbb{H}_0(F) + \mathbb{H}_0(c^*) = E_3 \\
\frac{d'}{d'_c}(E_2) = \mathbb{H}_0(F) + \mathbb{H}_0(c^*) + \mathbb{H}_1(c^*) = E_1 & \frac{d'}{d'_c}(E_3) = \mathbb{H}_0(c^*) = E_4 \\
\frac{d'}{d'_a}(E_4) = \emptyset & \frac{d'}{d'_a}(E_5) = \mathbb{H}_0(c^*) = E_4 \\
\frac{d'}{d'_b}(E_4) = \emptyset & \frac{d'}{d'_b}(E_5) = \mathbb{H}_0(c^*) = E_4 \\
\frac{d'}{d'_c}(E_4) = \mathbb{H}_0(c^*) = E_4 & \frac{d'}{d'_c}(E_5) = \mathbb{H}_1(c^*) + \mathbb{H}_0(c^*) = E_5
\end{array}$$

The derivative automaton of $E$ is given Figure 3.

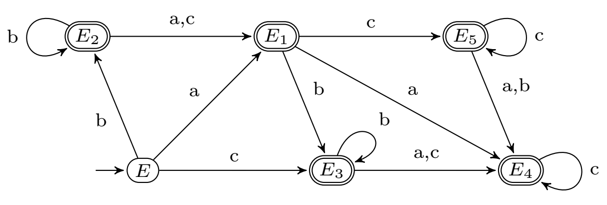


**Fig. 3.** The derivative automaton of $E = \mathbb{H}_1(b^*(a+b)c^*)$

## 4.3 Antimirov Derivatives

The derivative computation can be rewritten using sets of expressions in order to compute a nondeterministic recognizer from an ARE using partial derivatives. For convenience, let us set for $\mathcal{E}$ a set of expressions $\mathbb{F}_k(\mathcal{E}) = \bigcup_{E\in\mathcal{E}} \mathbb{F}_k(E)$ and $L(\mathcal{E}) = \bigcup_{E\in\mathcal{E}} L(E)$.

**Definition 11.** *Let $E = \mathbb{F}_k(E')$ be an ARE over an alphabet $\Sigma$ where $\mathbb{F}$ is associated with $\mathcal{F}$ and $a$ be a symbol in $\Sigma$. The* partial derivative *of $E$ w.r.t. $a$ is the set $\frac{\partial}{\partial_a}(E)$ computed as follows:*

$$\frac{\partial}{\partial_a}(E) = \begin{cases} \bigcup_{F\in X(E'),b\in\Sigma}(\mathbb{F}_{k-\mathcal{F}((a),(b))}(\frac{\partial}{\partial_b}(F))) \cup \bigcup_{F\in X(E')} \mathbb{F}_{k-\mathcal{F}((a),(\varepsilon))}(F) \\ \cup \ \frac{\partial}{\partial_a}(\bigcup_{F\in X(E'),b\in\Sigma,\mathcal{F}((\varepsilon),(b))\neq 0}(\mathbb{F}_{k-\mathcal{F}((\varepsilon),(b))}(\frac{\partial}{\partial_b}(F)))) \end{cases},$$

$$\text{where } \mathcal{W}_{\mathcal{F}} = (\textstyle\bigcup_{b\in\Sigma,\mathcal{F}((\varepsilon),(b))=0}\{b\})^* \text{ and } X(E') = \{E'\}\cup\bigcup_{w\in\mathcal{W}_{\mathcal{F}}}\frac{\partial}{\partial_w}(E').$$

**Lemma 12.** *Let $E = \mathbb{F}_k(E')$ be an ARE over an alphabet $\Sigma$ and $a$ be a symbol in $\Sigma$. Then $L(\frac{\partial}{\partial_a}(E)) = a^{-1}(L(E))$.*

Let $\mathcal{T}_E$ be the set of derivated terms of an ARE $E$ w.r.t. the words in $\Sigma^*$, that is the set of the elements of all the partial derivatives of $E$.

**Lemma 13.** *Let $E = \mathbb{F}_k(E')$ be an ARE over an alphabet $\Sigma$. Then:*

$$\mathcal{T}_E \subset \textstyle\bigcup_{k'\in\{0,\ldots,k\}}\mathbb{F}_{k'}(\mathcal{T}_{E'}).$$

*Moreover, the computation of $\mathcal{T}_E$ halts.*

**Corollary 14.** *Let $E = \mathbb{F}_k(E')$ be an ARE over an alphabet $\Sigma$. Then $\mathcal{T}_E$ is a finite set of AREs. Furthermore,* $\mathrm{Card}(\mathcal{T}_E) \leq \mathrm{Card}(\mathcal{T}_{E'}) \times (k+1)$.

**Lemma 15.** *Let $E = \mathbb{F}_k(E')$ be an ARE over an alphabet $\Sigma$ and $a$ be a symbol in $\Sigma$. Let $\mathcal{W}_{\mathcal{F}} = (\bigcup_{b\in\Sigma,\mathcal{F}((\varepsilon),(b))=0}\{b\})^*$ and $X(E') = \{E'\}\cup\bigcup_{w\in\mathcal{W}_{\mathcal{F}}}\frac{\partial}{\partial_w}(E')$. Then the two following propositions are equivalent:*

-$\varepsilon \in L(E)$,
-$k \neq \bot \wedge \varepsilon \in \bigcup_{F\in X(E')} L(F) \cup L(\bigcup_{F\in X(E'),b\in\Sigma,\mathcal{F}((\varepsilon),(b))\neq 0}(\mathbb{F}_{k-\mathcal{F}((\varepsilon),(b))}(\frac{\partial}{\partial_b}(F))))$

*Furthermore, this equivalence defines a membership test that halts.*

Corollary 14 ensures that the derivated term automaton $D'$ of an ARE $E$, computed from the set $\mathcal{T}_E$ of derivated terms of $E$ following the classical way, is a finite recognizer. Lemma 15 ensures that the set of final states can be computed. Finally, Lemma 12 ensures that the NFA $D'$ recognizes $L(E)$.

*Example 16.* Let $E$ be the ARE defined in Example 10. Partial derivatives of $E$ are the following sets of expressions:

$$\begin{array}{r||l}
\frac{\partial}{\partial_a}(E) = \{\mathbb{H}_0(F), \mathbb{H}_1(c^*), \mathbb{H}_0(c^*)\} & \frac{\partial}{\partial_a}(\mathbb{H}_1(c^*)) = \{\mathbb{H}_0(c^*)\} \\
\frac{\partial}{\partial_b}(E) = \{E, \mathbb{H}_1(c^*), \mathbb{H}_0(c^*)\} & \frac{\partial}{\partial_b}(\mathbb{H}_1(c^*)) = \{\mathbb{H}_0(c^*)\} \\
\frac{\partial}{\partial_c}(E) = \{\mathbb{H}_0(F), \mathbb{H}_0(c^*)\} & \frac{\partial}{\partial_c}(\mathbb{H}_1(c^*)) = \{\mathbb{H}_1(c^*)\} \\
\frac{\partial}{\partial_a}(\mathbb{H}_0(F)) = \{\mathbb{H}_0(c^*)\} & \frac{\partial}{\partial_a}(\mathbb{H}_0(c^*)) = \emptyset \\
\frac{\partial}{\partial_b}(\mathbb{H}_0(F)) = \{\mathbb{H}_0(F), \mathbb{H}_0(c^*)\} & \frac{\partial}{\partial_b}(\mathbb{H}_0(c^*)) = \emptyset \\
\frac{\partial}{\partial_c}(\mathbb{H}_0(F)) = \emptyset & \frac{\partial}{\partial_c}(\mathbb{H}_0(c^*)) = \{\mathbb{H}_0(c^*)\}
\end{array}$$

The derivated term automaton of $E$ is given Figure 4.

$$\begin{tikzcd}
& \mathbb{H}_0(F) \arrow[loop left, "b"] \arrow[rr, "{a,b}"] & & \mathbb{H}_0(c^*) \arrow[loop right, "c"] \\
{} \arrow[r] & \mathbb{H}_1(F) \arrow[loop above, "b"] \arrow[u, "{a,c}"] \arrow[urr, "{a,b,c}"] \arrow[rr, "{a,b}"'] & & \mathbb{H}_1(c^*) \arrow[u, "{a,b}"'] \arrow[loop right, "c"]
\end{tikzcd}$$

**Fig. 4.** The derivated term automaton of $E = \mathbb{H}_1(b^*(a+b)c^*)$

## 5 Conclusion

The similarity operators that equip the family of approximate regular expressions enhance the expressiveness of expressions and make AREs to be a nice tool to deal with approximate regular expression matching. From the computation of the derivatives of such an operator, it is possible to convert an ARE into a finite automaton. An additional advantage of similarity operators is that they can be combined with other regular operators, such as intersection and complementation operators [3], in order to produce even smaller expressions.

## References

1. Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. Theoret. Comput. Sci. 155, 291–319 (1996)
2. Brzozowski, J.A.: Derivatives of regular expressions. J. Assoc. Comput. Mach. 11(4), 481–494 (1964)
3. Caron, P., Champarnaud, J.-M., Mignot, L.: Partial Derivatives of an Extended Regular Expression. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 179–191. Springer, Heidelberg (2011)
4. Crochemore, M., Lecroq, T.: Text searching and indexing. In: Recent Advances in Formal Languages and Applications. SCI, vol. 25, pp. 43–80. Springer, Heidelberg (2006)
5. El-Mabrouk, N.: Recherche approchée de motifs - Application à des séquences biologiques structurées. Ph.D. thesis, LITP, Université Paris 7, France (1996)
6. Holub, J.: Dynamic Programming - NFA Simulation. In: Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2002. LNCS, vol. 2608, pp. 295–300. Springer, Heidelberg (2003)
7. Jokinen, P., Tarhio, J., Ukkonen, E.: A comparison of approximate string matching algorithms. Softw., Pract. Exper. 26(12), 1439–1458 (1996)
8. Kleene, S.: Representation of events in nerve nets and finite automata. Automata Studies Ann. Math. Studies 34, 3–41 (1956)
9. Muzátko, P.: Approximate regular expression matching. In: Stringology, pp. 37–41. Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University (1996)
10. Myers, E., Miller, W.: Approximate matching of regular expressions. Bulletin of Mathematical Biology 51, 5–37 (1989), http://dx.doi.org/10.1007/BF02458834, doi:10.1007/BF02458834

11. Myhill, J.: Finite automata and the representation of events. WADD TR-57-624, 112–137 (1957)
12. Navarro, G.: A guided tour to approximate string matching. ACM Computing Surveys 33(1), 31–88 (2001)
13. Navarro, G.: Approximate regular expression matching. In: Kao, M.-Y. (ed.) Encyclopedia of Algorithms, pp. 46–48. Springer, Heidelberg (2008)
14. Nerode, A.: Linear automata transformation. Proceedings of AMS 9, 541–544 (1958)
15. Schulz, K.U., Mihov, S.: Fast string correction with levenshtein automata. IJDAR 5(1), 67–85 (2002)
16. Ukkonen, E., Wood, D.: Approximate string matching with suffix automata. Algorithmica 10, 353–364 (1993), http://dx.doi.org/10.1007/BF01769703, doi:10.1007/BF01769703

# Catalytic Petri Nets Are Turing Complete*

Gabriel Ciobanu[1] and G. Michele Pinna[2]

[1] Institute of Computer Science, Romanian Academy, Iaşi, Romania
[2] Dipartimento di Matematica e Informatica, Università di Cagliari, Italy

**Abstract.** In this paper we introduce a class of Petri nets, called *catalytic Petri nets*, and a suitable firing strategy where transitions are fired only when they use tokens from specific places, called catalytic places. By establishing a one-to-one relationship with *catalytic membrane systems*, we can prove that the class of catalytic Petri nets with at least two catalytic places is Turing complete.

## 1 Introduction

Soon after their introduction in the early 60's, Petri nets have been acknowledged as a formalism for modeling distributed and concurrent computations and, from a formal language theoretic point of view, their *expressivity* has been investigated. On the one side, the classes of languages defined by Petri nets, i.e., sets of sequences of labeled or unlabeled transitions, have been studied from the beginning of the 70's (*e.g.* in [15] and references therein) and the research on this topic is still in progress. On another side the question of "how *expressive* Petri nets are" has been asked and the answer can be summarized as follows: Petri nets under the usual step firing strategy are not Turing complete (see [18] and the surveys [10,9] for a comprehensive knowledge), while they are Turing complete under suitable assumptions on the firing strategy, either *maximality* (all the possible transitions are fired together) or *ordering* (a transition should fire as soon as it is enabled), as shown in [3] and [4].

In order to make Petri nets Turing complete, different approaches have been taken; notably the one which *extends* the kind of arcs considered. The classic extension is the one where *inhibitor* arcs are considered, *i.e.*, arcs where the absence of tokens in certain places is modeled ([14]), or *reset* arcs, *i.e.*, arcs with the characteristic of emptying the preset, regardless of the number of tokens present in the place ([8]). Other extensions of Petri Nets, not necessarily always Turing complete, allow the introduction of *non blocking* arcs or *transfer* arcs ([11] and [13]) or making the transitions *marking* dependent ([5]).

Summing up, to make Petri nets Turing complete either suitable extensions of the model have to be considered or quite *heavy* assumptions on the firing rule have to be made. In the former case many among the main features of Petri nets are retained, *e.g.*, the *locality* of firing or the *distributed* state, whereas in

* Work partially supported by the Regione Autonoma della Sardegna; grants L.R. 7/2007 CRP2-120 (Project TESLA) and CRP-17285 (Project TRICS).

the latter the characteristics that have made Petri nets an appealing and useful model for concurrent and distributed computations are lost.

In this paper, inspired by *membrane computing*, we investigate the expressivity of Petri nets looking at suitable firing strategy and at the *structure* of the nets itself. The connection between membrane computing and Petri nets has already been established (see chapter 15 of the Oxford Handbook of Membrane Computing [21] and references therein), where it is shown that to each kind of membrane systems it is possible to associate a suitable *labeled* Petri net, where the labeling of transitions is used to model the membrane structure.

We establish a relation between Petri nets and membrane systems showing first that to each Petri net a membrane systems can be associated and vice versa, and then that various firing strategy definable on Petri nets are matched by corresponding evolutions steps in membrane systems. In particular we focus on membrane systems with only one membrane and on *catalytic* membrane systems, where the rules may *use* a *catalyst*, *i.e.*, an object needed for applying the rule but that it is never been *consumed*[1]. Then we use this relationship to establish the Turing completeness of a suitable class of Petri nets (intuitively the one that is translated into a catalytic membrane system) under a suitable firing strategy which establish that a particular (proper) subset of the enabled transitions are fired, namely the subset where the transitions involving catalytic places are used. We use results of membrane computing to prove that the number of catalytic places needed for Turing completeness is quite limited, namely just two.

The merit of the results presented in this paper is that they show that expressiveness of Petri nets can be increased without introducing suitable arcs, without requiring transitions fired upon a complete snapshot of the system, and therefore without loosing the locality of firing that is one of the main features of Petri nets. In other word the state is still *distributed*. Furthermore the relation established between Petri nets and membrane systems suggests that also other classes of Petri nets with *minimal parallelism* (*i.e.*, where a minimal number of concurrent transitions are considered) could be Turing complete.

The paper is organized as follows: in the next section we will recall the basic notions on Petri nets and their firing strategies, in section 3 we briefly recall the definition of (catalytic) membrane system and establish some expressiveness results. In section 4 we recall how to relate membrane systems to Petri nets and introduce the vice versa as well, and in section 5 we show that catalytic Petri nets with two catalysts are Turing complete when considering suitable firing strategies (which are not the maximal one, or based on ordered firing of transitions).

## 2 Petri Nets and Firing Strategies

*Notations.* With $\mathbb{N}$ we denote the set of *natural numbers* including zero, and with $\mathbb{N}^+$ the set of positive natural numbers.

[1] This notion is quite different from the one of *read* arc on Petri nets.

*Multisets.* Given a set $S$, a *multiset* over $S$ is a function $m : S \to \mathbb{N}$ and with $\partial S$ we denote the set of multisets of $S$. The *multiplicity* of an element $s$ in $m$ is given by $m(s)$. A multiset $m$ over $S$ is *finite* iff the set $dom(m) = \{s \in S \mid m(s) \neq 0\}$ is finite. All the multisets we consider in this paper are finite. A multiset $m$ such that $dom(m) = \emptyset$ is called *empty* and it is denoted by $\mathbf{0}$. The cardinality of a multiset is defined as $\#(m) = \sum_{s\in S} m(s)$. Given a multiset in $\partial S$ and a subset $\hat{S}$ of $S$, with $m|_{\hat{S}}$ we indicate the multiset over $\hat{S}$ such that $m|_{\hat{S}}(s) = m(s)$. We write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$, and $m \subset m'$ if $m \subseteq m'$ and $m \neq m'$. The operator $\oplus$ denotes *multiset union*: $m \oplus m'(s) = m(s) + m'(s)$. The operator $\ominus$ denotes *multiset difference*: $m \ominus m'(s) =$ if $m(s) > m'(s)$ then $m(s) - m'(s)$ else 0. The *scalar product* of a number $j$ with a multiset $m$ is $(j \cdot m)(s) = j \cdot (m(s))$. Sometimes a multiset $m \in \partial S$ is written as $\oplus_{s\in S} n_s \cdot s$, where $n_s = m(s)$; we omit the summands where $n_s$ is equal 0. If $m \in \partial S$, we denote by $[\![m]\!]$ the multiset defined as $[\![m]\!](a) = 1$ if $m(a) > 0$, and $[\![m]\!](a) = 0$ otherwise; sometimes $[\![m]\!]$ is identified (used interchangeably) with the corresponding subset $\{a \in S \mid [\![m]\!](a) = 1\}$ of $S$ (observe that this is different from $dom(m)$: this one is the set of elements on $S$ such that $m(s) > 0$, whereas $[\![m]\!]$ is a particular multiset associated to $m$).

*Petri Nets.* A Petri net is a tuple $N = ((S, T, F, m_0), \mathfrak{S})$ where $S$ is a set of *places*, $T$ is a set of *transitions*, $F : (S \times T) \cup (T \times S) \to \mathbb{N}$ is a *flow relation*, $m_0 \in \partial S$ is the initial marking and $\mathfrak{S} \subseteq S$ is a set of *final* places. Furthermore $S \cap T = \emptyset$. With ${}^\bullet x$ ($x^\bullet$, respectively) we indicate the multiset $F(\_, x)$ ($F(x, \_)$, respectively), and they are called the *preset* (*postset*, respectively) of $x$. We assume that for each transition $t$, $dom({}^\bullet t) \neq \emptyset$.

Given a net $N = ((S, T, F, m_0), \mathfrak{S})$, we say that $N$ is a *state machine* iff $\forall t \in T$. $|dom({}^\bullet t)| = |dom(t^\bullet)| = 1$; and $N$ is an *input state machine* iff $\forall t \in T$. $|dom({}^\bullet t)| = 1$ and ${}^\bullet t = [\![{}^\bullet t]\!]$. Let $S' \subseteq S$ be a subset of places, the *subnet* of $N$ generated by $S'$ is the net defined as follows: $N@S' = ((S', T@S', F@S', m_0@S'), \mathfrak{S} \cap S')$ where $T@S' = \{t \in T \mid \exists s \in S'$ such that either $F(s, t) > 0$ or $F(t, s) > 0\}$, $F@S'$ is the restriction of $F$ to $S'$, and $m_0@S' = m_0|_{S'}$.

*Firing strategies.* We discuss now the *dynamic* of Petri nets. A multiset over $S$ is called a marking for the net. Let $m \in \partial S$ be a marking of a net. A multiset $U \in \partial T$ of transitions is *enabled* under $m$ if for all $s \in S$ $\sum_{t\in T} U(t) \cdot F(s, t) \leq m(s)$, and it is written as $m\,[U\rangle_{st}$. If a finite multiset $U \in \partial T$ is *enabled* at a marking $m$ (i.e., $m\,[U\rangle_{st}$), then $U$ may *fire* reaching a new marking $m'$ defined as $m'(s) = m(s) + \sum_{t\in T} U(t) \cdot (F(t, s) - F(s, t))$, for all $s \in S$. We write $m\,[U\rangle_{st}\, m'$, and call $U$ a *step*. We observe that if $t \in dom(U)$ is such that either ${}^\bullet t = [\![{}^\bullet t]\!]$ or $t^\bullet = [\![t^\bullet]\!]$, then the firing of the transition $t$ consumes one token from each place in $dom({}^\bullet t)$ or produces one token in each place in $dom(t^\bullet)$. When considering steps, we often omit the subscript *st*, thus we write simply $m\,[U\rangle$ and $m\,[U\rangle\, m'$.

A *step firing sequence* is defined as follows: $m_0$ is a step firing sequence, and if $m_0\,[U_1\rangle\, m_1 \ldots m_{n-1}\,[U_n\rangle\, m_n$ is a step firing sequence and $m_n\,[U_{n+1}\rangle\, m_{n+1}$, then also $m_0\,[U_1\rangle\, m_1 \ldots m_{n-1}\,[U_n\rangle\, m_n\,[U_{n+1}\rangle\, m_{n+1}$ is a step firing sequence. Given a net $N = ((S.T, F, m_0), \mathfrak{S})$, a marking $m$ is *reachable* if there is a step firing sequence leading to it, *i.e.*, $m_0\,[U_1\rangle\, m_1 \ldots m_{n-1}\,[U_n\rangle\, m_n$ with $m = m_n$. The set of reachable marking of the net $N$ is denoted with $\mathfrak{M}(N)$. Given a Petri net $N$, it has been shown that the problem of deciding whether a given marking $m$ is reachable from the initial one (*i.e.*, if $m \in \mathfrak{M}(N)$) is decidable (see [10] for a survey).

The *ordinary* firing strategy of Petri nets (just one enabled transition is fired at each marking, regardless of how many are *simultaneously* enabled) is an instance of the step firing strategy we have previously revised: in this case the multiset $U$ is such that $U = [\![U]\!]$ and $\#(dom(U)) = 1$. We observe that, given a step $m\,[U\rangle\, m'$, it may be always *linearized* and $m'$ can be reached from $m$ with a possibly different step firing sequence.

Steps, and hence firing strategies, may have suitable characteristics. Here we concentrate on two of these, namely *maximality* and *place-awareness*. A step $U$ enabled at a marking $m$ is *maximal* iff each step $U'$ such that $U \subset U'$ is such that $\neg\ (m\,[U'\rangle\ )$. A maximal step will be denoted with $m\,[U\rangle_{max}\, m'$, and a maximal step firing sequence is a step firing sequence where each step is maximal. The set of reachable marking of a net $N$ with maximal step firing sequences is $\mathfrak{M}_{max}(N)$. In this case it holds that $\mathfrak{M}_{max}(N) \subseteq \mathfrak{M}(N)$ and the containment may be proper (indeed it often is, as reachability in the case of this firing strategy is undecidable [4]). Place-awareness concerns the requirement that suitable places are involved, if possible. More formally, consider a subset of places $\mathcal{S} \subseteq S$. A step $U$ enabled at a marking $m$ is $\mathcal{S}$-enabled iff for all $s \in \mathcal{S}$ either there exists a $t \in dom(s^\bullet)$ and $U(t) \neq 0$, or for all $t' \in dom(s^\bullet)$ it holds that $\neg m\,[t'\rangle$ . We write $m\,[U\rangle_{\mathcal{S}}$ to indicate this, and the corresponding step called $\mathcal{S}$-step is denoted by $m\,[U\rangle_{\mathcal{S}}\, m'$. Firing sequences where each step is a $\mathcal{S}$-step are defined as usual, and the set of reachable markings under this firing strategy is $\mathfrak{M}_{\mathcal{S}}(N)$. To ease the notation we use a subscript to indicate which firing strategy is used: *step*, *maximal*, $\mathcal{S}$-*step*; when no subscript is used, we assume that it is the step firing strategy.

Given a net $N$ and a marking $m$ of $N = ((S, T, F, m_0), \mathfrak{S})$, we say that $m$ is a *final* marking iff $m \in \mathfrak{M}(N)$ and for all $t \in T$ we have $\neg(m\,[t\rangle\ )$. We are interested in the reachable markings (under a firing strategy $fs \in \{step, max, \mathcal{S}\}$) that are also final: $\mathfrak{F}_{fs}(N) = \{m \in \mathfrak{M}_{fs}(N) \mid m \text{ is final}\}$. Among the final markings, we may be interested in considering either the numbers of tokens contained in certain places, or the marking in these places. Let $((S, T, F, m_0), \mathfrak{S})$ be a Petri net; we denote by $\mathfrak{F}^{\mathfrak{S}}_{fs}(N)$ and $\mathfrak{F}^{\#\mathfrak{S}}_{fs}(N)$ the sets $\{m|_{\mathfrak{S}} \mid m \in \mathfrak{F}_{fs}(N)\}$ and $\{\#(m|_{\mathfrak{S}}) \mid m \in \mathfrak{F}_{fs}(N)\}$, respectively.

## 3 Membrane Systems with Catalysts

Membrane systems (also called P systems) represent abstract computing models based on operations with multisets. They are inspired by the structure of (eukaryotic) cells, namely by the role of their membranes delimiting different compartments, and thus can help to understand information processes in the nature. The most comprehensive recent monograph is the Oxford Handbook of Membrane Computing [21]. Membrane computing was introduced in [20], volume [7] presents various applications, while the P Systems webpage presents recent developments.

Starting from the observation that there is an obvious parallelism in the cell biochemistry [2], and relying on the assumption that "if we wait enough, then all reactions which may take place will take place", a basic feature of the P systems is the maximally parallel way of using the rules: in each step, in each region of a system, we have to use a maximal multiset of rules. This maximal parallelism decreases the non-determinism of the systems evolution, and it provides enough power to get computational Turing completeness.

There are several types of membrane systems (defined by various ingredients inspired from cell biology); we use one of the basic types, namely catalytic P systems. A catalytic P system is formed by a hierarchical membrane structure with uniquely labeled membranes, the whole structure being embedded in a single skin membrane. Each membrane contains a multiset of abstract objects (members of a finite alphabet) which is divided into two parts: the set $C$ of catalysts, and the set of non-catalytic objects. Each membrane has assigned an initial multiset of objects and a fixed set of evolution rules of two possible types: catalytic rules $ca \to cv$, and non-cooperative rules $a \to v$, where $c$ is a catalyst object, $a$ is a non-catalyst object, and $v$ is a (possibly empty) multiset of objects. A catalytic P system is called *purely* catalytic if it contains only catalytic rules. The rules are (usually) applied in the maximally parallel mode: at each computational step and in each membrane, the selected multiset of applicable rules must be maximal, i.e., unused objects do not allow to apply an additional rule.

We formalize now the notion of *flat* P system and of *flat* catalytic P system. It is shown in [1] that the flattened version of a transition P system has the same computability power as a non flattened one; thus we can use the flattened one without losing generality. The idea behind the notion of flattening is the following one: the membrane structure of a P system, capturing the intuition that a rule uses objects in specific compartment and produces objects in the same compartment or in compartments that are *close* to this one (*i.e.*, either in the one surrounding it or compartments that are immediately internal to the specific one).

**Definition 1.** *A* (flat) P system *is the 4-tuple* $\Pi_f = (O, w^0, R, O')$ *where*

- $O$ *is a finite set of* objects*, and* $O' \subseteq O$ *are the* final *objects,*
- $w^0 \in \partial V$ *is a finite multiset of objects, called the* initial *configuration, and*
- $R$ *is a finite set of rules of the form* $r = u \to v$*, with* $u, v \in \partial O$ *and* $u \neq \mathbf{0}$*.*

*A* configuration *of a membrane system is any finite multiset of objects.*

*A flat membrane system* $\Pi_f$ *is called* catalytic *iff there is a designated subset* $O_C \subset O$ *of* catalysts *and the rules have the following form: either* $r = a \to v$ *with* $a \in O \backslash O_C$ *and* $v \in \partial(O \backslash O_C)$ *or* $r = ca \to cv$ *with* $a \in O \backslash O_C$*,* $v \in \partial(O \backslash O_C)$ *and* $c \in O_C$*. If all the rules are of the form* $r = ca \to cv$ *we say that the catalytic P system is* purely *catalytic. We denote catalytic P systems with* $C\Pi$*, and purely catalytic ones with* $CP\Pi$*.*

According to [1], any property proved for flat membranes could be also proved for non-flat ones.

Given a rule $r = u \to v$, with $lhs(r)$ we denote the multiset $u$ and with $rhs(r)$ the multiset $v$. The dynamic of a P system is formalized in the following way: given a multiset of rules $\mathcal{R}$ ($\mathcal{R} \in \partial R$) and a configuration $w$, $w \overset{\mathcal{R}}{\Rightarrow} w'$ iff $\oplus_{r\in R}\mathcal{R}(r) \cdot lhs(r) \subseteq w$ and $w' = w \ominus (\oplus_{r\in R}\mathcal{R}(r) \cdot lhs(r)) \oplus (\oplus_{r\in R}\mathcal{R}(r) \cdot rhs(r))$. The transition $w \overset{\mathcal{R}}{\Rightarrow} w'$ is called an evolution step. A configuration $w$ is reachable from the initial one iff there exists a finite sequence of evolution steps $w_1 \overset{\mathcal{R}_1}{\Rightarrow} w_1 \cdots w_n \overset{\mathcal{R}_r}{\Rightarrow} w_{n+1}$ such that $w_1 = w^0$ and $w = w_{n+1}$. The basic assumption is that each step $w \overset{\mathcal{R}}{\Rightarrow} w'$ is *maximal*, namely for each $\mathcal{R}' \supseteq \mathcal{R}$ it holds that $\oplus_{r\in R}\mathcal{R}'(r) \cdot lhs(r) \not\subseteq w$.

Computations are sequence of evolution steps, and a computation terminates (halts) iff at the configuration $w$ reached by this computation the only possible evolution step is $\mathbf{0}$, *i.e.*, $w \overset{\mathbf{0}}{\Rightarrow} w$. The result of the computation is $w|_{O'}$. The sets *calculated* by a flat P system $\Pi_f$ are $NO_{max}(\Pi) = \{\#(w|_{O'}) \mid$ there exists a halting computation (with maximal parallelism) and $w$ is the reached configuration$\}$ and $PsO_{max}(\Pi) = \{w|_{O'} \mid$ there exists a halting computation (with maximal parallelism) and $w$ is the reached configuration$\}$. The class *calculated* by a suitable class of P systems will be denoted with $NO_{max}$ and $PsO_{max}$, respectively, overloading the notation.

As we recalled previously, in [1] it is shown that the flattening preserves the computability power and expressiveness of P systems. With $\Pi^n$ we denote the membrane system with $n$ membranes, $n \geq 1$. Consider a membrane system $\Pi^n$ (not a flat one), we call $flat(\Pi^n)$ its flattened version; flattening a catalytic P system gives a flat catalytic P system.

A research topic in membrane computing is to find more realistic P systems from a biological point of view, and one target in this respect is to relax the condition of using the rules in a maximally parallel way. *Minimal* parallelism was introduced in [6]; it describes another way of applying the rules: if at least a rule from a set of rules associated with a membrane (or a region) *can* be used, then at least one rule from that membrane (or region) *must* be used, without any other restriction (e.g., more rules can be used, but we do not care how many). Even if it might look weak, this minimal parallelism still leads to Turing completeness for a suitable class of P systems, called symport/antiport P systems. The *minimal parallelism* stems out from the consideration that this way of using the rules ensures that all compartments (or regions) of the system evolve in parallel by using at least one rule, whenever such a rule is applicable. Considering the class of flat catalytic P systems, Turing completeness may be achieved relaxing the maximal parallelism requirement with the weaker one stating that for each catalyst $c$ at least one catalytic rule $r = ca \to cv$ is used, if possible. Thus the question is how many catalysts are needed to obtain Turing completeness. With $RE$ we denote the set of recursively enumerable sets of numbers and with $PsRE$ the set of recursively enumerable sets of Parikh vectors. Finally with $(C\Pi, n)$ we indicate the number of catalysts in the catalytic P system $C\Pi$.

A catalytic P system with a single membrane and only two catalysts has the power of a Turing machine, i.e., it can generate all the computable enumerable sets of (vectors of) natural numbers [12]. With $([p-]cat, n)$ we denote the class of flat (purely) catalytic P system with at least $n$ catalysts. Assuming that the computations obey to the *minimal* parallel rule (*i.e.*, at least one rule involving each catalyst is used, if possible), the following theorems present the results used in Section 5.

**Theorem 2.** $NO_{min}((cat, n)) = RE$ *and* $PsO_{min}((cat, n)) = PsRE$*, for* $n \geq 2$.

Three catalysts are needed in the case of purely catalytic systems.

**Theorem 3.** $NO_{min}((p - cat, n)) = RE$ *and* $PsO_{min}((p - cat, n)) = PsRE$*, for* $n \geq 3$.

We end this part recalling that chapter 4 of the handbook provides a good survey of the computability power of catalytic P systems ([21]). The above results are presented in that chapter, together with other interesting results.

## 4 Relating Petri Nets and Membrane Systems

We describe here how to associate a P system to a Petri net, and vice-versa.

*From membrane systems to Petri nets.* The intuition of encoding a membrane system into a Petri nets is exactly the same of the encoding presented in [17] and further developed thereof (see [16] and references therein): to each object of the membrane system a place is associated, and to each rule belonging to a membrane system a transition is associated. Such a translation from membrane systems to Petri nets (with localities) can be applied to (almost) any kind of membrane systems, including catalytic membrane systems where the objects are partitioned into two subsets and the rules comply with a specific pattern,

**Definition 4.** *Starting from a membrane system* $\Pi = (O, \mu, w^0, R, O')$*, we can associate to it a net structure* $\mathcal{F}(\Pi) = ((S, T, F, m), \mathfrak{S})$ *such that*

- $S = O$ *and* $T = \{t^r \mid r \in R\}$,
- *for all transitions* $t = t^r \in T$ *and all the places* $s \in S$ *with* $r = u \to v$*, we define*

$$F(s,t) = \begin{cases} u(a) & \text{if } s = a \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad F(t,s) = \begin{cases} v(a) & \text{if } s = a \\ 0 & \text{otherwise} \end{cases}$$

- $m(s) = \begin{cases} w^0(a) & \text{if } s = a \\ 0 & \text{otherwise} \end{cases}$, *and*
- $\mathfrak{S} = O'$.

The multiplicity of an object $v$ is modeled by the number of tokens in the place $v$. Obviously, this construction leads to a Petri net.

Following [17], we introduce two functions: one associating to a configuration of a membrane system a marking of its corresponding net, and another associating to the rules applied in an evolution of the membrane system a step in the net.

**Definition 5.** *Let $\Pi = (O, \mu, w^0, R, O')$ be a membrane system, and $\mathcal{F}(\Pi) = ((S, T, F, m), \mathfrak{S})$ be the associated Petri net. Let $w$ be a configuration of $\Pi$. Then $\nu(w)$ is the marking defined by $\nu(w) = w$.*

**Definition 6.** *Let $\Pi = (O, \mu, w^0, R, O')$ be a membrane system, and $\mathcal{F}(\Pi) = ((S, T, F, m), \mathfrak{S})$ be the associated Petri net. Let $C \xRightarrow{\boldsymbol{R}} C'$ be an evolution step of $\Pi$. Then $\sigma(\boldsymbol{R})$ is the multiset defined by $\sigma(\boldsymbol{R})(t^j) = \boldsymbol{R}(r^j)$, for all $t^j \in T$.*

We can now state the main result contained in [17].

**Theorem 7.** *Let $\Pi = (O, \mu, w^0, R, O')$ be a membrane system, and $\mathcal{F}(\Pi) = ((S, T, F, m), \mathfrak{S})$ be the associated Petri net. $w \xRightarrow{\boldsymbol{R}}_{fs} w'$ iff $\nu(w)\, [\sigma(\boldsymbol{R})\rangle_{fs}\, \nu(w')$ with $fs \in \{step, max, \mathcal{S}\}$.*

This theorem essentially says that the net associated to a membrane system behaves in the same way, provided that in the net steps are performed accordingly to the same way the evolution steps in the membrane system are defined.

From the previous section we know that the sets of natural numbers or of multisets *calculated* by a membrane system $\Pi$ under a suitable evolution strategy $e$ are denoted with $\mathcal{N}_e(\Pi)$ and $\mathcal{P}s_e(\Pi)$, respectively. A consequence of Theorem 7 is the following one:

**Theorem 8.** *Let $\Pi = (O, \mu, w^0, R, O')$ be a membrane system, and $\mathcal{F}(\Pi) = ((S, T, F, m), \mathfrak{S})$ be the associated Petri net. Then $\mathfrak{F}^{\mathfrak{S}}_{fs}(\mathcal{F}(\Pi)) = \mathcal{P}s_{fs}(\Pi)$ and $\mathfrak{F}^{\mathfrak{S}}_{fs}(\mathcal{F}(\Pi)) = \mathcal{N}_{fs}(\Pi)$, with $fs \in \{step, max, \mathcal{S}\}$.*

Thus the function calculated by the net associated to the membrane system is the same (again provided that evolutions are applied following the chosen firing sequence/evolution strategy).

*From Petri nets to membrane systems.* Given a Petri net, it is always possible to associate a *flat* membrane system to it. In the translation from membrane systems to Petri nets we have associated to each object a place. Now we do vice versa: to each place we associate an object. As for the purpose of this paper we are interested in *what* is calculated by a net, we explicitly point out the set of final places of a net.

**Definition 9.** *Let $N = ((S, T, F, m_0), \mathfrak{S})$ be a Petri net.*
*Then $\mathcal{K}(N) = (O, \mu, w^0, R, O')$ is the membrane system defined as follows:*

- *$O = S$ and $w^0(s) = m(s)$,*
- *for all $t \in T$ define a rule $r^t = u \to v$ in $R$ where $u = {}^\bullet t$ and $v = t^\bullet$, and*
- *$O' = \mathfrak{S}$.*

Moreover, we define two mappings $\xi$ and $\eta$ which associate the objects to the places and the rules to the transitions, respectively. Following Definition 9, we have $\xi(s) = s$, and $\eta(t)$ is the rule $r_i^t$ with $\mathcal{L}(t) = i$.

The first result is that $\mathcal{K}(N)$ is indeed a flat membrane system.

**Proposition 10.** *Let $N = ((S, T, F, m_0), \mathfrak{S})$ be a Petri net. Then $\mathcal{K}(N)$ is a flat membrane system.*

Moreover, the net associated to $\mathcal{K}(N)$ is indeed $N$.

**Proposition 11.** *Let $N = ((S, T, F, m_0), \mathfrak{S})$ be a Petri net. Then $\mathcal{F}(\mathcal{K}(N))=N$.*

When we consider flat membrane systems, we also have the following result.

**Proposition 12.** *Let $\Pi = (O, \mu, w^0, R, O')$ be a flat membrane system. Then $\mathcal{K}(\mathcal{F}(\Pi)) = \Pi$.*

We extend the two functions $\xi$ and $\eta$ to markings and multisets of transitions. These extensions relate markings to configurations, and steps in nets to evolution steps in membrane systems.

**Definition 13.** *Let $N = ((S, T, F, m_0), \mathfrak{S})$ be a Petri net and $\mathcal{K}(N)$ be its associate membrane system. Let $m$ be a marking of $N$. Then $\xi(m)$ is the configuration defined by $\xi(m)(s) = m(s)$, for all $s \in S$.*

**Definition 14.** *Let $N = ((S, T, F, m_0), \mathfrak{S})$ be a Petri net and $\mathcal{K}(N)$ be the associate membrane system. Let $m\,[U\rangle\, m'$ be a step of $N$. Then $\eta(U)$ is the evolution step defined by $\eta(U)(r^t) = U(t)$, for all $r^t \in R$.*

We are now ready to present the main result of this section.

**Theorem 15.** *Let $N = ((S, T, F, m_0), \mathfrak{S})$ be a Petri net and $\mathcal{K}(N)$ be its associate membrane system. For $fs \in \{step, max, \mathcal{S}\}$, we have*

$$m\,[U\rangle_{fs}\, m'' \quad \textit{iff} \quad \xi(m) \overset{\eta(U)}{\Longrightarrow}_{fs} \xi(m').$$

A consequence of Theorem 15 is the following one:

**Theorem 16.** *Let $N = ((S, T, F, m_0), \mathfrak{S})$ be a Petri net and $\mathcal{K}(N)$ be the associated flat membrane system. Then*

$$\mathfrak{F}^{\mathfrak{S}}_{fs}(\mathcal{F}(\Pi)) = \mathcal{P}s_{fs}(\Pi) \quad \textit{and} \quad \mathfrak{F}^{\mathfrak{S}}_{fs}(\mathcal{F}(\Pi)) = \mathcal{N}_{fs}(\Pi),$$

*where $fs \in \{step, max, \mathcal{S}\}$.*

## 5 Catalytic Petri Nets Are Turing Complete

In this section we present the main result of the paper, namely that there is a suitable class of Petri Nets (namely catalytic Petri Nets with at least two catalysts) which is able to calculate a recursive enumerable set under a suitable firing strategy. We first introduce the class of catalytic nets, then show the relations between catalytic nets and catalytic membrane system, and finally use the results of Section 3 to show that catalytic nets are Turing complete.

*Catalytic Petri nets.* The main motivation behind catalytic net is to *control* the firing of suitable transitions *locally* by putting tokens in suitable places (that maintain the same number of tokens during the whole execution).

**Definition 17.** *Let $N = ((S, T, F, m), \mathfrak{S})$ be a Petri net. $N$ is* catalytic *iff the set of places $S$ is partitioned into two disjoint sets $\mathcal{C}$ and $\mathcal{V}$ such that*

1. *the subnet $N@\mathcal{V}$ is an input state machine, and*
2. *the subnet of $N@\mathcal{C}$ is a state machine, and for all $t \in T@\mathcal{C}$ we have ${}^\bullet t = t^\bullet$ and $\#({}^\bullet t) = 1$.*

*A net $N = ((S, T, F, m), \mathfrak{S})$ is said to be* purely catalytic *iff $T = T@\mathcal{C}$.*

Places in a catalytic net are partitioned into two subsets, the catalytic places ($\mathcal{C}$) and the non catalytic ones ($S \setminus \mathcal{C}$). Condition 1 states that each transition may consume tokens from only one non catalytic place, whereas condition 2 says that each transition may use at most one token from a catalytic place, and the used token is made again available for further use.

Catalytic Petri nets $N = ((S, T, F, m), \mathfrak{S})$ are abbreviated by CPN. Sometime we explicitly indicate the set of catalytic places and their number, and so a catalytic net is presented as $N = ((S, T, F, m, \mathcal{C}, n), \mathfrak{S})$ with $|\mathcal{C}| = n$.

The intuition behind catalytic Petri nets is the following: a transition $t$ which uses a catalyst fires only if there is a token in the catalytic places associated to the transition. Tokens in catalytic places may be consumed/produced by transitions, but the transition using these tokens as catalysts do leave the token in the place. It is worth to stress the difference among catalytic nets and Petri nets with read arcs of Montanari and Rossi ([19]): in their case, if two transitions test for the presence of a token in place (with read arcs) it is enough to have a token in that place, whereas in catalytic nets if two transitions use the *same* catalytic place, this place must contain enough tokens for both transitions.

We *specialize* again the notion of enabling. Let $N = ((S, T, F, m, \mathcal{C}, n), \mathfrak{S})$ be a CPN. The step $U$ is catalyticly enabled at the marking $m$ iff it is enabled at $m$ and $\forall c \in \mathcal{C}$ either there is a transition $t \in dom(U)$ such that ${}^\bullet t(c) \neq 0$ or $\forall t \in dom(c^\bullet)$ it holds that $\neg m\,[t\rangle$.

In other words, a step is catalyticly enabled whenever for each catalyst places in $\mathcal{C}$, either all the transitions using tokens from this catalyst place are not enabled or there is at least one transition using a token from a catalytic place. We write $m\,[U\rangle_{\mathcal{C}}$ to denote that $U$ is catalyticly enabled, and we denote by $m\,[U\rangle_{\mathcal{C}}\, m'$ the firing of a catalyticly enabled step, where $U$ is a catalytic step. A catalytic firing sequence is a step firing sequence where each step is a catalytic one. The set of reachable markings is defined accordingly, and denoted by $\mathfrak{M}_{\mathcal{C}}(N)$. In the same ways are defined the sets $\mathfrak{F}_{\mathcal{C}}(N)$, $\mathfrak{F}_{\mathcal{C}}^{\mathfrak{S}}(N)$ and $\mathfrak{F}_{\mathcal{C}}^{\#\mathfrak{S}}(N)$.

*Catalytic P systems and catalytic nets.* The results of Section 4 can be also used for catalytic P systems and catalytic nets. In particular, we can state the following two results.

**Proposition 18.** *Let $\Pi = (O, O_C, \mu, w^0, R, O')$ be a catalytic P system, and $\mathcal{F}(\Pi)$ be its associated structure. Then $\mathcal{F}(\Pi)$ is a catalytic net.*

**Proposition 19.** *Let $N = ((S, T, F, m_0, \mathcal{C}, n), \mathfrak{S})$ be a catalytic Petri net. Then $\mathcal{K}(N)$ is a flat catalytic P system.*

*Turing completeness under the catalytic firing strategy* We denote by $CPN(n)$ the class of catalytic Petri nets with $n$ catalytic places, and by $PCPN(n)$ the class of purely catalytic Petri nets with $n$ catalytic places. Then Theorems [2] and [3] can be specialized as follows:

**Theorem 20.**

$$\{\mathfrak{F}_{\mathcal{C}}^{\#\mathfrak{S}}(N) \mid N \in CPN(n)\} = RE \quad \textit{and}$$
$$\{\mathfrak{F}_{\mathcal{C}}^{\mathfrak{S}}(N) \mid N \in CPN(n)\} = PsRE, \textit{ for } n \geq 2.$$

**Theorem 21.**

$$\{\mathfrak{F}_{\mathcal{C}}^{\#\mathfrak{S}}(N) \mid N \in PCPN(n)\} = RE \quad \textit{and}$$
$$\{\mathfrak{F}_{\mathcal{C}}^{\mathfrak{S}}(N) \mid N \in PCPN(n)\} = PsRE \textit{ for } n \geq 3.$$

## 6 Conclusion and Future Work

In this paper we have presented a new class of Petri nets called *catalytic Petri nets*. This class turns out to be Turing complete by using a suitable firing strategy. The firing of a step is done by first checking, for each catalytic place, if one of the transitions using the place is enabled, and then firing at least one of these transitions. This means that the rule requires a *minimal* parallelism which is also locally confined. Establishing a connection with the membrane systems allows to state also the minimal number of catalytic places which are needed.

In this class of Petri nets the main characteristic of the basic models is retained: the firing is still local, though the notion of catalyst changes slightly the meaning of *local* because now a confined location to be checked depends on the catalyst and not only on the transition to be executed.

Other kinds of membrane systems enjoy minimal parallelism evolution steps, and the relations with other suitable classes of Petri nets have to be established. Furthermore we want to investigate further the class of catalytic Petri nets, which seems to be an interesting and promising class.

**Acknowledgments.** Part of this work has been done while the first author was Visiting Professor at the Dipartimento di Matematica e Informatica of the Università di Cagliari, with a grant of the Regione Autonoma della Sardegna.

## References

1. Agrigoroaiei, O., Ciobanu, G.: Flattening the Transition P Systems with Dissolution. In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds.) CMC 2010. LNCS, vol. 6501, pp. 53–64. Springer, Heidelberg (2010)
2. Alberts, B., Johnson, A., Walter, P., Lewis, J.: Molecular Biology of the Cell. Garland Publishing, New York (2002)

3. Burkhard, H.D.: On Priorities of Parallelism. In: Salwicki, A. (ed.) Logic of Programs 1980. LNCS, vol. 148, pp. 86–97. Springer, Heidelberg (1983)
4. Burkhard, H.D.: Ordered firing in Petri nets. Elektronische Informationsverarbeitung und Kybernetik 17(2/3), 71–86 (1981)
5. Ciardo, G.: Petri Nets with Marking-Dependent Arc Cardinality: Properties and Analysis. In: Valette, R. (ed.) ICATPN 1994. LNCS, vol. 815, pp. 179–198. Springer, Heidelberg (1994)
6. Ciobanu, G., Pan, L., Păun, G., Pérez-Jiménez, M.J.: P systems with minimal parallelism. Theoretical Computer Science 378(1), 117–130 (2007)
7. Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (eds.): Applications of Membrane Computing. Natural Computing Series. Springer, Heidelberg (2006)
8. Dufourd, C., Finkel, A., Schnoebelen, P.: Reset Nets Between Decidability and Undecidability. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 103–115. Springer, Heidelberg (1998)
9. Esparza, J.: Decidability and complexity of Petri net problems - an introduction. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 374–428. Springer, Heidelberg (1998)
10. Esparza, J., Nielsen, M.: Decidability issues for Petri nets - a survey. Bulletin of the EATCS 52, 244–262 (1994)
11. Finkel, A., Geeraerts, G., Raskin, J.F., Begin, L.V.: On the $\omega$-language expressive power of extended Petri nets. Theor. Comput. Sci. 356(3), 374–386 (2006)
12. Freund, R., Kari, L., Oswald, M., Sosík, P.: Computationally universal P systems without priorities: two catalysts are sufficient. Theoretical Computer Science 330(2), 251–266 (2005)
13. Geeraerts, G.: On the expressive power of Petri nets with transfer arcs vs. Petri nets with reset arcs. Tech. Rep. 572, Université Libre de Bruxelles (2007)
14. Hack, M.: Decidability Questions for Petri Nets. In: Outstanding Dissertations in the Computer Sciences. Garland Publishing, New York (1975)
15. Jantzen, M., Zetzsche, G.: Labeled Step Sequences in Petri Nets. In: van Hee, K.M., Valk, R. (eds.) PETRI NETS 2008. LNCS, vol. 5062, pp. 270–287. Springer, Heidelberg (2008)
16. Kleijn, J., Koutny, M.: Petri nets and membrane computing. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 389–412. Oxford University Press (2010)
17. Kleijn, J., Koutny, M., Rozenberg, G.: Towards a Petri Net Semantics for Membrane Systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 292–309. Springer, Heidelberg (2006)
18. Kosaraju, S.R.: Decidability of reachability in vector addition systems (preliminary version). In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, pp. 267–281. ACM (1982)
19. Montanari, U., Rossi, F.: Contextual nets. Acta Informatica 32(6) (1995)
20. Păun, G.: Membrane Computing. An Introduction. Natural Computing Series. Springer, Heidelberg (2002)
21. Păun, G., Rozernberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press (2010)

# Computational Complexity of Rule Distributions of Non-uniform Cellular Automata

Alberto Dennunzio[2], Enrico Formenti[1,⋆], and Julien Provillard[1]

[1] Université Nice-Sophia Antipolis, Laboratoire I3S,
2000 Route des Colles, 06903 Sophia Antipolis, France
{enrico.formenti,julien.provillard}@unice.fr
[2] Università degli Studi di Milano–Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione,
Viale Sarca 336, 20126 Milano, Italy
dennunzio@disco.unimib.it

**Abstract.** $\nu$-CA are cellular automata which can have different local rules at each site of their lattice. Indeed, the spatial distribution of local rules completely characterizes $\nu$-CA. In this paper, sets of distributions sharing some interesting properties are associated with languages of bi-infinite words. The complexity classes of these languages are investigated providing an initial rough classification of $\nu$-CA.

## 1 Introduction

Cellular automata (CA) are discrete dynamical systems consisting in an infinite number of finite automata arranged on a regular lattice. All automata of the lattice are identical and work synchronously. The new state of each automaton is computed by a local rule on the basis of its current state and the one of a fixed set of neighboring automata. This simple definition contrasts the huge number of different dynamical behaviors that made the model widely used in many scientific disciplines for simulating phenomena characterized by the emergency of complex behaviors from simple local interactions (particle reaction-diffusion, pseudo-random number generation, cryptography, *etc.*), see for instance [4].

In many cases, the uniformity of the local rule is more a constraint than a helping feature. Indeed, the uniformity constraint has been relaxed, for example, for modeling cell colonies growth, fast pseudo-random number generation, and VLSI circuit design and testing. This gave rise to new models, called non-uniform cellular automata ($\nu$-CA) or hybrid cellular automata (HCA), in which the local rule of the finite automaton at a given site depends on its position. If the study of dynamical behavior has just started up [3,6], applications and analysis of structural properties have already produced a wide literature (see [10,11]).

In this paper, we adopt a formal languages complexity point of view. Consider a finite set $\mathcal{R}$ of local rules defined over the same finite state set $A$. A (one-dimensional) $\nu$-CA is essentially defined by the *distribution* or assignment of local

⋆ Corresponding author.

rules in $\mathcal{R}$ to sites of the lattice. Whenever $\mathcal{R}$ contains a single rule, the standard cellular automata model is obtained. Therefore, each $\nu$-CA can be associated with a unique bi-infinite word over $\mathcal{R}$. Consider now the class $C$ of $\nu$-CA defined over $\mathcal{R}$ and sharing a certain property $P$ (for example surjectivity, injectivity, *etc.*). Clearly, $C$ can be identified as a set of bi-infinite words contained in ${}^{\omega}\mathcal{R}^{\omega}$. In this paper, we analyze the language complexity of $C$ *w.r.t.* several well-known properties, namely number-conservation, surjectivity, injectivity, sensitivity to initial conditions and equicontinuity. We have proved that $C$ is a subshift of finite type and sofic, respectively, for the first two properties, while it is $\zeta$-rational for the last three properties in the list. Remark that for sensitivity to initial conditions and equicontinuity, the results are proved when $\mathcal{R}$ contains only linear local rules (*i.e.* local rules satisfying a certain additivity property) with radius 1. The general case seems very complicated and it is still open.

In order to prove the main theorems, some auxiliary results, notions and constructions have been introduced (variants of De Bruijn graphs and their products, *etc.*). We believe that they can be interesting in their own to prove further properties.

In the paper, for lack of space most of proofs have been removed. They will appear in the long version of the paper. They can also be found at [5].

## 2 Notations and Definitions

For all $i, j \in \mathbb{Z}$ with $i \leq j$ (resp. $i < j$), let $[i, j] = \{i, i+1, \ldots, j\}$ (resp. $[i, j) = \{i, \ldots, j-1\}$).

**Configurations and non-uniform automata.** Let $A$ be a finite *alphabet*. A *configuration* or *bi-infinite word* is a function from $\mathbb{Z}$ to $A$. For any configuration $x$ and any integer $i$, $x_i$ denotes the element of $x$ at index $i$. The *configuration set* $A^{\mathbb{Z}}$ is usually equipped with the metric $d$ defined as follows

$$\forall x, y \in A^{\mathbb{Z}},\ d(x, y) = 2^{-n},\ \text{where}\ n = \min\{i \geq 0\ :\ x_i \neq y_i\ \text{or}\ x_{-i} \neq y_{-i}\}\ .$$

The metric $d$ induces the Cantor topology on $A^{\mathbb{Z}}$. For any pair $i, j \in \mathbb{Z}$, with $i \leq j$, and any configuration $x \in A^{\mathbb{Z}}$ we denote by $x_{[i,j]}$ the word $w = x_i \ldots x_j \in A^{j-i+1}$, *i.e.*, the portion of $x$ inside $[i, j]$, and we say that the word $w$ appears in $x$. Similarly, $u_{[i,j]} = u_i \ldots u_j$ is the portion of a word $u \in A^l$ inside $[i, j]$ (here, $i, j \in [0, l)$). In both the previous notations, $[i, j]$ can be replaced by $[i, j)$ with the obvious meaning. For any word $u \in A^*$, $|u|$ denotes its length. With $0 \in A$, a configuration $x$ is said to be finite if the number of positions $i$ at which $x_i \neq 0$ is finite.

A *local rule* of radius $r \in \mathbb{N}$ on the alphabet $A$ is a map from $A^{2r+1}$ to $A$. Local rules are crucial in both the definitions of cellular automata and non-uniform cellular automata. A function $F : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ is a *cellular automaton* (CA) if there exist $r \in \mathbb{N}$ and a local rule $f$ of radius $r$ such that

$$\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \quad F(x)_i = f(x_{[i-r,i+r]})\ .$$

The *shift* map $\sigma : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ defined as $\sigma(x)_i = x_{i+1}, \forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}$ is one among the simplest examples of CA.

Let $\mathcal{R}$ be a set of local rules on $A$. A *distribution* on $\mathcal{R}$ is a function $\theta$ from $\mathbb{Z}$ to $\mathcal{R}$, *i.e.*, a bi-infinite word on $\mathcal{R}$. Denote by $\Theta$ the set of all distributions on $\mathcal{R}$. A *non-uniform cellular automaton* ($\nu$-CA) is a triple $(A, \theta, (r_i)_{i\in\mathbb{N}})$ where $A$ is an alphabet, $\theta$ a distribution on the set of all possible local rules on $A$ and $r_i$ is the radius of $\theta_i$. A $\nu$-CA defines a global transition function $H_\theta : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ by

$$\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \quad H_\theta(x)_i = \theta_i(x_{[i-r_i, i+r_i]}) \enspace .$$

In the sequel, when no misunderstanding is possible, we will identify a $\nu$-CA with its global transition function. From [3], recall that a function $H : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ is the global transition function of a $\nu$-CA if and only if it is continuous. In this paper, we will consider distributions on a finite set of local rules. In that case, one can assume without loss of generality that there exists an integer $r$ such that all the rules in $\mathcal{R}$ have the same radius $r$. $\nu$-CA constructed on such finite sets of local rules are called r$\nu$-CA (of radius $r$).

A *finite distribution* is a word $\psi \in \mathcal{R}^n$, *i.e.*, a sequence of $n$ rules of $\mathcal{R}$. Each finite distribution $\psi$ defines a function $h_\psi : A^{n+2r} \to A^n$ by

$$\forall u \in A^{n+2r}, \forall i \in [0, n), \quad h_\psi(u)_i = \psi_i(u_{[i,i+2r]}) \enspace .$$

These functions are called partial transition functions since they express the behavior of a $\nu$-CA on a finite set of sites: if $\theta$ is a distribution and $i \leq j$ are integers, then

$$\forall x \in A^{\mathbb{Z}}, \quad H_\theta(x)_{[i,j]} = h_{\theta_{[i,j]}}(x_{[i-r,j+r]}) \enspace .$$

**Languages.** Recall that a *language* is any set $\mathcal{L} \subseteq A^*$ and a *finite state automaton* is a tuple $\mathcal{A} = (Q, A, T, I, F)$, where $Q$ is a finite set of states, $A$ is the alphabet, $T \subseteq Q \times A \times Q$ is the set of *transitions*, and $I, F \subseteq Q$ are the sets of *initial* and *final* states, respectively. A *path* $p$ in $\mathcal{A}$ is a finite sequence $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} q_n$ visiting the states $q_0, \ldots, q_n$ and with label $a_0 \ldots a_{n-1}$ such that $(q_i, a_i, q_{i+1}) \in T$ for each $i \in [0, n)$. A path is *successful* if $q_0 \in I$ and $q_n \in F$. The language $\mathcal{L}(\mathcal{A})$ of an automaton $\mathcal{A}$ is the set of the labels of all successful paths in $\mathcal{A}$. A language $\mathcal{L}$ is *rational* if there exists a finite automaton $\mathcal{A}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{A})$.

A bi-infinite language is any subset of $A^{\mathbb{Z}}$. Let $\mathcal{A} = (Q, A, T, I, F)$ be a finite automaton. A bi-infinite path $p$ in $\mathcal{A}$ is a bi-infinite sequence $\ldots \xrightarrow{a_{-2}} q_{-1} \xrightarrow{a_{-1}} q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \ldots$ such that $(q_i, a_i, q_{i+1}) \in T$ for each $i \in \mathbb{Z}$. The bi-infinite word $\ldots a_{-1}a_0a_1 \ldots$ is the label of the bi-infinite path $p$. A bi-infinite path is *successful* if the sets $\{i \in \mathbb{N} : q_{-i} \in I\}$ and $\{i \in \mathbb{N} : q_i \in F\}$ are infinite. By similarity with the uniform case, we call this condition the *Büchi acceptance condition.* The bi-infinite language $\mathcal{L}^\zeta(\mathcal{A})$ of the automaton is the set of the labels of all successful bi-infinite paths in $\mathcal{A}$. A bi-infinite language $\mathcal{L}$ is *$\zeta$-rational* if there exists a finite automaton $\mathcal{A}$ such that $\mathcal{L} = \mathcal{L}^\zeta(\mathcal{A})$.

A bi-infinite language $X$ is a *subshift* if $X$ is (topologically) closed and $\sigma$–invariant, *i.e.*, $\sigma(X) = X$. For any $\mathcal{F} \subseteq A^*$ let $X_{\mathcal{F}}$ be the bi-infinite language of all bi-infinite words $x$ such that no word $u \in \mathcal{F}$ appears in $x$. A bi-infinite language $X$ is a subshift iff $X = X_{\mathcal{F}}$ for some $\mathcal{F} \subseteq A^*$. The set $\mathcal{F}$ is a set of *forbidden words* for $X$. A subshift $X$ is said to be a *subshift of finite type* (resp. *sofic*) iff $X = X_{\mathcal{F}}$ for some finite (resp. rational) $\mathcal{F}$.

For a more in deep introduction to the theory of formal languages, the reader can refer to [13] for rational languages,[16] for subshifts and [21,22] for $\zeta$-rational languages.

**Properties of non-uniform CA**. A $\nu$-CA is *sujective* (resp. *injective*) iff its defining map $H : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ is surjective (resp. injective). A $\nu$-CA $H$ is *equicontinuous* if $\forall \varepsilon > 0$ there exists $\delta > 0$ such that for all $x, y \in A^{\mathbb{Z}}$, $d(y, x) < \delta$ implies that $\forall n \in \mathbb{N}$, $d(H^n(y), H^n(x)) < \varepsilon$. A $\nu$-CA $H$ is *sensitive to the initial conditions* (or simply *sensitive*) if there exists a constant $\varepsilon > 0$ such that for any element $x \in A^{\mathbb{Z}}$ and any $\delta > 0$ there is a point $y \in A^{\mathbb{Z}}$ such that $d(y, x) < \delta$ and $d(H^n(y), H^n(x)) > \varepsilon$ for some $n \in \mathbb{N}$.

Given a finite set of local rules $\mathcal{R}$, a predicate $P$ over distributions is a function from $\Theta$ to $\{\perp, \top\}$, where $\perp, \top$ are the false and true symbols, respectively. In the sequel, we are interested in the complexity of the following language of bi-infinite words

$$\mathcal{L}_P = \{\theta \in \Theta \ : \ P(\theta) = \top\} \ .$$

## 3 Number Conservation

In physics, a lot of transformations are conservative: a certain quantity remains invariant during a whole experiment. Think to conservation laws of mass and energy for example. Both $CA$ and $\nu$-CA are used to represent phenomena from physics and it is therefore interesting to decide when they represent a conservative transformation. The case of uniform CA has been treated in [8], here we generalize those results to $\nu$-CA. Finally, we prove that the language of the set of distributions representing conservative r$\nu$-CA is a subshift of finite type.

In this section, without loss of generality, $A$ is $\{0, 1, \ldots, s-1\}$. Indeed, given any alphabet $A$, let $\phi : A \to \mathbb{N}$ be a morphism such that $0 \in \phi(A)$, then all the following results will hold by replacing $A$ by $\phi(A)$. Denote by $\underline{0}$ the configuration in which every element is 0. For all configuration $x \in A^{\mathbb{Z}}$, define the *partial charge* of $x$ between the index $-n$ and $n$ as $\mu_n(x) = \sum_{i=-n}^{n} x_i$ and the *global charge* of $x$ as $\mu(x) = \lim_{n\to\infty} \mu_n(x)$. Clearly $\mu(x) = \infty$, if $x$ is not a finite configuration.

**Definition 1 (FNC).** *A $\nu$-CA $H$ is* number-conserving on finite configurations *(FNC) if for all finite configurations $x$, $\mu(x) = \mu(H(x))$.*

Remark that if $H$ is FNC then $H(\underline{0}) = \underline{0}$ and, for any finite configuration $x$, $H(x)$ is a finite configuration.

**Definition 2 (NC).** *A $\nu$-CA $H$ is* number-conserving *(NC) if both the following conditions hold: 1) $H(\underline{0}) = \underline{0}$, 2) $\forall x \in A^{\mathbb{Z}} \setminus \{\underline{0}\}$, $\lim\limits_{n\to\infty} \frac{\mu_n(H(x))}{\mu_n(x)} = 1$.*

Remark that if $x \neq \underline{0}$ the fraction $\frac{\mu_n(H(x))}{\mu_n(x)}$ is well-defined for $n$ sufficiently large.

In the general case, a $\nu$-CA can be FNC without being NC. Indeed, consider the following example.

*Example 3.* Let $H : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ be the $\nu$-CA such that for all configuration $x$, for all integer $i$, $H(x)_{2i} = x_i$ and $H(x)_{2i+1} = 0$. Then $H$ is FNC but not NC. The configuration $\underline{1}$ in which every element is 1 verifies $\lim_{n\to\infty} \frac{\mu_n(H(\underline{1}))}{\mu_n(\underline{1})} = \frac{1}{2}$.

On the other hand, the following proposition shows that is not possible in the case of r$\nu$-CA.

**Proposition 4.** *Any r$\nu$-CA of radius $r$ is NC if and only if it is FNC.*

**Theorem 5.** *Let $\mathcal{R}$ be a finite set of local rules. Consider the predicate $P(\theta) =$ "$H_\theta$ is number conserving" over distributions $\theta \in \Theta$ on $\mathcal{R}$. Then, $\mathcal{L}_P$ is a subshift of finite type.*

*Proof.* We are going to prove that $\mathcal{L}_P = X_{\mathcal{F}}$ where $\mathcal{F}$ is the following set

$$\left\{\psi \in \mathcal{R}^{2r+1} : \exists u \in A^{2r+1}, \psi_{2r}(u) \neq u_0 + \sum_{i=0}^{2r-1} \psi_{i+1}(0^{2r-i}u_{[1,i+1]}) - \psi_i(0^{2r-i}u_{[0,i]})\right\}.$$

Assume that $\theta \in \mathcal{L}_P$ and let $j \in \mathbb{Z}$. For any $u \in A^{2r+1}$, let $x, y$ be two finite configurations such that $x_{[j-r,j+r]} = u$ and $y_{[j-r,j+r]} = 0u_{[1,2r]}$. As $H_\theta$ is NC, by Proposition 4, conditions $\mu(H(x)) = \mu(x)$ and $\mu(H(y)) = \mu(y)$ are true. So,

$$\sum_{i=0}^{2r} \theta_{j+i-2r}(0^{2r-i}u_{[0,i]}) + \sum_{i=1}^{2r} \theta_{j+i}(u_{[i,2r]}0^i) = \sum_{i=0}^{2r} u_i \ , \tag{1}$$

$$\sum_{i=1}^{2r} \theta_{j+i-2r}(0^{2r-i+1}u_{[1,i]}) + \sum_{i=1}^{2r} \theta_{j+i}(u_{[i,2r]}0^i) = \sum_{i=1}^{2r} u_i \ . \tag{2}$$

Subtracting (2) from (1), we obtain $\theta_j(u) = u_0 + \sum_{i=1}^{2r} \theta_{j+i-2r}(0^{2r-i+1}u_{[1,i]}) - \sum_{i=0}^{2r-1} \theta_{j+i-2r}(0^{2r-i}u_{[0,i]})$ which can be rewritten as

$$\theta_j(u) = u_0 + \sum_{i=0}^{2r-1} \theta_{j+i+1-2r}(0^{2r-i}u_{[1,i+1]}) - \theta_{j+i-2r}(0^{2r-i}u_{[0,i]}) \ .$$

Thus, for all $j \in \mathbb{Z}$, $\theta_{[j-2r,j]} \notin \mathcal{F}$, meaning that $\theta \in X_{\mathcal{F}}$. So, $\mathcal{L}_P \subseteq X_{\mathcal{F}}$

Suppose now that $\theta \in X_{\mathcal{F}}$, *i.e.*, for all integer $j$, $\theta_{[j-2r,j]} \notin \mathcal{F}$. Taking $u = 0^{2r+1}$, for all $j$ we have $\theta_{j+2r}(0^{2r+1}) = 0 + \sum_{i=0}^{2r-1} \theta_{j+i+1}(0^{2r+1}) - \theta_{j+i}(0^{2r+1})$ which leads to $\theta_j(0^{2r+1}) = 0$. For any finite configuration $x$, $\mu(H_\theta(x)) =$

$$\begin{aligned}
&= \sum_{j\in\mathbb{Z}} H_\theta(x)_j = \sum_{j\in\mathbb{Z}} \theta_j(x_{[j-r,j+r]}) \\
&= \sum_{j\in\mathbb{Z}} \left(x_j + \sum_{i=0}^{2r-1} \theta_{j+i+1-2r}(0^{2r-i}x_{[j-r+1,j-r+i+1]}) - \theta_{j+i-2r}(0^{2r-i}x_{[j-r,j-r+i]})\right) \\
&= \sum_{j\in\mathbb{Z}} x_j + \sum_{i=0}^{2r-1} \Big(\sum_{j\in\mathbb{Z}} \theta_{j+i+1-2r}(0^{2r-i}x_{[j-r+1,j-r+i+1]}) \\
&\qquad\qquad - \sum_{j\in\mathbb{Z}} \theta_{j+i-2r}(0^{2r-i}x_{[j-r,j-r+i]})\Big)
\end{aligned}$$

Since

$$\sum_{j\in\mathbb{Z}} \theta_{j+i+1-2r}(0^{2r-i}x_{[j-r+1,j-r+i+1]}) = \sum_{j\in\mathbb{Z}} \theta_{j+i-2r}(0^{2r-i}x_{[j-r,j-r+i]}) \ ,$$

we obtain $\mu(H_\theta(x)) = \sum_{j\in\mathbb{Z}} H_\theta(x)_j = \sum_{j\in\mathbb{Z}} x_j = \mu(x)$. Thus, $H_\theta$ is FNC and, by Proposition 4, NC. Hence, $\theta \in \mathcal{L}_P$. So, $X_{\mathcal{F}} \subseteq \mathcal{L}_P$. □

## 4 Surjectivity and Injectivity

In standard CA setting, injectivity is a fundamental property which is equivalent to reversibility [12]. It is well-known that it is decidable for one-dimensional CA and undecidable in higher dimensions [1,14]. Surjectivity is also a dimension sensitive property (*i.e.* decidable in dimension one and undecidable for higher dimensions) and it is a necessary condition for many types of chaotic behaviors.

In this paper, we prove that the language associated with distributions inducing surjective (resp. injective) $\nu$-CA is sofic (resp. $\zeta$-rational). Remark that constructions for surjectivity and injectivity are noticeably different, contrary to what happens for the classical CA when dealing with the decidability of those properties. Indeed in the uniform case, thanks to the Garden of Eden theorem [19,20], one construction is sufficient [23]. However the Garden of Eden theorem does not hold on $\nu$-CA [3], therefore distinct constructions are necessary.

Before proceeding to the main results of the section we need some technical lemma and new constructions. We believe that these constructions, inspired by [23], might be of interest in their own and could be of help for proving new results.

**Lemma 6.** *For any fixed $\theta \in \Theta$, the $\nu$-CA $H_\theta$ is surjective if and only if $h_{\theta_{[i,j]}}$ is surjective for all integers $i, j$ with $i \leq j$.*

**Definition 7.** *Let $\mathcal{R}$ be a finite set of rules of radius $r$. The* De Bruijn graph *of $\mathcal{R}$ is the labeled multi-edge graph $\mathcal{G} = (V, E)$, where $V = A^{2r}$ and edges in $E$ are all the pairs $(aw, wb)$ with label $(f, f(awb))$, obtained varying $a, b \in A$, $w \in A^{2r-1}$, and $f \in \mathcal{R}$.*

*Example 8.* Let $A = \{0, 1\}$ and consider the set $\mathcal{R} = \{\oplus, id\}$ where $\oplus$ and $id$ are the rules of radius 1 defined as $\forall x, y, z \in A$, $\oplus(x, y, z) = (x + z) \bmod 2$, and $id(x, y, z) = y$. The De Bruijn graph of $\mathcal{R}$ is the graph $\mathcal{G}$ in Figure 1.

Given two alphabets $A, B$ and a finite word $w = (a_0, b_0) \dots (a_n, b_n) \in (A \times B)^*$, the words $a = Pr_1(w)$, $b = Pr_2(w)$, are the projections of $w$ on $A$ and $B$, respectively. By abuse of notation, we will write $(a, b) \in (A \times B)^*$ instead of $w \in (A \times B)^*, a = Pr_1(w), b = Pr_2(w)$. The same holds for bi-infinite words.

**Lemma 9.** *Let $\mathcal{G}$ be the De Bruijn graph of a finite set of rules $\mathcal{R}$. Consider $\mathcal{G}$ as an automaton where all states are both initial and final. Then, $\mathcal{L}(\mathcal{G}) = \{(\psi, u) \in (\mathcal{R} \times A)^* : h_\psi^{-1}(u) \neq \emptyset\}$.*

**Theorem 10.** *Let $\mathcal{R}$ be a finite set of local rules. Consider the predicate $P(\theta) =$ "$H_\theta$ is surjective" over distributions $\theta \in \Theta$ on $\mathcal{R}$. Then $\mathcal{L}_P$ is a sofic subshift.*

*Proof.* Set $\mathcal{F} = \{\psi \in \mathcal{R}^* : h_\psi$ is not surjective$\}$. By Lemma 6, $\mathcal{L}_P$ is just the subshift $X_\mathcal{F}$. Consider the De Bruijn graph $\mathcal{G}$ of $\mathcal{R}$ as an automaton $\mathcal{A}$ where all states are both initial and final. By Lemma 9, $\mathcal{L}(\mathcal{A}) = \{(\psi, u) \in (\mathcal{R} \times A)^* : h_\psi^{-1}(u) \neq \emptyset\}$. Build now the automaton $\mathcal{A}^c$ that recognizes $\mathcal{L}^c = \{(\psi, u) \in (\mathcal{R} \times A)^* : h_\psi^{-1}(u) = \emptyset\}$. Remove from $\mathcal{A}^c$ all second components of edge labels

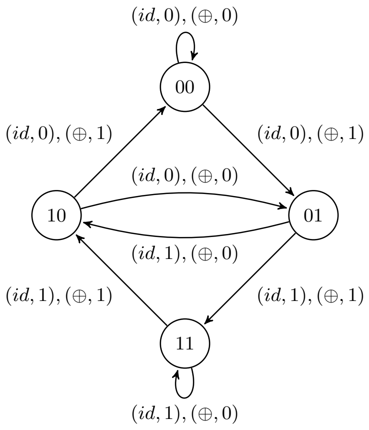


**Fig. 1.** De Bruijn graph of $\mathcal{R} = \{\oplus, id\}$ (every printed edge represents two edges of the graph, labels are concatenated)

and let $\tilde{\mathcal{A}}$ be the obtained automaton. A word $\psi \in \mathcal{R}^*$ is recognized by $\tilde{\mathcal{A}}$ if and only if there exists $u \in A^*$ such that $(\psi, u) \in \mathcal{L}^c$, *i.e.*, iff $h_\psi$ is not surjective. Thus, $\mathcal{L}(\tilde{\mathcal{A}}) = \mathcal{F}$ and $\mathcal{L}_P = X_\mathcal{F}$ is a sofic subshift. □

The proof of Theorem 10 provides an algorithm to build an automaton that recognizes the language $\mathcal{F}$ of the forbidden words for the sofic subshift $\mathcal{L}_P$ of distributions on a given finite set of rules $\mathcal{R}$. It consists of the following steps: 1) Build the De Bruijn graph $\mathcal{G}$ of $\mathcal{R}$; 2) Consider $\mathcal{G}$ as an automaton whose all states are both initial and final and determinize it to obtain the automaton $\mathcal{A}$; 3) Complete $\mathcal{A}$ if necessary and make all final states non-final and vice versa to obtain $\mathcal{A}^c$; 4) Delete all second components of edge labels of $\mathcal{A}^c$ to obtain $\tilde{\mathcal{A}}$.

*Example 11.* With the set $\mathcal{R}$ from the Example 8 as input, this algorithm gives the automaton in Figure 2. Thus, we deduce that $\mathcal{F} = \mathcal{R}^* id \oplus (\oplus\oplus)^* id \mathcal{R}^*$ and $\mathcal{L}_P$ is the well-known even subshift.

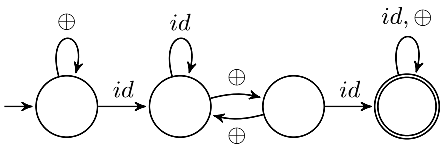


**Fig. 2.** The automaton $\tilde{\mathcal{A}}$ obtained from the set $\mathcal{R} = \{id, \oplus\}$

In the final part of this section, we will use product graphs to study the injectivity property. Those graphs were first defined to deal with the decidability of the ambiguity of finite automata in [2].

**Definition 12.** *Let $\mathcal{R}$ be a finite set of rules of radius $r$. Consider the De Bruijn graph $\mathcal{G} = (V, E)$ of $\mathcal{R}$. The* product graph *$\mathcal{P}$ of $\mathcal{R}$ is the labeled graph $(V \times V, W)$ where $((u, u'), (v, v')) \in W$ with label $(f, a) \in \mathcal{R} \times A$ if and only if $(u, v), (u', v') \in E$ both with the same label $(f, a)$.*

*Remark 13.* Any bi-infinite path in $\mathcal{P}$ with label $(\theta_i, z_i)_{i \in \mathbb{Z}} \in (\mathcal{R} \times A)^{\mathbb{Z}}$ corresponds to two bi-infinite paths in $\mathcal{G}$ in which the visited vertexes define two configurations $x$ and $y$ such that $H_\theta(x) = H_\theta(y) = z$.

We call *quick-fail acceptance condition* for bi-infinite paths in a finite automaton $\mathcal{A}$, the acceptance condition which accepts bi-infinite paths visiting at least once a final state. The set of labels of all such successful bi-infinite paths is said to be the language recognized by $\mathcal{A}$ under the quick-fail acceptance condition. We chose this terminology since the words belonging to the language recognized by $\mathcal{A}$ fail to induce injective $\nu$-CA (Theorem 15).

**Lemma 14.** *Let $\mathcal{A} = (Q, A, T, I, F)$ be a finite automaton. The bi-infinite language $\mathcal{L}$ recognized by $\mathcal{A}$ with the quick-fail acceptance condition is $\zeta$-rational.*

**Theorem 15.** *Let $\mathcal{R}$ be a finite set of local rules. Consider the predicate $P(\theta) =$ "$H_\theta$ is injective" over distributions $\theta \in \Theta$ on $\mathcal{R}$. Then, $\mathcal{L}_P$ is $\zeta$-rational.*

*Proof.* Let $\mathcal{P}$ be the product graph of $\mathcal{R}$. Consider now $\mathcal{P}$ as a finite automaton where all the states are initial and the final states are the pairs $(u, u')$ with $u \neq u'$. Remove from $\mathcal{P}$ all second components of edge labels and let $\tilde{\mathcal{P}}$ be the obtained automaton. Then, the language recognized by $\tilde{\mathcal{P}}$ with the quick-fail acceptance condition is $\mathcal{L}_P^c$. By Lemma 14, $\mathcal{L}_P^c$ is $\zeta$-rational and, therefore, $\mathcal{L}_P$ is $\zeta$-rational too. $\square$

## 5 Equicontinuity and Sensitivity for Linear r$\nu$-CA

Sensitivity to initial conditions is a widely known property indicating a possible chaotic behavior of a dynamical system and it is often popularized under the metaphor of *butterfly effect* [7]. At the opposite equicontinuity is an element of stability of a system. In this section, we are going to study these properties in the context of $\nu$-CA.

In the uniform case, equicontinuity points are characterized by blocking words [15]. Some extensions have been made in the case of $\nu$-CA [3]. In general, the problem of establishing if a CA admits a blocking word is undecidable [9] but, in the case of linear CA, the problem turns out to be decidable [18]. Moreover, the dichotomy theorem between sensitivity and presence of equicontinuity points [15] is not ensured in the context of $\nu$-CA. Therefore, in this preliminary study of the complexity of distributions, we preferred to focus on a sub-class in which the dichotomy theorem still holds, namely linear $\nu$-CA (Proposition 17).

In order to consider linear $\nu$-CA, the alphabet $A$ is endowed with a sum $(+)$ and a product $(\cdot)$ operation that make it a commutative ring and we denote by

0 and 1 the neutral elements of + and $\cdot$, respectively. Of course, $A^n$ and $A^{\mathbb{Z}}$ are also commutative rings where sum and product are defined component-wise and, with an abuse of notation, they will be denoted by the the same symbols.

**Definition 16.** *A local rule $f$ of radius $r$ is said to be* linear *if and only if there exists a word $\lambda \in A^{2r+1}$ such that $\forall u \in A^{2r+1}, f(u) = \sum_{i=0}^{2r} \lambda_i \cdot u_i$. A $\nu$-CA $H$ is said to be* linear *if it is defined by a distribution of linear local rules.*

**Proposition 17.** *Any linear $\nu$-CA $H$ is either sensitive or equicontinuous.*

From now on we consider finite sets $\mathcal{R}$ in which all rules are linear of radius $r$.

**Definition 18 (Wall).** *A* right-wall *is any element $\psi \in \mathcal{R}^*$ of length $n \geq r$ such that, for all word $v \in A^r$, the sequence $u_\psi(v) : \mathbb{N} \to A^n$ recursively defined by $u_\psi(v)_0 = 0^n$, $u_\psi(v)_1 = h_\psi(0^r u_\psi(v)_0 v)$, and $u_\psi(v)_{k+1} = h_\psi(0^r u_\psi(v)_k 0^r)$ for $k > 1$, verifies $\forall k \in \mathbb{N}, (u_\psi(v)_k)_{[0,r-1]} = 0^r$.* Left-walls *are defined similarly.*

Roughly speaking, the sequence $u_\psi(v)$ gives the dynamical evolution of the function $h_\psi$ when the leftmost and rightmost inputs are fixed.

**Lemma 19.** *For any right-wall $\psi \in \mathcal{R}^n$ and any $f \in \mathcal{R}$, both $f\psi$ and $\psi f$ are right-walls. Furthermore, if $\psi \in \mathcal{R}^*$ is a right-wall, then $\psi'\psi\psi''$ is a right-wall for any $\psi', \psi'' \in \mathcal{R}^*$. Similar results hold for left-walls.*

**Proposition 20.** *For any $\theta \in \Theta$, $H_\theta$ is sensitive if and only if one of the two following conditions holds: 1) There exists $n \in \mathbb{N}$ such that for all integer $m \geq n + r$, $\theta_{[n+1,m]}$ is not a right-wall; 2) There exists $n \in \mathbb{N}$ such that for all integer $m \leq -n - r$, $\theta_{[m,-n-1]}$ is not a left-wall.*

The characterization of walls for the general case is still under investigation. However, remark that, given any set of linear local rules with radius $r \geq 1$ on a ring $(A, +, \cdot)$, it is possible to transform it in a set of linear local rules with radius 1 on a non-commutative ring. In this case, Lemma 19 and Proposition 20 remain true whereas the characterisation of walls of the Proposition 21 does not hold anymore. For these reasons, in the remaining part of this section, we will assume that $\mathcal{R}$ is a finite set of linear rules of radius 1 (over a commutative ring). In this case, any rule $f \in \mathcal{R}$ will be expressed in the following form: $\forall a, b, c \in A, f(a, b, c) = \lambda_f^- \cdot a + \tilde{\lambda}_f \cdot b + \lambda_f^+ \cdot c$ for some $\lambda_f^-, \tilde{\lambda}_f, \lambda_f^+ \in A$.

**Proposition 21.** *A finite distribution $\psi \in \mathcal{R}^n$ is a right-wall (resp. a left-wall), if and only if $\prod_{i=0}^{n-1} \lambda_{\psi_i}^+ = 0$ (resp. $\prod_{i=0}^{n-1} \lambda_{\psi_i}^- = 0$).*

For any set $\mathcal{R}$ of linear rules of radius $r = 1$, an automaton $\mathcal{A} = (Q, Z, T, I, F)$ recognizing walls can be constructed. The alphabet $Z$ is $\mathcal{R}$, the set of states $Q$ is $\{-, +\} \times A$, $I = \{(-, 0)\}$, $F = \{(+, 0)\}$ and the transition rule $T$ is as follows

1. $((-, a), f, (-, \lambda_f^- \cdot a))$, $\forall a \in A \setminus \{0\}, \forall f \in \mathcal{R}$ (minimal left-wall detection).
2. $((-, 0), f, (-, 1))$, $\forall f \in \mathcal{R}$ (end of detection).
3. $((-, 1), f, (-, 1))$, $\forall f \in \mathcal{R}$ (waiting).

4. $((-,1),f,(+,1))$, $\forall f\in\mathcal{R}$ (transition from left part to right part).
5. $((+,1),f,(+,1))$, $\forall f\in\mathcal{R}$ (waiting).
6. $((+,1),f,(+,0))$, $\forall f\in\mathcal{R}$ (beginning of detection).
7. $((+,\lambda_f^+\cdot a),f,(+,a))$, $\forall a\in A\setminus\{0\},\forall f\in\mathcal{R}$ (minimal right-wall detection).

Practically speaking, $\mathcal{A}$ consists of two components, the left and the right part, with a non-deterministic transition from left to right. Each component has two special states: the first one (the state $(-,1)$ for the left part or $(+,1)$ for the right part) on which $\mathcal{A}$ loops waiting for the detection of a minimal (wrt the length) wall, the second one on which $\mathcal{A}$ starts ($(+,0)$ for the right part) or ends ($(-,0)$ for the left part) the detection of such a wall. The graph structure of $\mathcal{A}$ is schematized in Figure 3.

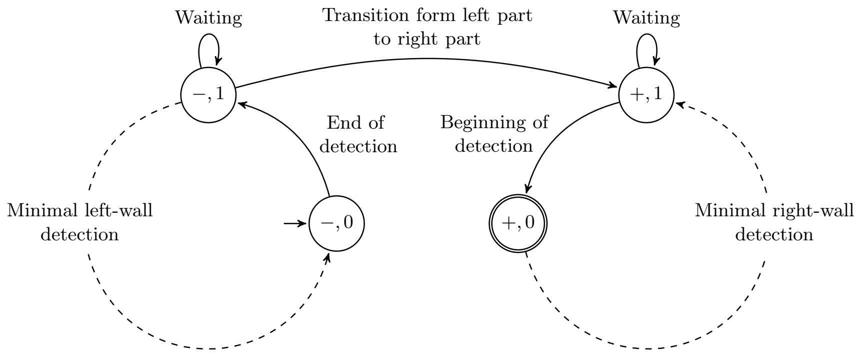

Waiting
Transition form left part to right part
Waiting
$-,1$
$+,1$
End of detection
Beginning of detection
Minimal left-wall detection
$-,0$
$+,0$
Minimal right-wall detection

**Fig. 3.** Conceptual structure of the automaton $\mathcal{A}$ for walls detection

**Theorem 22.** *Let $\mathcal{R}$ be a finite set of linear local rules of radius $r=1$. Consider the predicates $P_1(\theta)=$ "$H_\theta$ is equicontinuous" and $P_2(\theta)=$ "$H_\theta$ is sensitive" over distributions $\theta\in\Theta$ on $\mathcal{R}$. Then, both $\mathcal{L}_{P_1}$ and $\mathcal{L}_{P_2}$ are $\zeta$-rational.*

*Proof.* We are going to prove that $\mathcal{L}^\zeta(\mathcal{A})=\mathcal{L}_{P_1}$ where $\mathcal{A}$ is the automaton above introduced for the set $\mathcal{R}$ with Büchi acceptance condition. This permits to immediately state that $\mathcal{L}_{P_1}$ is $\zeta$-rational, and that, by Proposition 17, $\mathcal{L}_{P_2}$ is $\zeta$-rational too.

Let $\theta\in\mathcal{L}^\zeta(\mathcal{A})$. We show that for any $n\in\mathbb{N}$, there exists $m\le -n-1$ such that $\theta_{[m,-n-1]}$ is a left-wall. Let $n\in\mathbb{N}$. There is a successful path $p=\ldots\xrightarrow{\theta_{-1}}(s_0,a_0)\xrightarrow{\theta_0}(s_1,a_1)\ldots$ in $\mathcal{A}$ and integers $i,j$ with $i<j<-n$ such that $(s_i,a_i)=(s_j,a_j)=(-,0)$ are two successive initial states. If $m\in(i,j)$ is the greatest integer with $(s_m,a_m)=(-,1)$, the finite path $(s_m,a_m)\xrightarrow{\theta_m}(s_{m+1},a_{m+1})\xrightarrow{\theta_{m+1}}\ldots\xrightarrow{\theta_{j-1}}(s_j,a_j)$ is obtained by transitions of $\mathcal{A}$ from 1). Then, $0=a_j=a_m\cdot\prod_{l=m}^{j-1}\lambda_{\theta_l}^-$, and, by Proposition 21, $\theta_{[m,j-1]}$ is a left-wall. By Lemma 19, $\theta_{[m,-n-1]}$ is a left-wall too. Similarly, it holds that for any $n\in\mathbb{N}$, there exists

$m \geq n+1$ such that $\theta_{[n+1,m]}$ is a right-wall. Hence, by Propositions 20, $H_\theta$ is equicontinuous, *i.e.*, $\theta \in \mathcal{L}_{P_1}$.

Let $\theta \in \mathcal{L}_{P_1}$. By Proposition 20, the sequence $(i_k)_{k\in\mathbb{Z}}$ such that $i_0 = 0$ and $\forall k \leq 0, i_{k-1} = \max\{j \in \mathbb{Z} : j < i_k \text{ and } \theta_{[j,i_k-2]} \text{ is a left-wall}\}$, and $\forall k \geq 0, i_{k+1} = \min\{j \in \mathbb{Z} : j > i_k \text{ and } \theta_{[i_k+2,j]} \text{ is a right-wall}\}$ is well-defined. For any $k < 0$, $\theta_{[i_k,i_{k+1}-2]}$ is a left-wall and then $\prod_{j=i_k}^{i_{k+1}-2} \lambda^-_{\theta_j} = 0$. So, for any $k < 0$, setting $n = \min\{l \in \mathbb{Z} : \prod_{j=i_k}^{l} \lambda^-_{\theta_j} = 0\}$, $p_k = (-,1) \xrightarrow{\theta_{i_k}} (-,\lambda^-_{\theta_{i_k}}) \xrightarrow{\theta_{i_k+1}} \ldots \xrightarrow{\theta_n} (-,\prod_{j=i_k}^{n} \lambda^-_{\theta_j}) \xrightarrow{\theta_{n+1}} (-,1) \ldots \xrightarrow{\theta_{i_{k+1}-1}} (-,1)$ is a finite path in $\mathcal{A}$ from $(-,1)$ to $(-,1)$ with label $\theta_{[i_k,i_{k+1}-1]}$ which visits an initial state. Similarly, for any $k \geq 0$, there exists a finite path $p_k$ in $\mathcal{A}$ from $(+,1)$ to $(+,1)$ with label $\theta_{[i_k+1,i_{k+1}]}$ which visits a final state. Then, $p = (p_k)_{k\in\mathbb{N}}$ is a successful bi-infinite path in $\mathcal{A}$ with label $\theta$. Hence, $\theta \in \mathcal{L}^\zeta(\mathcal{A})$. □

## 6 Conclusions

This paper investigates complexity classes associated to languages characterizing distributions of local rules in $\nu$-CA. Several interesting research directions should be explored.

First, we have proved that the language associated with distributions of equicontinuous or sensitive $\nu$-CA is $\zeta$-rational for the class of linear $\nu$-CA with radius 1. It would be interesting to extend this result to sets of local rule distributions with higher radius. This seems quite difficult because this problem reduces to the study of the equicontinuity of $\nu$-CA of radius 1 on a non-commutative ring, loosing in this way "handy" results like Proposition 21.

Second, there is no complexity gap between sets of distributions which give injective $\nu$-CA and sensitive (plus the previously mentioned constraints). This is contrary to intuition since injectivity is a property of the global transition function whereas sensitivity is a property of its iterates. Indeed, we suspect that the characterization of distributions giving injective $\nu$-CA could be strengthened to deterministic $\zeta$-rational languages.

As a third research direction, it would be interesting to study which dynamical property of $\nu$-CA is associated with languages of complexity higher than $\zeta$-rational. We believe that sensitivity to initial conditions (with no further constraints) is a good candidate.

A further research direction would diverge from $\nu$-CA domain and investigate the topological structure of languages given by the quick-fail acceptance condition for finite automata in the vein of [17]. The authors have just started investigating this last subject.

## References

1. Amoroso, S., Patt, Y.N.: Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures. J. Comput. Syst. Sci. 6(5), 448–464 (1972)

2. Berstel, J., Perrin, D.: Theory of Codes. Academic Press (1985)
3. Cattaneo, G., Dennunzio, A., Formenti, E., Provillard, J.: Non-Uniform Cellular Automata. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 302–313. Springer, Heidelberg (2009)
4. Chaudhuri, P., Chowdhury, D., Nandi, S., Chattopadhyay, S.: Additive Cellular Automata Theory and Applications, vol. 1. IEEE Press (1997)
5. Dennunzio, A., Formenti, E., Provillard, J.: Local rule distributions, language complexity and non-uniform cellular automata. ArXiv e-prints (2011)
6. Dennunzio, A., Formenti, E., Provillard, J.: Non-uniform cellular automata: classes, dynamics, and decidability. ArXiv e-prints (2011)
7. Devaney, R.L.: An Introduction to Chaotic Dynamical Systems, 2nd edn. Westview Pr., Short Disc (2003)
8. Durand, B., Formenti, E., Róka, Z.: Number-conserving cellular automata I: decidability. Theoretical Computer Science 299(1-3), 523–535 (2003)
9. Durand, B., Formenti, E., Varouchas, G.: On Undecidability of Equicontinuity Classification for Cellular Automata. In: Calude, C.S., Dinneen, M.J., Vajnovszki, V. (eds.) DMTCS 2003. LNCS, vol. 2731, pp. 117–128. Springer, Heidelberg (2003)
10. Fúster-Sabater, A., Caballero-Gil, P., Pazo-Robles, M.E.: Application of Linear Hybrid Cellular Automata to Stream Ciphers. In: Moreno Díaz, R., Pichler, F., Quesada Arencibia, A. (eds.) EUROCAST 2007. LNCS, vol. 4739, pp. 564–571. Springer, Heidelberg (2007)
11. Gerlee, P., Anderson, A.R.A.: Stability analysis of a hybrid cellular automaton model of cell colony growth. Phys. Rev. E 75, 051911 (2007)
12. Hedlund, G.A.: Endomorphisms and automorphisms of the shift dynamical system. Theory of Computing Systems 3(4), 320–375 (1969)
13. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley (2006)
14. Kari, J.: Reversibility and surjectivity problems of cellular automata. Journal of Computer and System Sciences 48, 149–182 (1994)
15. Kůrka, P.: Languages, equicontinuity and attractors in cellular automata. Ergodic Theory and Dynamical Systems 17(2), 417–433 (1997)
16. Lind, D., Marcus, B.: An introduction to symbolic dynamics and coding. Cambridge University Press, New York (1995)
17. Litovsky, I., Staiger, L.: Finite acceptance of infinite words. Theoretical Computer Science 174, 1–21 (1997)
18. Manzini, G., Margara, L.: A complete and efficiently computable topological classification of d-dimensional linear cellular automata over Zm. Theoretical Computer Science 221(1-2), 157–177 (1999)
19. Moore, E.F.: Machine models of self-reproduction. In: Proceedings of Symposia in Applied Mathematics, vol. 14, pp. 17–33 (1962)
20. Myhill, J.: The converse of Moore's garden-of-Eden theorem. Proceedings of the American Mathematical Society 14(4), 685–686 (1963)
21. Nivat, M., Perrin, D.: Ensembles reconnaissables de mots biinfinis. In: STOC, pp. 47–59. ACM (1982)
22. Perrin, D., Pin, J.E.: Infinite Words, Pure and Applied Mathematics, vol. 141. Elsevier (2004)
23. Sutner, K.: De Bruijn graphs and linear cellular automata. Complex Systems 5, 19–30 (1991)

# Conservative Groupoids Recognize Only Regular Languages

Danny Dubé[1], Mario Latendresse[2], and Pascal Tesson[1]

[1] Université Laval, Québec, Canada
{danny.dube,pascal.tesson}@ift.ulaval.ca
[2] Canada
latendre@iro.umontreal.ca

**Abstract.** The notion of recognition of a language by a finite semigroup can be generalized to recognition by finite groupoids, i.e. sets equipped with a binary operation '$\cdot$' which is not necessarily associative. It is well known that $L$ can be recognized by a groupoid iff $L$ is context-free. But it is also known that some subclasses of groupoids can only recognize regular languages. For example, *loops* recognize exactly the regular open languages and Beaudry et al. described the largest class of groupoids known to recognize only regular languages.

A groupoid $H$ is said to be *conservative* if $a \cdot b$ is in $\{a, b\}$ for all $a$, $b$ in $H$. The main result of this paper is that conservative groupoids can only recognize regular languages. This class is incomparable with the one of Beaudry et al. so we are exhibiting a new sense in which a groupoid can be too weak to have context-free capabilities.

## 1 Introduction

A semigroup $S$ is a set with a binary associative operation '$\cdot$'. It is a monoid if it also has an identity element. The algebraic point of view on automata, which is central to some of the most important results in the study of regular languages, relies on viewing a finite semigroup as a language recognizer. This enables one to classify a regular language according to the semigroups or monoids able to recognize it. There are various ways in which to formalize this idea but the following one will be useful in our context: a language $L \subseteq \Sigma^*$ is recognized by a finite monoid $M$ if there is a homomorphism $\phi$ from the free monoid $\Sigma^*$ to the free monoid $M^*$ and a set $F \subseteq M$ such that $w \in L$ iff $\phi(w)$ is a sequence of elements whose product lies in $F$. Since the operation of $M$ is associative, this product is well defined. This framework underlies algebraic characterizations of many important classes of regular languages (see [7] for a survey).

These ideas have been extended to non-associative binary algebras, i.e. groupoids. If a groupoid is non-associative, a string of groupoid elements does not have a well-defined product so the above notion of language recognition must be tweaked. A language $L$ is said to be recognized by a finite groupoid $H$ if there is a homomorphism $\phi$ from the free monoid $\Sigma^*$ to the free monoid $H^*$ and a set $F \subseteq H$ such that $w \in L$ iff the sequence of groupoid elements $\phi(w)$

can be bracketed so that the resulting product lies in $F$. It can be shown that a language can be recognized by a finite groupoid iff it is context-free.

However, certain groupoids are too weak to recognize non-regular languages. The first non-trivial example was provided by Caussinus and Lemieux who showed that loops (groupoids with an identity element and left/right inverses) can only recognize regular languages. Beaudry et al. later showed that the languages recognized by loops are precisely the regular open languages [4]. Along the same lines, Beaudry showed in [3] that if $H$ is a groupoid whose multiplication monoid $\mathcal{M}(H)$ is in the variety **DA** then it can only recognize regular languages. ($\mathcal{M}(H)$ is the transformation monoid generated by the rows of the multiplication table of $H$.) Finally, Beaudry et al. proved that this still holds if the multiplication monoid lies in the larger variety **DO** [5].

A groupoid $H$ is said to be *conservative* if for all $x, y \in H$ we have $x \cdot y = x$ or $x \cdot y = y$. The simplest example of a non-associative, conservative groupoid is the one defined by the Rock-Paper-Scissors game. In this game, two players simultaneously make a sign with their fingers chosen among Rock, Paper, and Scissors. Rock beats Scissors, Scissors beats Paper, Paper beats Rock, and identical signs result in a tie. The associated groupoid has three elements $R, P, S$ and the multiplication is given by $R \cdot R = R \cdot S = S \cdot R = R$; $P \cdot P = P \cdot R = R \cdot P = P$; and $S \cdot S = S \cdot P = P \cdot S = S$. Note that $H$ is indeed conservative and also non-associative since $(R \cdot P) \cdot S = S \neq R = R \cdot (P \cdot S)$. The main result of our paper is that conservative groupoids recognize only regular languages. Our results are incomparable to those of Beaudry et al. Indeed a straightforward calculation shows that the multiplication monoid of the Rock-Paper-Scissors groupoid does not belong to the variety **DO** nor to the larger variety **DS**.

### 1.1 Conservative Groupoids and Tournaments

It is convenient to think of conservative groupoids as defining a generalization of the Rock-Paper-Scissors game. For any conservative groupoid $H$, we define the game in which players 1 and 2 each choose an element of $H$ (say $a$ and $b$ respectively) and player 1 wins iff $a \cdot b = a$. In fact, it is helpful to think of this game as a competition between elements of $H$.

Consider now a sequence $w \in H^*$ of elements of the groupoid. A bracketing of this sequence can be viewed as specifying a tournament structure involving the symbols of $w$, i.e. a specific way to determine a winner among the elements of $w$. For instance, if $w = abcd$, then $(a \cdot b) \cdot (c \cdot d)$ is the tournament that first pits $a$ against $b$ and $c$ again $d$ and then has the two winners of that first round competing. Similarly in the tournament $((a \cdot b) \cdot c) \cdot d$ we first have $a$ facing $b$ with the winner then facing $c$ and the winner of that facing $d$. Note that this analogy makes sense because $H$ is conservative and the "winner" of any such tournament (i.e. the value of the product given this bracketing) is indeed one of the participants (in the above example, one of $a$, $b$, $c$, or $d$). We intend to study languages $\Lambda(x) = \{w \in H^* \mid w \text{ can be bracketed to give } x\}$ and we accordingly think of them as $\Lambda(x) = \{w \in H^* \mid \text{ an organizer can rig a tournament structure for } w \text{ to ensure that } x \text{ wins }\}$.

Let us define the *contest trees*. We denote the set of all contest trees by $\mathcal{T}$. It is the smallest set such that the leaf tree $a$ is in $\mathcal{T}$, for any $a$ in the alphabet $\Sigma$, and the tree $t_1 \otimes t_2$ is also in $\mathcal{T}$, for any two trees $t_1$ and $t_2$ in $\mathcal{T}$. Let $T : \Sigma^+ \to 2^{\mathcal{T}}$ be the function that computes the set of possible contest trees in a given contest.

$$\begin{aligned} T(a) &= \{a\} \\ T(w) &= \{t_1 \otimes t_2 \mid u, v \in \Sigma^+,\ u\,v = w,\ t_1 \in T(u),\ t_2 \in T(v)\}, \qquad \text{if } |w| > 1 \end{aligned}$$

Note that, when performing the left-to-right traversal of a tree in $T(w)$, the leaves that we successively reach are the symbols that form $w$. Next, function $W : \mathcal{T} \to \Sigma$ computes the *winner* of a contest tree.

$$\begin{aligned} W(a) &= a \\ W(t_1 \otimes t_2) &= W(t_1) \cdot W(t_2) \end{aligned}$$

Note that the winner of a contest tree is unique. Next, we define the set of possible *winners* in a given contest $w$ by overloading function $W$ with an additional definition of type $\Sigma^+ \to 2^{\Sigma}$. We define $W(w)$ as $\{W(t) \mid t \in T(w)\}$. Finally, for $a \in \Sigma$, we denote by $\Lambda(a)$ the *language* of the words for which we can arrange a contest in which $a$ is the winner. We can give a more formal, alternative definition of $\Lambda(a)$ as $\{w \in \Sigma^+ \mid a \in W(w)\}$.

Variable $t$ denotes contest trees. Variables $A$ and $B$ are used to denote the non-terminals of context-free grammars. Variable $r$ denotes regular expressions. The language generated by $A$ (resp. $r$) is denoted by $L(A)$ (resp. $L(r)$). We denote the empty string by $\epsilon$.

The paper is organized as follows. In Section 2, we show that *commutative* conservative groupoids recognize only regular languages. This is extended to the general case in Section 3. In Section 4, we give a partial algebraic characterization of the languages recognized by conservative groupoids.

## 2 Commutative Case

Let us consider an RPS-like game in which the operator is defined everywhere, conservative, and commutative.

**Theorem 1.** *Given an arbitrary symbol $a$ in $\Sigma$, the language $\Lambda(a)$ is regular.*

The demonstration proceeds in two steps. In the first step, we build a context-free grammar $G$ that generates $\Lambda(a)$. In the second step, we show that $G$ can be rewritten into a regular expression.

### 2.1 Building the Grammar

Let us first define the auxiliary function $f : \Sigma \to 2^{\Sigma}$ that, given a symbol $b$, returns the symbols that are *favorable* to $b$; i.e. the symbols that $b$ defeats. Formally: $f(b) = \{c \in \Sigma \mid b \cdot c = b\}$.

We build the grammar $G = (N, \Sigma, A_a, R)$ where $N$ is the set of non-terminals, $\Sigma$ is the set of terminals, $A_a$ is the start non-terminal, and $R$ is the set of productions, where:

$$\begin{aligned}
N &= \{A_a\} \cup \{B_\sigma \mid \emptyset \neq \sigma \subseteq \Sigma\} \quad \text{and} \\
R &= \{A_a \to B_\sigma\, a\, B_\sigma \mid \sigma = f(a)\} \\
&\quad \cup \{B_\sigma \to B_{\sigma'}\, b\, B_{\sigma'} \mid \sigma \subseteq \Sigma,\ b \in \sigma,\ \sigma' = \sigma \cup f(b)\} \\
&\quad \cup \{B_\sigma \to \epsilon \mid \emptyset \neq \sigma \subseteq \Sigma\}.
\end{aligned}$$

We claim that $G$ is built in such a way that all the words generated by $A_a$ allow $a$ to be the winner if we arrange the contest properly. We also claim that those generated by $B_\sigma$ are either empty or allow the contest to be arranged so that the winner is in $\sigma$. We intend to demonstrate that the non-terminals generate words with such properties but, also, that they generate *all* such words.

## 2.2 Correctness of the Grammar

**Lemma 2.** *For all non-empty subsets $\sigma$ of symbols, $L(B_\sigma) \subseteq \bigcup_{b\in\sigma} \Lambda(b) \cup \{\epsilon\}$.*

*Proof.* We proceed by induction on the length of the words generated by the family of the $B$ non-terminals. *Base case.* Let us consider a word $w$ of length 0. This means that $w$ must be $\epsilon$. Then the inclusion is trivially respected, for every $\emptyset \neq \sigma \subseteq \Sigma$. *Induction hypothesis (IH).* Let us suppose that, for all $\emptyset \neq \sigma \subseteq \Sigma$ and for all $w \in \Sigma^*$ such that $|w| \leq n$, we have that, whenever $w \in L(B_\sigma)$, then $w \in \bigcup_{b\in\sigma} \Lambda(b) \cup \{\epsilon\}$. *Induction step.* Let us consider a word $w$ of length $n+1$ and $\emptyset \neq \sigma \subseteq \Sigma$ such that $w \in L(B_\sigma)$. Since $w$ is non-empty, we must show that it is in $\bigcup_{b\in\sigma} \Lambda(b)$. We will do so by constructing a contest tree for $w$ whose winner is in $\sigma$. Let us choose a derivation tree for $w$. Given that $w \neq \epsilon$, let $B_\sigma \to B_{\sigma'}\, b\, B_{\sigma'}$ be the production that is used at the root of the derivation tree, where $\sigma' = \sigma \cup f(b)$. This implies that there exist $u$, $v \in \Sigma^*$ such that $w = u\, b\, v$, where $u$, $v \in L(B_{\sigma'})$. Since both $u$ and $v$ are of length at most $n$, then the IH applies to each. So, if $u \neq \epsilon$, then there exists a contest tree $t_u \in T(u)$ such that $W(t_u) \in \sigma'$. Similarly, if $v \neq \epsilon$, then there exists $t_v \in T(v)$ such that $W(t_v) \in \sigma'$. There are three cases to consider for $u$ (and similarly for $v$): $u = \epsilon$, $W(t_u) \in \sigma' - \sigma$, and $W(t_u) \in \sigma$. Hence, we would have a total of nine cases to analyze. In each case, we would have to show that we can build a contest tree $t_w \in T(w)$ such that $W(t_w) \in \sigma$. For the sake of conciseness, we only examine the case where $u \neq \epsilon \neq v$, $W(t_u) \in \sigma' - \sigma$, and $W(t_v) \in \sigma$. In this case, we select $t_w = (t_u \otimes b) \otimes t_v$ and we have that:

$$\begin{aligned}
& W(t_w) \\
&= W((t_u \otimes b) \otimes t_v) \\
&= W(t_u \otimes b) \cdot W(t_v) && \text{by def. of } W \\
&= (W(t_u) \cdot W(b)) \cdot W(t_v) \\
&= (W(t_u) \cdot b) \cdot W(t_v) \\
&= b \cdot W(t_v) && \text{because } W(t_u) \in \sigma' - \sigma \subseteq f(b) \\
&\in \sigma && \text{because } b,\ W(t_v) \in \sigma \text{ and ` $\cdot$ ' is conserv.}
\end{aligned}$$

| | $v = \epsilon$ | $W(t_v) \in \sigma' - \sigma$ | $W(t_v) \in \sigma$ |
|---|---|---|---|
| $u = \epsilon$ | $b$ | $b \otimes t_v$ | $b \otimes t_v$ |
| $W(t_u) \in \sigma' - \sigma$ | $t_u \otimes b$ | $(t_u \otimes b) \otimes t_v$ | $(t_u \otimes b) \otimes t_v$ |
| $W(t_u) \in \sigma$ | $t_u \otimes b$ | $t_u \otimes (b \otimes t_v)$ | $t_u \otimes (b \otimes t_v)$ |

**Fig. 1.** Key step in the inductive cases in the proof of correctness

Figure 1 presents a $t_w$ that should be selected in each of the nine cases.

We now have established that $L(B_\sigma) \subseteq \bigcup_{b \in \sigma} \Lambda(b) \cup \{\epsilon\}$. What remains to show is Lemma 3.

**Lemma 3.** *For any $a \in \Sigma$, $L(A_a) \subseteq \Lambda(a)$.*

*Proof.* This is easy as the only $A_a$-production is $A_a \to B_{f(a)}\, a\, B_{f(a)}$, where $B_{f(a)}$ generates either the empty word or a word for which we can arrange a contest such that the winner is favorable to $a$. So, by analyzing four cases, we can show that we can arrange for all words in $L(A_a)$ to have $a$ as a winner.

## 2.3 Completeness of the Grammar

**Lemma 4.** *For all non-empty subsets $\sigma$ of symbols, $\bigcup_{b \in \sigma} \Lambda(b) \cup \{\epsilon\} \subseteq L(B_\sigma)$.*

*Proof.* We do so by induction on the length of the words. *Base case.* Let us consider a word $w$ of length 0. We have that $w = \epsilon$. Because of the productions of the form $B_\sigma \to \epsilon$, we have that $\epsilon \in L(B_\sigma)$, for all $\sigma \subseteq \Sigma$. *Induction hypothesis.* We suppose that the inclusion holds for all $\emptyset \neq \sigma \subseteq \Sigma$ and all words of length at most $n$. *Induction step.* Let us consider $\emptyset \neq \sigma \subseteq \Sigma$ and $w$ of length $n+1$ such that $w \in \bigcup_{b \in \sigma} \Lambda(b) \cup \{\epsilon\}$. Since $w \neq \epsilon$, let $b \in \sigma$ such that $w \in \Lambda(b)$. Let $\sigma' = \sigma \cup f(b)$. There exists a contest tree $t_w \in T(w)$ such that $W(t_w) = b$. There are two possible shapes for $t_w$: it is a leaf or it is a larger tree. If $t_w$ is a leaf, then $w = b$ and the derivation $B_\sigma \Rightarrow B_{\sigma'}\, b\, B_{\sigma'} \Rightarrow b\, B_{\sigma'} \Rightarrow b$ shows that $w \in L(B_\sigma)$. Otherwise, $t_w = t_u \otimes t_v$ where $b$ is the winner of at least one of $t_u$ and $t_v$ (by conservativeness of '$\cdot$'). Without loss of generality, let us suppose that $W(t_v) = b$. We know that $W(t_u) \in f(b)$, since $W(t_u)$ is defeated by $b$. Let $u$, $v \in \Sigma^+$ such that $t_u \in T(u)$ and $t_v \in T(v)$. Note that $1 \leq |v| \leq n$. Since $W(t_v) = b$, then $v \in \Lambda(b)$, so $v \in \Lambda(b) \cup \{\epsilon\}$, and (by IH) $v \in L(B_{\{b\}})$. Since $v \neq \epsilon$, there exists a derivation $B_{\{b\}} \Rightarrow^+ v$ that first uses the production $B_{\{b\}} \to B_{\sigma''}\, b\, B_{\sigma''}$, where $\sigma'' = \{b\} \cup f(b)$. Let $v'$, $v'' \in \Sigma^*$ such that $v = v'\, b\, v''$ and $v'$, $v'' \in L(B_{\sigma''})$. At this point, we have that $w = u\, v'\, b\, v''$, that there exists a contest tree $t_u \in T(u)$ such that $W(t_u) \in f(b)$, either that $v' = \epsilon$ or that there exists a contest tree $t_{v'} \in T(v')$ such that $W(t_{v'}) \in \sigma''$, and either that $v'' = \epsilon$ or that there exists a contest tree $t_{v''} \in T(v'')$ such that $W(t_{v''}) \in \sigma''$. Given the possible emptiness of $v'$ and that of $v''$, we would have four cases to analyze. We only examine the case where both $v'$ and $v''$ are non-empty, as the other cases are simpler. Let $t_{u\, v'} = t_u \otimes t_{v'}$. By conservativeness, we have that $W(t_{u\, v'}) \in \sigma''$ and, so,

$W(t_{u\,v'}) \in \sigma'$. Also, we have that $W(t_{v''}) \in \sigma''$ and, so, $W(t_{v''}) \in \sigma'$. Since $|u\,v'| \leq n \geq |v''|$, we use the IH and obtain that $u\,v'$, $v'' \in L(B_{\sigma'})$. Thus the derivation $B_\sigma \Rightarrow B_{\sigma'}\, b\, B_{\sigma'} \Rightarrow^* u\,v'\,b\,B_{\sigma'} \Rightarrow^* u\,v'\,b\,v'' = w$ shows that $w \in L(B_\sigma)$.

At this point, we have established that $\bigcup_{b\in\sigma} \Lambda(b) \cup \{\epsilon\} \subseteq L(B_\sigma)$. What remains to show is Lemma 5.

**Lemma 5.** *For any $a \in \Sigma$, $\Lambda(a) \subseteq L(A_a)$.*

*Proof.* This is shown by noting that, for any word $w$ in $\Lambda(a)$, there is a contest tree $t_w \in T(w)$ such that $W(t_w) = a$ and analyzing the three following cases: $t_w = a$, $t_w = t_u \otimes t_v$ where $W(t_u) = a$, and $t_w = t_u \otimes t_v$ where $W(t_u) \neq a$. In each case, it is simple to exhibit a derivation $A_a \Rightarrow^* w$, using arguments similar to those used in the demonstration of completeness for the $B_\sigma$'s.

## 2.4 Regularity of the Language Generated by the Grammar

Before we demonstrate that $L(G)$ is regular, we present a couple of lemmas. In the first lemma, we make the rather intuitive observation that, the larger the set $\sigma$ in $B_\sigma$, the larger the generated language.

**Lemma 6.** *For any $\emptyset \neq \sigma \subseteq \sigma' \subseteq \Sigma$, we have that $L(B_\sigma) \subseteq L(B_{\sigma'})$.*

*Proof.*
$$L(B_\sigma) = \bigcup_{b\in\sigma} \Lambda(b) \cup \{\epsilon\} \subseteq \bigcup_{b\in\sigma'} \Lambda(b) \cup \{\epsilon\} = L(B_{\sigma'}).$$

**Lemma 7.** *For any $\emptyset \neq \sigma \subseteq \sigma' \subseteq \Sigma$, we have that $L(B_{\sigma'}) = L(B_{\sigma'}\, B_\sigma)$.*

*Proof.* Showing that $L(B_{\sigma'}) \subseteq L(B_{\sigma'}\, B_\sigma)$ is trivial as it is sufficient to systematically use production $B_\sigma \to \epsilon$. Let us show $L(B_{\sigma'}\, B_\sigma) \subseteq L(B_{\sigma'})$ by induction on the length of the words. *Induction basis.* Let $w \in \Sigma^*$ of length 0. Then, $w$ has to be $\epsilon$ and $w \in L(B_{\sigma'})$. *Induction step.* Let $w \in L(B_{\sigma'}\, B_\sigma)$ of length $n+1$. There exist $u \in L(B_{\sigma'})$ and $v \in L(B_\sigma)$ such that $w = u\,v$. If $u = \epsilon$, then $w = v \in L(B_\sigma) \subseteq L(B_{\sigma'})$ and we are done. Otherwise, there exist $b \in \sigma'$, $\sigma'' = \sigma' \cup f(b)$, and $u'$, $u'' \in L(B_{\sigma''})$ such that $u = u'\,b\,u''$. Note that $|u''\,v| \leq n$ and, since $u''\,v \in L(B_{\sigma''}\, B_\sigma)$ with $\sigma \subseteq \sigma''$, the IH can be used to get that $u''\,v \in L(B_{\sigma''})$. Thus $w = u'\,b\,u''\,v \in L(B_{\sigma''}\, \{b\}\, B_{\sigma''}) \subseteq L(B_{\sigma'})$ and we are done.

Now we can turn to the main task of showing Lemma 8.

**Lemma 8.** *$G$ generates a regular language.*

*Proof.* Let us examine $G$'s productions. Note that each time there is a production of the form $B_\sigma \to B_{\sigma'}\, b\, B_{\sigma'}$, then we have that $\sigma \subseteq \sigma'$. The productions can be classified into two kinds: those for which $\sigma = \sigma'$ and those for which $\sigma \subset \sigma'$. The second kind of productions introduces no recursion among the non-terminals. The first kind does but only via self-recursion. We show that this does not lead to an non-regular language.

$$\begin{array}{lllll|lllllll}
C_i & \to & C_i & b_1 & C_i & r_i & = & ( & & b_1 & & \mid \\
& \ldots & & & & & & & & \ldots & & \mid \\
C_i & \to & C_i & b_{k'} & C_i & & & & & b_{k'} & & \mid \\
C_i & \to & C_{l_{k'+1}} & b_{k'+1} & C_{l_{k'+1}} & & & & r_{l_{k'+1}} & b_{k'+1} & r_{l_{k'+1}} & \mid \\
& \ldots & & & & & & & & \ldots & & \mid \\
C_i & \to & C_{l_{k-1}} & b_{k-1} & C_{l_{k-1}} & & & & r_{l_{k-1}} & b_{k-1} & r_{l_{k-1}} & )^* \\
C_i & \to & \epsilon & & & & & & & & &
\end{array}$$

$$\text{where } \max(l_{k'+1}, \ldots, l_{k-1}) < i$$

**Fig. 2.** Converting productions (possibly) with self-recursion into a regular expression

The non-empty subsets of $\Sigma$ form a partially ordered set, with respect to inclusion ($\supseteq$). Let $\sigma_1, \ldots, \sigma_{2^{|\Sigma|}-1}$ be a topological ordering of the non-empty subsets of $\Sigma$ such that if $\sigma_j \supset \sigma_i$, then $j < i$. As a consequence, $\sigma_1$ has to be $\Sigma$ and $\sigma_{2^{|\Sigma|}-1}$ has to be one of the singletons. Let us use the alias non-terminals $C_1, \ldots, C_{2^{|\Sigma|}-1}$ for the permutation of the $B_\sigma$'s according to this ordering; i.e. $C_i = B_{\sigma_i}$, for $1 \leq i \leq 2^{|\Sigma|} - 1$. Consequently, we now view production $B_{\sigma_i} \to \epsilon$ as $C_i \to \epsilon$ and production $B_{\sigma_i} \to B_{\sigma_j}\, b\, B_{\sigma_j}$ as $C_i \to C_j\, b\, C_j$, where $i = j$ if and only if $\sigma_i = \sigma_j$.

In order to show that $A_a$ generates a regular language, we successively show, by induction on $i$, that each non-terminal $C_i$ generates the same language as a regular expression $r_i$. We can then conclude that $A_a$ also generates a regular language. We do an inductive reasoning on the $C_i$'s but, as will be apparent, there is no need to provide a special case for the basis.

Let $1 \leq i \leq 2^{|\Sigma|}-1$. By IH, we know that each $C_j$ generates the same language as some regular expression $r_j$, for $j < i$. Let us consider all the $C_i$-productions $P_1, \ldots, P_k$. Without loss of generality, let us suppose that $P_1, \ldots, P_{k'}$ are the productions that involve self-recursion, that $P_{k'+1}, \ldots, P_{k-1}$ are those that involve non-self-recursion, and that $P_k$ is $C_i \to \epsilon$. Note that we have $0 \leq k' < k$. Value $k'$ might be as low as 0 because there might exist non-terminals for which all productions are non-self-recursive. Value $k'$ cannot get higher than $k - 1$ because there is production $P_k$.[1] For the sake of illustration, we list the productions on the left-hand side of Figure 2. On the right-hand side of Figure 2, there is a single replacement equation whose right member is regular expression $r_i$. The regular expression is a Kleene iteration over a union that contains an alternative for each non-$\epsilon$-production. Each self-recursive production is converted into its middle symbol. Each non-self-recursive production is trivially converted into a regular expression, as the concerned non-terminals already denote regular languages, by IH. We must show that $r_i$ generates $L(C_i)$.

We start by showing that $L(r_i) \subseteq L(C_i)$. We do so by induction on the size of the words. *Induction basis.* Let $w \in \Sigma^*$ of length 0. Then, $w$ must be $\epsilon$ and, clearly, $w \in L(C_i)$. *Induction step.* Let $w \in L(r_i)$ of length $n+1$. By construction,

[1] For instance, in the case of $C_1 = B_{\sigma_1} = B_\Sigma$, $k' = k - 1$ since all the productions are self-recursive except $C_1 \to \epsilon$.

we have that $r_i = (r_i')^*$, where $r_i'$ is a union. Since $w \neq \epsilon$, then $w \in L(r_i'\, r_i)$, which means that there exist $u \in \Sigma^+$ and $v \in \Sigma^*$ such that $w = u\, v$, $u \in L(r_i')$, and $v \in L(r_i)$. Since $|v| \leq n$, we know that $v \in L(C_i)$, by IH. There are two cases for $u$: either $u = b_m$, for $1 \leq m \leq k'$, or $u \in L(r_{l_m}\, b_m\, r_{l_m})$, for $k'+1 \leq m \leq k-1$. In the first case, the following derivation shows that $w \in L(C_i)$: $C_i \Rightarrow C_i\, b_m\, C_i \Rightarrow b_m\, C_i \Rightarrow^* b_m\, v = u\, v = w$. In the second case, we have that there exist $u'$, $u'' \in L(r_{l_m})$ such that $u = u'\, b_m\, u''$. By IH, we have that $u'$, $u'' \in L(C_{l_m})$. By construction of $G$, we know that $C_i = B_\sigma$ and $C_{l_m} = B_{\sigma'}$ such that $\sigma \subset \sigma'$. Consequently, $L(C_i) \subseteq L(C_{l_m})$. By the second lemma, $L(C_{l_m}) = L(C_{l_m}\, C_i)$. Consequently, $w = u\, v = u'\, b_m\, u''\, v$, where $u' \in L(C_{l_m})$ and $u''\, v \in L(C_{l_m}\, C_i) = L(C_{l_m})$, which guarantees that $C_i \Rightarrow C_{l_m}\, b_m\, C_{l_m} \Rightarrow^* w$.

We continue by showing that $L(C_i) \subseteq L(r_i)$. Once again, we show this result by induction on the length of the words. *Induction basis.* Let $w \in \Sigma^*$ of length 0. Then, $w$ must be $\epsilon$ and, clearly, $w \in L(r_i)$. *Induction step.* Let $w \in L(C_i)$ of length $n + 1$. Since $w \neq \epsilon$, the derivation $C_i \Rightarrow^* w$ must start with the use of a non-$\epsilon$-production. Then, two cases are possible. If the production is one of the first $k'$ ones, then it is $C_i \to C_i\, b_m\, C_i$, for $m \leq k'$, and there exist $u$, $v \in L(C_i)$ such that $w = u\, b_m\, v$. Since $|u| \leq n \geq |v|$, we use the IH and have that $u$, $v \in L(r_i)$. Consequently, $w = u\, b_m\, v \in L(r_i)\, L(r_i)\, L(r_i) \subseteq L(r_i)$, since $r_i$ is a Kleene iteration. In the other case, the first production used is $C_i \to C_{l_m}\, b_m\, C_{l_m}$, for $k' < m \leq k-1$, and there exist $u$, $v \in L(C_{l_m})$ such that $w = u\, b_m\, v$. By construction of $G$ and the ordering of the $C$ non-terminals, we know that $C_{l_m}$ appears *before* $C_i$ (i.e. $l_m < i$) in the ordering and so $C_{l_m}$ generates the same language as $r_{l_m}$, by IH. Consequently, $w = u\, b_m\, v \in L(r_{l_m})\, \{b_m\}\, L(r_{l_m}) \subseteq L(r_i)$.

## 3 Non-commutative Case

We now study the case where the operator is non-commutative but still conservative and defined everywhere. That is, there exist $a$, $b \in \Sigma$ such that $a \cdot b \neq b \cdot a$. We show Theorem 9. We show it using an adaptation of the grammar-based method of Section 2.

**Theorem 9.** *Given an arbitrary symbol $a$ in $\Sigma$, the language $\Lambda(a)$ is regular.*

As in the commutative case, we start by giving the construction of a context-free grammar that generates $\Lambda(a)$ and then show that its language is regular.

### 3.1 Building the Grammar

Due to the loss of the commutativity property, we now need *two* auxiliary functions that return, for a given symbol $b$, the symbols that are *favorable* to $b$: functions $f_L, f_R : \Sigma \to 2^\Sigma$ for the symbols that are defeated when they appear on the left-hand side of the operator and those that are defeated when they appear on the right-hand side of the operator, respectively. Formally: $f_L(b) = \{c \in \Sigma \mid c \cdot b = b\}$, and $f_R(b) = \{c \in \Sigma \mid b \cdot c = b\}$.

We define the context-free grammar $G = (N, \Sigma, A_a, R)$ where all the components are the same as in the commutative case except for the productions:

$$\begin{aligned} R = \{ &A_a \to B_{\sigma'}\, a\, B_{\sigma''} \mid \sigma' = f_L(a),\ \sigma'' = f_R(a)\} \\ &\cup \{B_\sigma \to B_{\sigma'}\, b\, B_{\sigma''} \mid \sigma \subseteq \Sigma,\ b \in \sigma,\ \sigma' = \sigma \cup f_L(b),\ \sigma'' = \sigma \cup f_R(b)\} \\ &\cup \{B_\sigma \to \epsilon \mid \emptyset \neq \sigma \subseteq \Sigma\}. \end{aligned}$$

The main difference is that we take care of handling the sets of defeated symbols independently on the left- and on the right-hand sides. We do not show the following lemmas as the proofs are similar to those of Section 2.

**Lemma 10.** $L(B_\sigma) = \bigcup_{b \in \sigma} \Lambda(b) \cup \{\epsilon\}$.

**Lemma 11.** $L(A_a) = \Lambda(a)$.

### 3.2 Regularity of the Language Generated by the Grammar

**Lemma 12.** *G generates a regular language.*

*Proof.* In order to show Lemma 12, we use the alias variables $C_1, \ldots, C_{2^{|\Sigma|}-1}$ once again. We remind the reader that $C_i = B_{\sigma_i}$, for $1 \le i \le 2^{|\Sigma|} - 1$, that, if $\sigma_j \supseteq \sigma_i$, we have $j \le i$, and finally that, for any production $C_i \to \alpha$, any $C_j$ that appears in $\alpha$ is such that $j \le i$. Again, we show by induction on $i$ that each $C_i$ generates a regular language, which is the same as the one generated by the regular expression $r_i$. At step $i$, we suppose that $C_j$ is equivalent to $r_j$, for all $j < i$. The changes that must be made to the proof in the non-commutative case relate to the construction of the regular expression $r_i$ and the demonstrations that $L(r_i) \subseteq L(C_i)$ and $L(C_i) \subseteq L(r_i)$.

Let us consider all the $C_i$-productions $P_1, \ldots, P_k$. Without loss of generality, let us suppose that the productions are grouped by kind of recursion. Note that the non-$\epsilon$-productions are of the form $C_i \to C_j\, b\, C_h$, with $j \le i \ge h$, and hence cannot be merely categorized as being self-recursive or not. Let $0 \le k' \le k'' \le k''' < k$ such that $P_1, \ldots, P_{k'}$ are completely self-recursive, $P_{k'+1}, \ldots, P_{k''}$ are self-recursive on the left only, $P_{k''+1}, \ldots, P_{k'''}$ are self-recursive on the right only, and $P_{k'''+1}, \ldots, P_{k-1}$ are not self-recursive at all. Figure 3 presents the original $C_i$-productions and the regular expression $r_i$ in which they are transformed.

As in the commutative case, there remains to show that $L(r_i) = L(C_i)$. Due to lack of space, we omit the proof that each language is contained into the other. However, the arguments are similar to those used in the commutative case, except that there are a few extra sub-cases to analyze; i.e. those for the productions that are self-recursive on the left only and for the productions that are self-recursive on the right only.

## 4 Languages Recognized by Conservative Groupoids

We now know that languages recognized by conservative groupoids are regular and it is natural to seek a more precise characterization. This seems challenging.

$$
\begin{array}{llll|lllllll}
C_i & \to C_i & b_1 & C_i & r_i & = & ( & & b_1 & & | \\
\dots & & & & & & & & \dots & & | \\
C_i & \to C_i & b_{k'} & C_i & & & & & b_{k'} & & | \\
C_i & \to C_i & b_{k'+1} & C_{l_{k'+1}} & & & & & b_{k'+1} & r_{l_{k'+1}} & | \\
\dots & & & & & & & & \dots & & | \\
C_i & \to C_i & b_{k''} & C_{l_{k''}} & & & & & b_{k''} & r_{l_{k''}} & | \\
C_i & \to C_{l_{k''+1}} & b_{k''+1} & C_i & & & & r_{l_{k''+1}} & b_{k''+1} & & | \\
\dots & & & & & & & & \dots & & | \\
C_i & \to C_{l_{k'''}} & b_{k'''} & C_i & & & & r_{l_{k'''}} & b_{k'''} & & | \\
C_i & \to C_{l_{k'''+1}} & b_{k'''+1} & C_{l'_{k'''+1}} & & & & r_{l_{k'''+1}} & b_{k'''+1} & r_{l'_{k'''+1}} & | \\
\dots & & & & & & & & \dots & & | \\
C_i & \to C_{l_{k-1}} & b_{k-1} & C_{l'_{k-1}} & & & & r_{l_{k-1}} & b_{k-1} & r_{l'_{k-1}} & )^* \\
C_i & \to \epsilon & & & & & & & & &
\end{array}
$$

$$\text{where } \max(l_{k'+1}, \dots, l_{k-1}, l'_{k'''+1}, \dots, l'_{k-1}) < i$$

**Fig. 3.** Converting productions into a regular expression in the non-commutative case

One starting point is to consider conservative groupoids which are also associative, i.e. semigroups for which $x \cdot y \in \{x, y\}$. In particular, these satisfy $x^2 = x$ but we can give an exact characterization.

**Lemma 13.** *A semigroup $S$ is conservative iff its set of elements can be partitioned into $k$ classes $C_1, \dots C_k$ such that $x \cdot y = y \cdot x = x$ whenever $x \in C_i$ and $y \in C_j$ for $i > j$ and for any $j$ either $x \cdot y = x$ for all $x, y \in C_j$ (left-zero) or $x \cdot y = y$ for all $x, y \in C_j$ (right-zero).*

*Proof.* By definition, such a semigroup is conservative. Also, the operation defined above is associative. Indeed if $x, y, z$ are three elements lying in the same class $C_i$ then $(x \cdot y) \cdot z = x \cdot (y \cdot z) = x$ if $C_i$ is left-zero and $(x \cdot y) \cdot z = x \cdot (y \cdot z) = z$ if it is right-zero. If $x, y, z$ are not in the same class then associativity follows because the elements in the most absorbing class are the only ones that matter. Suppose for instance that $x$ and $z$ lie in the same class $C_i$ while $y$ lies in some $C_j$ with $i > j$. Since $x \cdot y = x$ and $y \cdot z = z$ we clearly have $(x \cdot y) \cdot z = x \cdot (y \cdot z) = xz$.

Conversely, suppose $S$ is a conservative semigroup. For any $x, y$, one of three cases must hold: (1) $x \cdot y = y \cdot x = x$ (or $\dots = y$), (2) $x \cdot y = x$ and $y \cdot x = y$, or (3) $x \cdot y = y$ and $y \cdot x = x$ and we say that the pair $x, y$ is of type 1, 2, or 3.

First note that cases (2) and (3) define equivalence relations on $S$. Moreover, if $x \neq y$ is a pair of type (2) then there cannot exist a $z \neq x$ such that $x, z$ is a pair of type (3). Indeed, we would then have $z \cdot y = (x \cdot z) \cdot (y \cdot x) = x \cdot z \cdot y \cdot x$. Because $S$ is conservative we must have $z \cdot y \in \{y, z\}$ but this either leads to $x \cdot z \cdot y \cdot x = x \cdot z \cdot x = x$ or $x \cdot z \cdot y \cdot x = x \cdot y \cdot x = x$. Both cases form a contradiction.

These facts allow us to partition $S$ into classes such that within each either $x \cdot y = x$ for all $x, y \in C_j$ or $x \cdot y = y$ for all $x, y \in C_j$. (These $C_j$ correspond to the $\mathcal{J}$-classes of the semigroup.) It remains to show that we can impose a total order on these classes. We simply choose to place class $A$ below class $B$ if there is some $x$ in $A$ and some $y$ in $B$ such that $x \cdot y = y \cdot x = x$. This is well defined:

if, e.g., we choose $z$ another representative of $B$ with $y \cdot z = y$ and $z \cdot y = z$, then $x \cdot z = x \cdot y \cdot z = x \cdot y = x$. Moreover, this forms a total order since for any pair $x, y$ not of type (2) or (3), we must have $x \cdot y = y \cdot x = x$ or $x \cdot y = y \cdot x = y$. It is now straightforward to check that $S$ has the structure described in the statement.

This characterization can be translated into a description of the languages recognizable by conservative semigroups. For an alphabet $\Sigma$, consider a partition $C_1, \ldots, C_k$ each with an associated direction $d_1, \ldots, d_k$ with $d_i \in \{\mathsf{L}, \mathsf{R}\}$. For $a \in C_j$ with $d_j = \mathsf{L}$ (resp. $d_j = \mathsf{R}$), define the language $L_a$ (resp. $R_a$) of words that contain no occurrence of letters in classes $C_i$ with $i < j$ and where the first (resp. last) occurrence of a letter in $C_j$ is an $a$. A language can be recognized by a conservative semigroup iff it is the disjoint union of some $L_a$ and $R_a$.

Note that the class of languages recognized by conservative semigroups does not have many closure properties. For instance, it is not closed under union or intersection: each of the languages $\Sigma^* a \Sigma^*$ and $\Sigma^* b \Sigma^*$ can be recognized but their union (or intersection) has a syntactic semigroup which is not conservative.

The apparent absence of closure properties makes it difficult to provide a complete characterization of languages recognized by *non-associative*, conservative groupoids. We believe that this is nevertheless an interesting challenge and we end this section with a discussion of basic examples and avenues for research.

First, while any language $L$ recognized by a conservative semigroup must be idempotent, this need not be the case for one recognized by a conservative groupoid. A language is idempotent if, for any $x, y, z \in \Sigma^*$, we have $xyz \in L \Leftrightarrow xy^2z \in L$. A language recognized by a conservative groupoid $G$ need not be idempotent despite the fact that $g^2 = g$ for any $g \in G$. For instance, the language $\{a, b\}^* a \{a, b\}^* a \{a, b\}^*$ consisting of words with at least two $a$s is not idempotent. It can however be recognized by the Rock-Paper-Scissors groupoid by setting $\phi(a) = RPS$ and $\phi(b) = \epsilon$ and choosing $\{P\}$ as the accepting set. If $w$ contains no $a$ then $\phi(w) = \epsilon$ cannot produce $P$. If $w$ contains a single $a$ then $\phi(w) = RPS$ and one readily checks that $R \cdot (P \cdot S) = R$ and $(R \cdot P) \cdot S = S$. However, $\phi(aa) = RPSRPS$ which can be bracketed as $((R \cdot P) \cdot (S \cdot (R \cdot (P \cdot S)))) = P$. More generally, if $k \geq 3$ the $k$ copies of $RPS$ can be bracketed as follows. First bracket each of the $k - 2$ last copies of $RPS$ individually to obtain $S$, absorb these $S$s to obtain $RPSRPS$, and use the bracketing above.

Similarly, $\{a, b\}^* a \{a, b\}^* a \{a, b\}^* a \{a, b\}^*$ can be recognized by the four-element groupoid with the multiplication table besides. One can see that both $\phi(a) = 1234$ and $\phi(aa) = 12341234$ can never be bracketed to obtain 3. In both cases, the rightmost 3 cannot win against 4 and the leftmost 3 cannot win against 1, possibly after having defeated 2. On the other hand, 3 can be the winner of $\phi(aaa) = 123412341234$, as shown by the bracketing: $((((1 \cdot (2 \cdot 3)) \cdot (4 \cdot 1)) \cdot 2) \cdot (3 \cdot (((4 \cdot 1) \cdot 2) \cdot (3 \cdot 4))))$.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 4 |
| 2 | 1 | 2 | 3 | 2 |
| 3 | 1 | 3 | 3 | 4 |
| 4 | 4 | 2 | 4 | 4 |

We think that there are similar conservative groupoids that can "count" up to $k$ for any $k$ and have verified this up to $k = 6$. In fact, distinguishing between five and six occurrences of $a$ can be achieved, somewhat counter-intuitively, with a five-element groupoid.

On the other hand, we believe it is impossible to count the occurrences of a letter modulo some $p$ using a conservative groupoid and more generally that every language $L$ recognized by a conservative groupoid $G$ is star-free, i.e. that for some $k$ and any $x, y, z \in \Sigma^*$ we have $xy^kz \in L \Leftrightarrow xy^{k+1}z \in L$.

For any $g$ in a conservative groupoid $G$, we have $g^2 = g$. So whenever $g \in W(u)$, we must also have $g = g^2 \in W(u^2)$. As a referee kindly pointed out, this suffices to show a partial result along those lines, which is that the language $(aa)^*a$ of words of odd length cannot be recognized by $G$.

Due to space restrictions, further partial results are omitted here. An up-to-date extended version of the paper is available from the authors' websites.

## 5 Conclusion and Future Work

We have shown that conservative groupoids can only recognize regular languages. Beaudry, Lemieux, and Thérien had previously exhibited a large class of groupoids with the same limitations but our work is incomparable to theirs and our methods are, accordingly, quite different. It is natural to ask whether our approach can be generalized to find a wider class of "weak" groupoids and an obvious target are the 0-conservative groupoids, i.e. groupoids $H$ with a 0 element such that $0 \cdot x = x \cdot 0 = 0$ for all $x \in H$ and $x \cdot y \in \{x, y, 0\}$ for all $x, y \in H$; i.e. all non-conservative products are 0.

It is well known that left-linear and right-linear context-free grammars generate regular languages [6]. The work presented in this paper can be used to recognize a larger family of context-free grammars that generate regular languages. The work of [1,2] is similar in that regard: context-free grammars with productions of the form $A \to A\alpha A$ (called self-embedded), and other simpler forms of recursion, are shown to generate regular languages.

## References

1. Andrei, S., Cavadini, S., Chin, W.N.: Transforming self-embedded context-free grammars into regular expressions. Tech. Rep. TR 02-06, University "A.I.Cuza" of Iaşi, Faculty of Computer Science (2002)
2. Andrei, S., Chin, W.N., Cavadini, S.V.: Self-embedded context-free grammars with regular counterparts. Acta Inf. 40(5), 349–365 (2004)
3. Beaudry, M.: Languages recognized by finite aperiodic groupoids. Theor. Comput. Sci. 209(1-2), 299–317 (1998)
4. Beaudry, M., Lemieux, F., Thérien, D.: Finite Loops Recognize Exactly the Regular Open Languages. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 110–120. Springer, Heidelberg (1997)
5. Beaudry, M., Lemieux, F., Thérien, D.: Groupoids That Recognize Only Regular Languages. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 421–433. Springer, Heidelberg (2005)
6. Linz, P.: An Introduction to Formal Languages and Automata, 2nd edn. Jones and Bartlett Publishers, Inc., USA (1997)
7. Pin, J.E.: Syntactic semigroups. In: Handbook of Language Theory, ch. 10, vol. 1, pp. 679–746. Springer, Heidelberg (1997)

# Advice Complexity of Online Coloring for Paths*

Michal Forišek[1], Lucia Keller[2], and Monika Steinová[2]

[1] Comenius University, Bratislava, Slovakia
forisek@dcs.fmph.uniba.sk
[2] ETH Zurich, Switzerland
{lucia.keller,monika.steinova}@inf.ethz.ch

**Abstract.** In online graph coloring a graph is revealed to an online algorithm one vertex at a time, and the algorithm must color the vertices as they appear. This paper starts to investigate the advice complexity of this problem – the amount of oracle information an online algorithm needs in order to make optimal choices. We also consider a more general problem – a trade-off between online and offline graph coloring.

In the paper we prove that precisely $\lceil n/2 \rceil - 1$ bits of advice are needed when the vertices on a path are presented for coloring in arbitrary order. The same holds in the more general case when just a subset of the vertices is colored online. However, the problem turns out to be non-trivial for the case where the online algorithm is guaranteed that the vertices it receives form a subset of a path and are presented in the order in which they lie on the path. For this variant we prove that its advice complexity is $\beta n + O(\log n)$ bits, where $\beta \approx 0.406$ is a fixed constant (we give its closed form). This suggests that the generalized problem will be challenging for more complex graph classes.

**Keywords:** Advice Complexity, Online Graph Coloring, Partial Coloring.

## 1 Overview of Advice Complexity

A great challenge in classical algorithmics are problems that work in an online fashion: The instance is not shown to the algorithm all at once. Instead, the algorithm receives it piecewise in consecutive turns. In each turn, the algorithm must produce a piece of the output which must not be changed afterwards. Such algorithms are called *online algorithms*, more on them can be found in [4].

Obviously, it is harder (and sometimes even impossible) to compute the best partial solutions without knowing the future. The output quality of online algorithms is measured by the *competitive ratio* of a particular instance, see [4,12]. This ratio is the quotient of the cost of the solution produced by the given online algorithm and the cost of an optimal solution (i.e., the cost of the output of an optimal offline algorithm for that particular instance).

* The research is partially funded by the SNF grant 200021-132510/1 and by VEGA grants 1/0726/09 and 1/0979/12.

The exact advantage of the offline algorithm can be measured as the amount of additional information, "advice", the online algorithm needs in order to produce an optimal solution. This notion can be formalized by providing the online algorithm with an additional source of information: an advice tape. The content of the advice tape is assumed to be prepared in advance by an oracle which has unlimited computational power and has access to the whole input instance. This model of advice complexity has first been introduced in [6].

The original model [6] had a minor technical disadvantage: the advice tape was finite, hence its length could be used to encode additional information. There were two attempts to fix this. In [8], the authors force the algorithm to use a fixed number of advice bits in every turn. The drawback of this approach is that it is not possible to determine sublinear advice complexity. A better version of the model was proposed in [3]. In this version the advice tape is considered to be infinite, and we consider the length of its prefix that is actually examined by the online algorithm. We are using this model in our paper.

There are indeed problems where already a small amount of information is enough for an online algorithm to be optimal. For example, the online problem SkiRental needs only 1 bit for being optimal, see [4,6]. In recent years, advice complexity was also investigated for other classical online problems, such as the k-server problem [2] and the paging problem [13].

Advice complexity has its theoretical importance in measuring an exact quantity of information that directly characterizes the hardness of an online problem compared to its offline version. It is also worth noting that advice complexity is closely related to both non-determinism (in terms of the oracle) and randomization. In [14] relations between advice complexity and randomized algorithms are shown, and a new randomized algorithm is designed based on a careful computation of the advice complexity of a given problem. This makes advice complexity a new and important point of view on online algorithms.

In this paper we present the initial results of advice complexity for a very important online problem: online graph coloring. The general offline version of this problem is a well-known NP-complete problem, and all online algorithms without advice are known to be extremely poor in the worst case [11]. This huge difference between online and offline algorithms makes this problem extremely interesting from the advice complexity point of view. Our goal in this paper is to provide an exact analysis of its simplest versions. In particular, we analyze online 2-coloring of paths. We also consider a more general version of graph coloring, where only a subset of vertices is colored online. Many such problems on subgraphs, including path coloring, were considered in [1] and are known to be very hard to approximate. Finally, we outline the approach that should be taken when analyzing more general versions of online graph coloring.

## 2 Definitions

### 2.1 Advice Complexity Definitions

In this part of our paper we provide the necessary subset of definitions from [3].

**Definition 1.** *An online algorithm $A$ with advice is defined as follows: The input for the algorithm is a sequence $(x_1, \ldots, x_n)$ and an infinite advice word $\varphi \in \{0,1\}^\omega$. The algorithm produces an output sequence $(y_1, \ldots, y_n)$ with the restriction that $\forall i : y_i$ is computed only from $x_1, \ldots, x_i$ and $\varphi$.*

The computation of $A$ can be seen as a sequence of turns, where in $i$-th turn $A$ reads $x_i$ and then produces $y_i$ using all the information read so far and possibly some new bits of the advice word. Note that the definition does not restrict the computational power of the online algorithms. Still, all of the algorithms in our paper will be deterministic and they will all have a polynomial time complexity.

**Definition 2.** *The advice complexity of $A$ is a function $s$ such that $s(n)$ is the smallest value such that for no input sequence of size $n$ the algorithm $A$ examines more than the first $s(n)$ bits of the advice word $\varphi$.*

**Definition 3.** *The advice complexity of an online problem is the smallest advice complexity an online algorithm with advice needs to produce an optimal solution (i.e., a solution as good as an optimal offline algorithm would produce).*

In [3] the authors also provide definitions for online approximation algorithms with advice. For these algorithms, one can look at the tradeoff between the amount of advice available and the competitive ratio of the algorithm. Sometimes already very little advice can reduce the competive ratio significantly. One example is the paging problem, also considered in [3]. In our paper we only consider algorithms that produce optimal solutions, hence we omit these definitions. However, in the conclusion of this paper we show that this question will be very interesting when our problem is considered for more general classes of graphs.

## 2.2 Online Coloring Definitions

**Definition 4.** *In* OnlineColoring *the instance is an undirected graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$. This graph is presented to an online algorithm in turns: In $k$-th turn the online algorithm receives the graph $G_k = G[\,\{1, 2, \ldots, k\}\,]$, i.e., a subgraph of $G$ induced by the vertex set $\{1, 2, \ldots, k\}$. As its reply, the online algorithm must return a positive integer: the color it wants to assign to vertex $k$. The goal is to produce the optimal coloring of $G$ – the online algorithm must assign distinct integers to adjacent vertices, and the largest integer used must be as small as possible.*

Vertex labels in $G$ correspond to the order in which the online algorithm is asked to color the vertices. An equivalent definition: in the $k$-th turn the online algorithm is given a list of edges between $k$ and vertices in $\{1, 2, \ldots, k-1\}$.

Note that the value $n$ is not known a priori by the online algorithm. This is intentional and necessary. Announcing the value $n$ to the online algorithm would give it additional information about the instance it will be processing, and this amount of information would then not be reflected in the advice complexity.

**Definition 5.** OnlinePartialColoring *is a generalization of* OnlineColoring. *In this case, the instance is $G$ along with an integer sequence $(a_1, \ldots, a_m)$ such that $1 \leq a_1 < \cdots < a_m \leq n$.*

*For a given integer $k$, let $\overline{G_k} = G[\{a_1, a_2, \ldots, a_k\}]$ be the subgraph induced by $\{a_1, \ldots, a_k\}$. The graph $G_k$ can be obtained from $\overline{G_k}$ by changing vertex labels from $(a_1, \ldots, a_k)$ to $(1, \ldots, k)$. In turn $k$, the online algorithm is shown the graph $G_k$ and it must return the color assigned to the most recently added vertex (the one with label $k$ in $G_k$ and label $a_k$ in $G$). The goal is to produce a coloring of $G[\{a_1, a_2, \ldots, a_m\}]$ that can be extended to an optimal coloring of $G$.*

This problem can be seen as a parametrized trade-off between the online and offline version of the coloring problem. We start with an uncolored graph $G$. First, a part of $G$ is colored in an online fashion. At the end, the rest of $G$ is colored offline. (By setting $m = 0$ in OnlinePartialColoring we obtain the offline version, and for $m = n$ we get the online version of classic coloring.)

Neither $n$ nor $m$ are known by the online algorithm. The relabeling of vertices prevents the online algorithm from deducing their location in $G$.

Both OnlineColoring and OnlinePartialColoring can be considered not only for general graphs, but also for specific graph classes – the online player will then know that the entire graph $G$ belongs to the considered class of graphs. In this paper, we will be considering these two online problems for two of the simplest graph classes – arbitrarily numbered and sequentially numbered paths. (In some literature, these are denoted "paths with arbitrary presentation" and "paths with connected presentation". For many graph classes, OnlineColoring is hard even with connected presentation. See [5] for more.)

**Definition 6.** *An arbitrary numbered path with $n$ vertices is a graph with vertices $\{1, \ldots, n\}$, $n - 1$ edges and max. degree 2. A sequentially numbered path with $n$ vertices is a graph with vertices $\{1, \ldots, n\}$ and edges $\{(i, i+1) | 1 \leq i < n\}$.*

We will be analyzing the advice complexity of these problems given that $G$ comes from one of these graph classes and that the online algorithm is deterministic. Already for these simple cases the results will be non-trivial.

Note that for any particular class of graphs, an algorithm with advice that solves OnlinePartialColoring can be directly used to solve OnlineColoring. But on the other hand, we will show that for sequentially numbered paths OnlinePartialColoring requires strictly more advice than OnlineColoring. (Here it is a slightly artificial difference, as OnlineColoring needs no advice, but we expect to see similar differences in more complex graph classes.)

## 3 Theorems

In this section we list our results stated as theorems.

**Theorem 1.** *For an arbitrary path on $n$ vertices the advice complexity of both* OnlineColoring *and* OnlinePartialColoring *is exactly $\lceil n/2 \rceil - 1$ bits.*

**Theorem 2.** *For a sequentially numbered path on $n$ vertices the advice complexity of* OnlineColoring *is zero and the advice complexity of* OnlinePartialColoring *is $\beta n + O(\log n)$, where $\beta \approx 0.40568523$ is the binary logarithm of the plastic constant.*

In all proofs, $n$ is implicitly used as the number of vertices of $G$ for the given instance. More details on $\beta$, including its closed form, are given in Section 5. The notation "lg" in the proofs is the base-2 (binary) logarithm.

## 4 Proof of Theorem 1

**Lemma 1.** *An online algorithm with advice that solves* OnlinePartialColoring *and knows that $G$ is 2-colorable never needs to access advice bits whenever the degree of the currently processed vertex $k$ in the current graph $G_k$ is positive.*

*Proof.* If the degree of $k$ in $G_k$ is positive, in $G$ the vertex $a_k$ must be adjacent to some $a_i$ such that $i < k$. The online algorithm already assigned a color to $a_i$, and now it must use the other color for $a_k$. This can be done without advice. □

**Lemma 2.** *There is a deterministic online algorithm solving* OnlinePartialColoring *for arbitrary paths with advice complexity $\lceil n/2 \rceil - 1$.*

*Proof.* As suggested by Lemma 1, our algorithm only asks for advice (i.e., reads the next bit of the advice word) whenever the current vertex $k$ is isolated in $G_k$. One bit of advice is sufficient – the advice can be interpreted as the correct color to use. The above only applies for $k > 1$, as we may pick an arbitrary color for the first isolated vertex.

Let $S$ be the set of vertices that were isolated at the moment we processed them. Clearly, it follows that no two of them are adjacent in $G$, hence $S$ is an independent set in $G$ and therefore $|S| \leq \lceil n/2 \rceil$. □

**Lemma 3.** *Any deterministic online algorithm solving* OnlineColoring *for arbitrary paths needs at least $\lceil n/2 \rceil - 1$ bits of advice in the worst case.*

Note that the proof of the lower bound of $\lfloor n/2 \rfloor - 1$ bits is reasonably simple; for odd $n$ we have to use a more careful analysis to force the extra bit.

*Proof.* We will denote the vertices $v_1, \ldots, v_n$ in the order in which they appear on the path: for all $i$ the vertices $v_i$ and $v_{i+1}$ are adjacent. Note that we do not know the exact numbers of these vertices. (Each path produces two such sequences. But in our proof we will only consider sequences where $v_1 = 1$, so different sequences $(v_1, \ldots, v_n)$ will indeed correspond to different graphs $G$.)

Let $k = \lfloor n/2 \rfloor$. The graph $G_k$ will be called the prefix of an instance. We will only consider instances where the prefix consists of $k$ isolated vertices. Out of these instances, we will pick a set of instances $S$ with the following property: for no two instances in $S$ can their prefixes be colored in the same way. (Note that each instance has exactly two valid colorings, hence the prefix of each instance

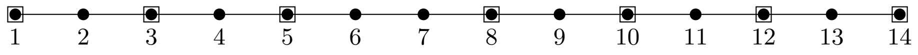


**Fig. 1.** Example for $n = 14$ and $x = 3$: $P_x = \{1, 3, 5\}$ and $Q_x = \{8, 10, 12, 14\}$

also has exactly two valid colorings – one a complement of the other.) As for a deterministic algorithm the prefixes of instances in $S$ are indistinguishable, all information about their correct coloring must be given as advice.

For any $x$ ($1 \le x \le \lfloor n/2 \rfloor$) consider the two sets of positions on the path $P_x = \{2i - 1 \mid 1 \le i \le x\}$ and $Q_x = \{2i \mid x + 1 \le i \le \lfloor n/2 \rfloor\}$ (see Fig. 1). Note that all vertices $v_i$ for $i \in P_x$ must share one color, and all vertices $v_j$ for $j \in Q_x$ must share the other color. (Also note, that $P_x$ and $Q_x$ are sets of indices of vertices $v_i$ and not their labels.)

Let $I_x$ be the set of all instances where the vertices on positions in $P_x \cup Q_x$ form the prefix. Formally, in these instances $\forall i \in P_x \cup Q_x : v_i \le k$. Note that for any such instance the prefix indeed consists of $k$ isolated vertices.

We will now define the set $S$: Consider all strings $w \in \{\mathtt{p}\} \cdot \{\mathtt{p}, \mathtt{q}\}^{k-1}$. For each such string, we pick into $S$ a single instance: Let $x$ be the number of ps in $w$. We will pick the lexicographically smallest[1] instance from $I_x$ such that $\forall i \le n$ : if $v_i \le k$, then ($i \in P_x$ iff the $i$-th letter of $w$ is $\mathtt{p}$). In words: The string $w$ determines the value of $x$ and gives the order in which vertices from $P_x$ and $Q_x$ are picked for coloring. There is always at least one such instance; if there are multiple, any will do, so we pick the lexicographically smallest one.

In this way we constructed a set $S$ of $2^{k-1}$ instances ($S$ contains one instance per each string $\{\mathtt{p}\} \cdot \{\mathtt{p}, \mathtt{q}\}^{\mathtt{k}-1}$) such that for no two instances in $S$ the prefix can be colored in the same way. By the pigeon-hole principle, if there was a deterministic online algorithm that always uses less than $k - 1$ bits of advice, two of the instances would receive the same advice, hence the algorithm would produce the same coloring of their prefixes, which is a contradiction. Therefore any deterministic online algorithm needs at least $k - 1 = \lfloor n/2 \rfloor - 1$ bits of advice. That concludes the proof for even $n$.

For odd $n$, we want to prove that any deterministic online algorithm must use at least $k$ bits of advice. By contradiction. Assume that the algorithm always uses less than $k$ bits of advice for paths of length $n$. This means that on instances from $S$ the algorithm always reads all $k - 1$ bits of advice, and different instances in $S$ must correspond to different $k - 1$ bits of advice.

In $S$ we have an instance $J_1$ that corresponds to $\mathtt{p}^{\mathtt{k}}$. For this instance all vertices in the prefix must receive the same color. Let $\varphi_1$ be the first $k - 1$ bits of advice for this instance. Consider any instance (possibly with more than $n$ vertices) such that $G_k$ consists of isolated vertices that should receive the same color as $J_1$. Clearly, for any such instance the first $k - 1$ bits of advice must be $\varphi_1$ – otherwise the deterministic algorithm would color the first $k$ vertices in a different way.

Now consider one additional instance $J_2$: the lexicographically smallest one where $v_i \le k + 1$ iff $i$ is odd. (This instance is similar to $J_1$, but in $J_1$ we have

[1] I.e., one for which the vector $(v_1, v_2, \dots)$ is the smallest.

$v_2 = k+1$ and in $J_2$ we have $v_n = k+1$. For $J_2$ the graph $G_{k+1}$ has $k+1$ isolated vertices.) As our algorithm never uses $k$ bits of advice, it must process $v_n$ without any additional advice. Thus the instances $J_1$ and $J_2$ are for the algorithm undistinguishable and the algorithm cannot use extra bits of advice to color $J_2$. Hence whenever our algorithm is presented with an instance (of any size) such that the first $k$ vertices are isolated and must share the same color, it will color the next isolated vertex using the same color. And this is a contradiction: we can easily create an instance of size $n+1$ where the $(k+1)$-st isolated vertex should have the opposite color. □

**Proof of Theorem 1.** By Lemma 3, any deterministic online algorithm for OnlineColoring needs at least $\lceil n/2 \rceil - 1$ bits of advice in the worst case. As OnlinePartialColoring is a generalization of OnlineColoring, this lower bound transfers to OnlinePartialColoring. By Lemma 2 there is a deterministic online algorithm solving OnlinePartialColoring with $\lceil n/2 \rceil - 1$ bits of advice, hence that is the exact advice complexity for both problems. □

## 5 Proof of Theorem 2

**Lemma 4.** OnlineColoring *for sequentially numbered paths can be solved without advice.*

*Proof.* Trivially follows from Lemma 1. □

Before we give the proof for OnlinePartialColoring, we first note some trivial upper and lower bounds. The algorithm shown in Lemma 2 gives us an upper bound of $\lceil n/2 \rceil - 1$ bits of advice. As we show below, this is not optimal. A trivial lower bound of $\lfloor n/3 \rfloor - 1$ bits of advice is obtained by considering instances where $m = \lfloor n/3 \rfloor$, $a_1 = 1$ and $a_i \in \{3i-2, 3i-1\}$ for $1 < i \leq m$. These are $2^{m-1}$ instances, and the algorithm needs to receive different advice for all of them, hence $\lg 2^{m-1} = m-1$ bits of advice are necessary.

**Lemma 5.** *For any positive integer $s$ and real number $r \geq 1$*

$$\frac{1}{4rs} \cdot \left( \frac{(r+1)^{r+1}}{r^r} \right)^s \leq \binom{(r+1)s}{s} \leq \left( \frac{(r+1)^{r+1}}{r^r} \right)^s .$$

*Proof.* These bounds are given in [15]. □

**Lemma 6.** *Every deterministic online algorithm that solves* OnlinePartialColoring *for sequentially numbered paths needs at least $\beta n - \lg n + O(1)$ bits of advice, where $\beta \approx 0.40568523$ is the binary logarithm of the plastic constant.*

*Proof.* We will describe a special set $S$ of instances. In all these instances $m$ will be the same and $a_{i+1} \geq a_i + 2$ for all $i$ – hence all queries will be isolated vertices. No two instances in $S$ will admit the same coloring, hence the amount of necessary advice bits will be bounded from below by $\lceil \lg |S| \rceil$.

The set $S$ will be formed by all instances that have $a_1 = 1$ and $a_{i+1} \in \{a_i + 2, a_i + 3\}$, for $i \geq 1$, with the additional restriction that in all instances there are exactly $k$ values $i$ such that $a_{i+1} = a_i + 2$ and exactly $l$ other values $i$. Hence $|S| = \binom{k+l}{k}$, $m = k + l + 1$, and the number of vertices in each instance is $n = 2k+3l+1$. (No two of these instances admit the same coloring: $a_{i+1} = a_i + 3$ always forces a color change.) To make formulas simpler, let $x = n - 1$.

We are now looking for the values $k$ and $l$ such that the number of such instances is maximized. Mathematical intuition suggests $k = l$ as a likely optimum, but this turns out to be false. (But note that taking $k = l$ leads to a correct and good lower bound of $0.4x$ bits of advice.)

For a fixed $n = 2k + 3l + 1$ we are maximizing $\binom{k+l}{l}$. Let $l = \alpha x$ for some $\alpha$, hence $k = (1 - 3\alpha)x/2$ and we are maximizing $\binom{(1-\alpha)x/2}{\alpha x}$ as a function of $\alpha \in (0, 1)$. It is easily verified that the function is decreasing for $\alpha \in [1/5, 1)$ (which corresponds to the case $k < l$). Hence we just consider $\alpha \in (0, 1/5]$.

In Lemma 5, let $s = l$ and $r = k/l = (x - 3l)/(2l) = (1 - 3\alpha)/(2\alpha)$. (Note that $r \geq 1$.) This bounds our binomial coefficient from below:

$$\begin{aligned}
\binom{k+l}{l} &= \binom{(1-\alpha)x/2}{\alpha x} \\
&\geq \frac{2\alpha}{4(1-3\alpha)\alpha x} \cdot \left( \frac{\left(\frac{1-\alpha}{2\alpha}\right)^{\frac{1-\alpha}{2\alpha}}}{\left(\frac{1-3\alpha}{2\alpha}\right)^{\frac{1-3\alpha}{2\alpha}}} \right)^{\alpha x} \\
&= \frac{1}{2(1-3\alpha)x} \cdot \left( \frac{(1-\alpha)^{(1-\alpha)/2}}{(1-3\alpha)^{(1-3\alpha)/2} \cdot (2\alpha)^{\alpha}} \right)^{x} = f(\alpha)
\end{aligned}$$

Hence we get the lower bound: $\max_{0<\alpha\leq 1/5} \binom{(1-\alpha)x/2}{\alpha x} \geq \max_{0<\alpha\leq 1/5} f(\alpha)$.

When constructing the set $S$, we may pick the value $\alpha$ that maximizes $f(\alpha)$, thereby ensuring that any deterministic online algorithm will need at least $\lceil \lg \max_{0<\alpha\leq 1/5} f(\alpha) \rceil$ bits of advice. We may now compute:

$$\begin{aligned}
&\lg \max_{0<\alpha\leq 1/5} f(\alpha) \\
&= \lg \max_{0<\alpha\leq 1/5} \frac{1}{2(1-3\alpha)x} \cdot \left( \frac{(1-\alpha)^{(1-\alpha)/2}}{(1-3\alpha)^{(1-3\alpha)/2} \cdot (2\alpha)^{\alpha}} \right)^{x} \\
&= \lg \frac{1}{2x} + \max_{0<\alpha\leq 1/5} \left( -\lg(1-3\alpha) + x \cdot \lg \left( \frac{(1-\alpha)^{(1-\alpha)/2}}{(1-3\alpha)^{(1-3\alpha)/2} \cdot (2\alpha)^{\alpha}} \right) \right) \\
&\geq \lg \frac{1}{2x} - \lg \min_{0<\alpha\leq 1/5} (1-3\alpha) + x \cdot \lg \max_{0<\alpha\leq 1/5} \underbrace{\left( \frac{(1-\alpha)^{(1-\alpha)/2}}{(1-3\alpha)^{(1-3\alpha)/2} \cdot (2\alpha)^{\alpha}} \right)}_{g(\alpha)} \\
&= -1 - \lg x - \lg(2/5) + \beta x \qquad \text{(see below for the value of } \beta\text{)}
\end{aligned}$$

Let $\alpha_m$ be the value of $\alpha$ for which the last expression, denoted $g(\alpha)$, is maximized. A numerical computation showed that $\alpha_m \approx 0.17700882$, which is indeed in the correct range (hence indeed $l < k$). The value of the maximized expression is $g(\alpha_m) \approx 1.3247179572$, hence its binary logarithm is $\beta = \lg g(\alpha_m) \approx 0.40568523$.

Using these approximate values we were able to guess and then to verify closed forms for these constants. All three constants are related to a well-known mathematical constant $P$, often called the plastic constant – see [9] for more details. The constant $P$ is the only real root of the polynomial $x^3 - x - 1$. In terms of $P$, we can express our constants as $\alpha_m = 1/(2P+3)$ and $g(\alpha_m) = P$, therefore we obtain $\beta = \lg\left(\sqrt[3]{9-\sqrt{69}} + \sqrt[3]{9+\sqrt{69}}\right) - \lg \sqrt[3]{18}$.

Note that in the calculation of this lower bound we worked with possibly non-integer values for $k$ and $l$. For an actual lower bound, we should take their floors. Clearly, this only decreases the required advice by a constant. Hence we may conclude that the advice complexity of solving OnlinePartialColoring on a sequentially numbered path with $n$ vertices is at least $\beta n - \lg n + O(1)$. □

Before we prove a very close upper bound, we repeat that the naive strategy "each time a vertex needs coloring, the advice is the color" is not optimal. It turns out that the instances identified in the lower bound are precisely the worst case. A general idea of the proof can be summarized as follows: whenever $m$ is close to $n/2$ (hence the instance is dense and we would need too much advice in the naive approach), the number of color changes in the instance is necessarily small, and we may only encode their locations as advice.

**Lemma 7.** *There is a deterministic online algorithm that solves* OnlinePartialColoring *for sequentially numbered paths and never uses more than* $\beta n + 2\lg n + \lg\lg n + O(1)$ *bits of advice, where* $\beta$ *is exactly the same constant as in Lemma 6 and before.*

*Proof.* The general outline of this proof: We will transform each instance to an instance where all queries are isolated vertices that are as close to the left of the path as possible, given the colors they are supposed to receive. The advice will then be a triple containing $n$; the number of color changes in the (optimal) coloring; and the index of the lexicographically smallest instance with the same coloring as our given instance has. We will prove that this will indeed solve the problem, and that the number of advice bits matches the claim above.

To improve our trivial upper bound and match it with the lower bound presented in Lemma 6, we need to make a few observations. First of all, we only need to consider instances in which $a_{i+1} \geq a_i + 2$ for all $i$: the online algorithm can handle queries where $a_{i+1} = a_i + 1$ without advice, see Lemma 1.

Hence we only consider instances where the colored vertices form an independent set. Let $(a_1, \ldots, a_m)$ be the sequence of requests for one such instance $I$. Let $l$ be the number of times the value $a_{i+1} - a_i$ is odd, i.e., the number of color changes. Let $k = \lfloor (n-1-3l)/2 \rfloor$.

Now consider the set $S$ of instances constructed in the proof of Lemma 6 for the values $k$ and $l$ we just defined. Clearly, for our instance $I$ there is an instance

$I'$ in $S$ with the following property: the sequence of colors assigned to queries in $I$ is a prefix of the sequence of colors assigned to the queries in $I'$.

In fact, $I'$ can be easily constructed from $I$: We start with the sequence $(a_1, \dots, a_m)$ of queries in $I$. We subtract $a_1 - 1$ from all elements, getting a sequence that starts with 1. Then we process the values $a_i$ for $i = 2..n$, and for each of these values we change $a_i$ into $(a_{i-1} + 2 + (a_i - a_{i-1}) \bmod 2)$. (That is, we shift $a_i$ to the left so that the distance $a_i - a_{i-1}$ becomes either 2 or 3; we preserve parity of the distance.) Finally, if now the last request $a_m$ is less than $n - 1$, we append additional requests, each of them in distance two from the previous one.

If our deterministic online algorithm receives the instance $I$, the advice it will receive will consist of three parts: the value $n$, the value $l$, and a number $d$. To obtain $d$, we construct the correct set $S$ and order all its elements lexicographically; $d$ is the position of the instance $I'$ in this order.

The value $n$ can easily be encoded into the first $\lg n + \lg\lg n + 1$ bits of advice using a suitable prefix code, such as the Elias delta code [7]. As the deterministic online algorithm already knows $n$ before reading $l$ and $d$, we can avoid using a prefix code for these: after the bits representing $n$, the advice word will contain exactly $\lfloor \lg n \rfloor + 1$ bits for $l$ and another $\lceil \lg |S| \rceil$ bits for $d$.

(We note that the computational power of computing advice is not considered in the advice complexity model, and neither is the time complexity of the online algorithm. However, the value $d$ can in fact be computed in time polynomial in $n$, and also reconstructing $I'$ from $d$ can be done in polynomial time using suitable combinatorial algorithms. As this is not the scope of our paper, we omit these algorithms.)

We see that the amount of advice needed depends on the size of the largest possible set $S$. To compute it we need to bound the same value $\binom{k+l}{l} = \binom{(1-\alpha)x/2}{\alpha x}$ as in the previous Lemma, but this time from above. Again, we apply Lemma 5:

$$\begin{aligned}\binom{k+l}{l} &= \binom{(1-\alpha)x/2}{\alpha x} \\ &\le \left( \frac{\left(\frac{1-\alpha}{2\alpha}\right)^{\frac{1-\alpha}{2\alpha}}}{\left(\frac{1-3\alpha}{2\alpha}\right)^{\frac{1-3\alpha}{2\alpha}}} \right)^{\alpha x} \\ &= \left( \frac{(1-\alpha)^{(1-\alpha)/2}}{(1-3\alpha)^{(1-3\alpha)/2} \cdot (2\alpha)^{\alpha}} \right)^{x} = g(\alpha)^x\end{aligned}$$

Hence we get the upper bound:

$$\max_{0<\alpha\le 1/5} \binom{(1-\alpha)x/2}{\alpha x} \le \max_{0<\alpha\le 1/5} g(\alpha)^x$$

And using the same computations as in Lemma 6 we arrive at the conclusion:

$$\lg \max_{0<\alpha\le 1/5} g(\alpha)^x = x \cdot \lg \max_{0<\alpha\le 1/5} \left( \frac{(1-\alpha)^{(1-\alpha)/2}}{(1-3\alpha)^{(1-3\alpha)/2} \cdot (2\alpha)^{\alpha}} \right) = \beta x,$$

where $\beta$ is the same constant as in Lemma 6. As in the previous lemma, we may need to round the optimal $k$ and $l$ to integers (this time to ceilings of their actual values), which may increase $2k + 3l$ by a constant, hence it may increase the number of advice bits by a constant. Therefore $\beta n + 2 \lg n + \lg \lg n + O(1)$ bits of advice are sufficient. □

**Proof of Theorem 2.** Follows trivially from Lemma 6 and Lemma 7. □

## 6 Conclusion

In the paper we showed that already the simplest versions of online graph coloring are non-trivial in terms of advice complexity. We analyzed two such versions and obtained exact (in one case) and almost exact (in the other case) bounds on their advice complexity.

One direction of future research is clear. All of the problems considered in this paper generalize to more complex graph classes. In particular, coloring of sequentially numbered paths generalizes to coloring of graphs numbered according to their depth-first or breadth-first traversal. It should be possible to generalize our results to more complex graph classes. We also expect that this point of view may lead to new randomized approximation algorithms and/or new inapproximability results for some graph classes.

However, for more complex graph classes a more fine-grained analysis should also be possible. We will now give an outline of its general idea. Note that for all problems presented in the paper there is a trivial online approximation algorithm that never uses more than three colors. Hence it does not make sense to consider online approximation algorithms – either the online algorithm gets it right, or it does not, in which case its competitive ratio is trivially $3/2$. However, as we move to more complex graph classes, the difference between online and offline algorithms increases. For instance, every tree is trivially 2-colorable, but it has been shown [10] that any online algorithm without advice can be forced to use $\Theta(\log n)$ colors on a $n$-vertex tree.

The advice complexity of obtaining the optimal coloring can be high. For instance, there are $n$-vertex trees with $n-2$ leaves, and for these we obviously need $n-3$ bits of advice in order to produce the optimal coloring. A natural question here is "what can we get for less?" That is, it should be possible to analyze the tradeoff between the amount of advice available to the online algorithm and the quality of the obtained approximation. This more detailed analysis may then also lead to new, interesting approximation algorithms.

Such analysis can be done for all of the important graph classes. We expect most of these problems to be hard, but worth solving – their solutions should give us a better understanding of online coloring.

**Acknowledgement.** The authors are thankful to Juraj Hromkovič for bringing this problem to their attention, and to Hans-Joachim Böckenhauer and Maria Paola Bianchi for the interesting and helpful discussions about this problem.

## References

1. Bartal, Y., Fiat, A., Leonardi, S.: Lower bounds for on-line graph problems with application to on-line circuit and optical routing. SIAM J. Comput. 36, 354–393 (2006)
2. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R.: On the Advice Complexity of the k-Server Problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 207–218. Springer, Heidelberg (2011)
3. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the Advice Complexity of Online Problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press (1998)
5. Cieślik, I.: On-line graph coloring. Ph.D. thesis, Jagiellonian University Kraków (2006)
6. Dobrev, S., Královič, R., Pardubská, D.: How Much Information About the Future is Needed? In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 247–258. Springer, Heidelberg (2008)
7. Elias, P.: Universal codeword sets and representations of the integers. IEEE Transactions on Information Theory 21, 194–203 (1975)
8. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online Computation with Advice. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S.E., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 427–438. Springer, Heidelberg (2009)
9. Finch, S.R.: Mathematical Constants (Encyclopedia of Mathematics and its Applications). Cambridge University Press (August 2003)
10. Gyrfs, A., Lehel, J.: On-line and first fit colorings of graphs. Journal of Graph Theory 12(2), 217–227 (1988)
11. Halldórsson, M.M., Szegedy, M.: Lower bounds for on-line graph coloring. In: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1992, pp. 211–216. Society for Industrial and Applied Mathematics, Philadelphia (1992)
12. Hromkovič, J.: Design and analysis of randomized algorithms. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2005)
13. Hromkovič, J., Královič, R., Královič, R.: Information Complexity of Online Problems. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 24–36. Springer, Heidelberg (2010)
14. Komm, D., Královič, R.: Advice Complexity and Barely Random Algorithms. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jeffery, K.G., Královič, R., Vukolic, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 332–343. Springer, Heidelberg (2011)
15. Sondow, J., Zudilin, W.: Eulers constant, q-logarithms, and formulas of Ramanujan and Gosper. The Ramanujan Journal 12, 225–244 (2006)

# A Faster Grammar-Based Self-index

Travis Gagie[1], Paweł Gawrychowski[2,⋆], Juha Kärkkäinen[3], Yakov Nekrich[4], and Simon J. Puglisi[5]

[1] Aalto University, Finland
[2] University of Wrocław, Poland
[3] University of Helsinki, Finland
[4] University of Bonn, Germany
[5] King's College, London, UK

**Abstract.** To store and search genomic databases efficiently, researchers have recently started building compressed self-indexes based on straight-line programs and LZ77. In this paper we show how, given a balanced straight-line program for a string $S[1..n]$ whose LZ77 parse consists of $z$ phrases, we can add $\mathcal{O}(z \log \log z)$ words and obtain a compressed self-index for $S$ such that, given a pattern $P[1..m]$, we can list the occ occurrences of $P$ in $S$ in $\mathcal{O}\big(m^2 + (m + \text{occ}) \log \log n\big)$ time. All previous self-indexes are either larger or slower in the worst case.

## 1 Introduction

With the advance of DNA-sequencing technologies comes the problem of how to store many individuals' genomes compactly but such that we can search them quickly. Any two human genomes are 99.9% the same, but compressed self-indexes based on compressed suffix arrays, the Burrows-Wheeler Transform or LZ78 (see [21] for a survey) do not take full advantage of this similarity. Researchers have recently started building compressed self-indexes based on context-free grammars (CFGs) and LZ77 [23], which better compress highly repetitive strings. A compressed self-index stores a string $S[1..n]$ in compressed form such that, given a pattern $P[1..m]$, we can quickly list the occ occurrences of $P$ in $S$.

Claude and Navarro [4] gave the first compressed self-index based on grammars, which takes $\mathcal{O}(r \log r) + r \log n$ bits, where $r$ is the number of rules in the given grammar, and $\mathcal{O}\big((m^2 + h(m + \text{occ})) \log r\big)$ time, where $h \geq \log n$ is the height of the parse tree. Throughout this paper, we write log to mean $\log_2$. Very recently, the same authors [5] described another grammar-based compressed self-index, which takes $2R \log r + R \log n + \epsilon r \log r + o(R \log r)$ bits, where $R$ is the total length of the rules' right-hand sides and $0 < \epsilon \leq 1$, and $\mathcal{O}\big((m^2/\epsilon) \log R + \text{occ} \log r\big)$ time[1]. Whereas their first index requires a straight-line program (SLP), their new index can be built on top of an arbitrary CFG.

⋆ Supported by MNiSW grant number N206 492638, 2010–2012.

[1] They have now reduced the $\log R$ factor in this time bound to $\log(\log n / \log r)$.

**Table 1.** Space and time bounds for Claude and Navarro's compressed self-indexes [4,5], Kreft and Navarro's [16] and our own. Throughout, $r$ is the number of rules in a given CFG generating a string $S[1..n]$ and only $S$, but the CFG must be an SLP in Row 1 and a balanced SLP in Row 4. In Row 1, $h$ is the height of the parse tree. In Row 2, $R$ is the total length of the rules' right-hand sides and $0 < \epsilon \leq 1$. In Row 3, $d$ is the depth of nesting in the parse.

| source | total space (bits) | search time |
|---|---|---|
| [4] | $\mathcal{O}(r \log r) + r \log n$ | $\mathcal{O}\big((m^2 + h(m + \mathrm{occ})) \log r\big)$ |
| [5] | $2R \log r + R \log n + \epsilon\, r \log r + o(R \log r)$ | $\mathcal{O}\big((m^2/\epsilon) \log R + \mathrm{occ} \log r\big)$ |
| [16] | $2z \log(n/z) + z \log z + 5z \log \sigma + \mathcal{O}(z) + o(n)$ | $\mathcal{O}\big(m^2 d + (m + \mathrm{occ}) \log z\big)$ |
| Thm. 4 | $2r \log r + \mathcal{O}(z(\log n + \log z \log\log z))$ | $\mathcal{O}\big(m^2 + (m + \mathrm{occ}) \log\log n\big)$ |

Kreft and Navarro [16] gave the first (and, so far, only) compressed self-index based on LZ77, which takes $2z \log(n/z)+z \log z+5z \log \sigma+\mathcal{O}(z)+o(n)$ bits, where $z$ is the number of phrases in the LZ77 parse of $S$, and $\mathcal{O}\big(m^2 d + (m + \mathrm{occ}) \log z\big)$ time, where $d \leq z$ is the depth of nesting in the parse. Throughout this paper, we consider the non-self-referential LZ77 parse, so $z \geq \log n$. The $o(n)$ term in Kreft and Navarro's space bound can be dropped at the cost of increasing the time bound by a factor of $\log(n/z)$. In this paper we show how, given a balanced SLP for $S$, we can add $\mathcal{O}(z(\log n + \log z \log\log z))$ bits and obtain a compressed self-index that answers queries in $\mathcal{O}\big(m^2 + (m + \mathrm{occ}) \log\log n\big)$ time. We note that Maruyama et al. [18] recently described another grammar-based index but its search time depends on "the number of occurrences of a maximal common subtree in [edit-sensitive parse] trees of $P$ and $S$", making it difficult to compare their data structure to those mentioned above.

In order to better compare our bounds, those by Claude and Navarro and those by Kreft and Navarro — all of which are summarized in Table 1 — it is useful to review some bounds for the LZ77, balanced and unbalanced SLPs, and arbitrary CFGs. The LZ77 compression algorithm works by parsing $S$ from left to right into phrases: after parsing $S[1..i-1]$, it finds the longest prefix $S[i..j-1]$ of $S[i..m]$ that has occurred before and selects $S[i..j]$ as the next phrase. The previous occurrence of $S[i..j-1]$ is called the phrase's source. Kreft and Navarro considered the non-self-referential LZ77 parse, in which each phrase's source must end before the phrase itself begins, so $z \geq \log n$. For ease of comparison, we also consider the non-self-referential parse.

In an SLP of size $r$, each of the non-terminals $X_1, \ldots, X_r$ appears on the left-hand side of exactly one rule and each rule is either of the form $X_i \rightarrow a$, where $a$ is a terminal, or $X_i \rightarrow X_j X_k$, where $i > j, k$; thus, the SLP generates exactly one string and can be encoded in $2r \log r + \mathcal{O}(r)$ bits. Rytter [22] and Charikar et al. [2] showed that, in every CFG generating $S$ and only $S$, the total length of the rules' right-hand sides is at least $z$. Rytter showed how we can build an SLP for $S$ with $\mathcal{O}(z \log n)$ rules whose parse tree has the shape of an AVL tree, which is height-balanced, and Charikar et al. showed how to build such an SLP whose parse tree is weight-balanced. Finally, Charikar et al. showed that finding an algorithm for building a CFG generating $S$ and only $S$ in which the total length of the rules' right-hand sides is $o(z \log n / \log \log n)$, would imply progress on the well-studied problem of building addition chains.

With Rytter's and Charikar et al.'s bounds in mind, it is clear that the time bound of our index is strictly better than those of the others. Kreft and Navarro's index has the best worst-case space bound but only when we drop the $o(n)$ term, increasing their time bound by a factor of $\log(n/z)$. As it is not known whether it is possible, regardless of the time taken, to build a CFG that generates $S$ and only $S$ and in which the total length of the rules' right-hand sides is $o(z \log n)$, it is also not known whether Claude and Navarro's space bounds are ever strictly better than ours. It is even conceivable that they could be worse: suppose $z = \log^{\mathcal{O}(1)} n$ and the minimum total length of the rules' right-hand sides in a CFG generating $S$ and only $S$, is $z \log n$; then the $\mathcal{O}(r \log n)$ and $R \log n$ terms in Claude and Navarro's space bounds are $\Theta(z \log^2 n)$, while the dominant term in our own space bound is $2r \log r = \Theta(z \log n \log \log n)$.

To build our index, we add bookmarks to the SLP that allow us to extract substrings quickly from around boundaries between phrases in the LZ77 parse. In Section 2 we show that a bookmark for a position $b$ takes $\mathcal{O}(\log n)$ bits and, for any length $\ell$, allows us to extract $S[b - \ell..b + \ell]$ in $\mathcal{O}(\ell + \log \log n)$ time. Kreft and Navarro showed how, if we can extract substrings quickly from around phrase boundaries, then we can quickly list all occurrences of $P$ that finish at or cross those boundaries, which are called primary occurrences. They first use two Patricia trees and access at the boundaries to find, for each position $i \geq 1$ in $P$, the lexicographic range of the reverses of phrases ending with $P[1..i]$, and the lexicographic range of suffixes of $S$ starting with $P[i + 1..m]$ at phrase boundaries. They then use a wavelet tree as a data structure for 2-dimensional range reporting to find all the phrase boundaries immediately preceded by $P[1..i]$ and followed by $P[i+1..m]$. In Section 3 we show how, using bookmarks for faster access and a faster (albeit larger) range-reporting data structure, we can list all primary occurrences in $\mathcal{O}\big(m^2 + (m + \mathrm{occ}) \log \log n\big)$ time. Occurrences of $P$ that neither finish at nor cross phrase boundaries are called secondary and they can be found recursively from the primary occurrences. We build a data structure for 2-sided range reporting on an $n \times n$ grid on which we place a point $(i, j)$ for each phrase's source $S[i..j]$. Since the queries are 2-sided, this data structure takes $\mathcal{O}(z \log n)$ bits and answers queries in $\mathcal{O}(\log \log n + p)$ time, where $p$ is the number of points reported. If a phrase contains a secondary occurrence at a certain position in the phrase, then that phrase's source must also contain an occurrence at the corresponding position. It follows that, by querying the data structure with $(a, b)$ for each primary or secondary occurrence $S[a..b]$ we find, we can list all secondary occurrences of $P$ in $\mathcal{O}(\mathrm{occ} \log \log n)$ time.

## 2 Adding Bookmarks to a Balanced SLP

An SLP for $S$ is called balanced if its parse tree is height- or weight-balanced. We claim that, for $1 \leq i \leq j = i + 2g \leq n$, we can choose two nodes $v$ and

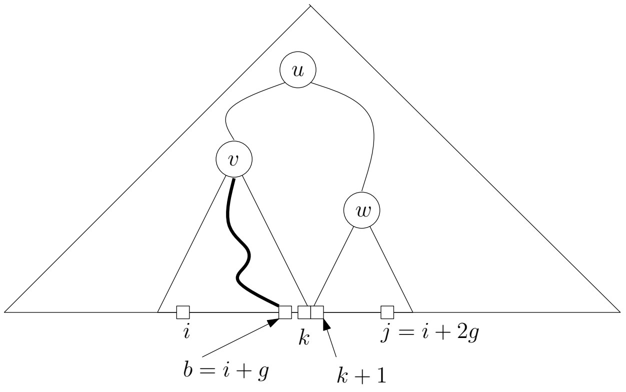


**Fig. 1.** Suppose we are given an SLP for $S$ and consider its parse tree. For $1 \leq i \leq j = i + 2g \leq n$, we can choose two nodes $v$ and $w$ of height $\mathcal{O}(\log g)$ in the parse tree whose subtrees include the $i$th through $j$th leaves. Without loss of generality, assume the $b$th leaf is in $v$'s subtree, where $b = i + g$. By storing the non-terminals at $v$ and $w$ and the path from $v$ to $b$, we can extract $S[b - \ell..b + \ell]$ in $\mathcal{O}(\ell + \log g)$ time.

$w$ of height $\mathcal{O}(\log g)$ in the parse tree whose subtrees include the $i$th through $j$th leaves. To see why, let $u$ be the lowest common ancestor of the $i$th and $j$th leaves. If $u$ has height $\mathcal{O}(\log g)$, then we choose its children as $v$ and $w$; otherwise, suppose the rightmost leaf in $u$'s left subtree is the $k$th in the tree. We choose as $v$ the lowest common ancestor of the $i$th through $k$th leaves, which must lie on the rightmost path in $u$'s left subtree. Since $v$'s right subtree has fewer than $k - i + 1 \leq 2g$ leaves, $v$ has height $\mathcal{O}(\log g)$. We choose as $w$ the lowest common ancestor of the $(k + 1)$st through $j$th leaves, which must lie on the leftmost path in $u$'s right subtree. Since $w$'s left subtree has fewer than $j - k \leq 2g$ leaves, $w$ also has height $\mathcal{O}(\log g)$. Figure 1 illustrates our choice of $v$ and $w$.

Without loss of generality, assume $b = i + g \leq k$; the case when $b > k$ is symmetric. Then the $b$th leaf — which is midway between the $i$th and $j$th — is in $v$'s subtree and we can store the path from $v$ to the $b$th leaf in $\mathcal{O}(\log g)$ bits. With this information, in $\mathcal{O}(\ell + \log g)$ time we can perform partial depth-first traversals of $v$'s subtree that start by descending from $v$ to the $b$th leaf and then visit the $\ell \leq g$ leaves immediately to its left and right, if they are in $v$'s subtree. Some of the $\ell$ leaves immediately to right of the $b$th leaf may be the leftmost leaves in $w$'s subtree, instead; we can descend from $w$ and visit them in $\mathcal{O}(\ell + \log g)$ time. It follows that, if we store the non-terminals at $v$ and $w$ and $\mathcal{O}(\log g)$ extra bits then, given $\ell \leq g$, we can extract $S[b - \ell..b + \ell]$ in $\mathcal{O}(\ell + \log g)$ time.

**Lemma 1.** *Given a balanced SLP for $S$ with $r$ rules and integers $b$ and $g$, we can store $2 \log r + \mathcal{O}(\log g)$ bits such that later, given $\ell$, we can extract $S[b - \ell..b + \ell]$ in $\mathcal{O}(\ell + \log g)$ time.*

The drawback to Lemma 1 is that we must know in advance an upper bound on the number of characters we will want to extract. To remove this restriction, for each position we wish to bookmark, we apply Lemma 1 twice: the first time, we set $g = n$ so that, given $\ell \geq \log n$, we can extract $S[b - \ell..b + \ell]$ in $\mathcal{O}(\ell)$ time; the second time, we set $g = \log n$ so that, given $\ell \leq \log n$, we can extract $S[b - \ell..b + \ell]$ in $\mathcal{O}(\ell + \log \log n)$ time.

**Theorem 2.** *Given a balanced SLP for $S$ and a position $b$, we can store $\mathcal{O}(\log n)$ bits such that later, given $\ell$, we can extract $S[b-\ell..b+\ell]$ in $\mathcal{O}(\ell + \log \log n)$ time.*

We note that, without changing the asymptotic space bound in Theorem 2, the time bound can be improved to $\mathcal{O}\left(\ell + \log^{[c]} n\right)$, where $c$ is any constant and $\log^{[c]} n$ is the result of recursively applying the logarithm function $c$ times to $n$. If we increase the space bound to $\mathcal{O}(\log n \log^* n)$, where $\log^*$ is the iterated logarithm, then the time bound decreases to $\mathcal{O}(\ell)$. Even as stated, however, Theorem 2 is enough for our purposes in this paper as we use only the following corollary, which follows by applying Theorem 2 for each phrase boundary in the LZ77 parse.

**Corollary 3.** *Given a balanced SLP for $S$, we can add $\mathcal{O}(z \log n)$ bits such that later, given $\ell$, we can extract $\ell$ characters to either side of any phrase boundary in $\mathcal{O}(\ell + \log \log n)$ time.*

As an aside, we note that Rytter's construction produces an SLP in which, for each phrase in the LZ77 parse, there is a non-terminal whose expansion is that phrase. For any balanced SLP with this property, it is easy to add bookmarks such that we can extract quickly from phrase boundaries: for each phrase, we store the non-terminal generating that phrase and the leftmost and rightmost non-terminals at height $\log \log n$ in its parse tree.

## 3 Listing Occurrences

As explained in Section 1, we use the same framework as Kreft and Navarro but with faster access at the phrase boundaries, using Corollary 3 and a faster data structure for 2-dimensional range reporting. Given a pattern $P[1..m]$, for $1 \leq i \leq m$, we find all the boundaries where the previous phrase ends with $P[1..i]$ and the next phrase starts with $P[i + 1..m]$. To do this, we build one Patricia tree for the reversed phrases and another for the suffixes starting at phrase boundaries. A Patricia tree [20] is a compressed trie in which the length of each edge label is recorded and then all but the first character of each edge label are discarded. It follows that the two Patricia trees have $\mathcal{O}(z)$ nodes and take $\mathcal{O}(z \log n)$ bits; notice that no two suffixes can be the same and, by the definition of the LZ77 parse, no two reversed phrases can be the same. We label each leaf in the first Patricia tree by the number of the phrase whose reverse leads to that leaf; we label each leaf in the second Patricia tree by the number of the phrase starting the suffix that leads to that leaf.

For example, if $S =$ "alabar_a_la_alabarda\$", then its LZ77 parse is as shown in Figure 2 (from Kreft and Navarro's paper). For convenience, we treat all

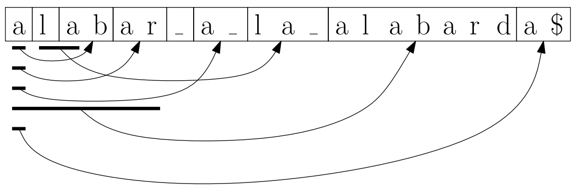


**Fig. 2.** The LZ77 parse of "alabar_a_la_alabarda$", from Kreft and Navarro's paper [16]. Horizontal lines indicate phrases' sources, with arrows leading to the boxes containing the phrases themselves.

strings as ending with a special character \$. In this case, the reversed phrases are as shown below with the phrase numbers, in order by phrase number on the left and in lexicographic order on the right.

| | |
|---|---|
| 1) a$ | 9) $a$ |
| 2) l$ | 5) _$ |
| 3) ba$ | 6) _a$ |
| 4) ra$ | 7) _al$ |
| 5) _$ | 1) a$ |
| 6) _a$ | 3) ba$ |
| 7) _al$ | 8) drabala$ |
| 8) drabala$ | 2) l$ |
| 9) $a$ | 4) ra$ |

The suffixes starting at phrase boundaries are as shown below with the numbers of the phrases with which they begin, in order by phrase number on the left and in lexicographic order on the right.

| | |
|---|---|
| 2) labar_a_la_alabarda$ | 5) _a_la_alabarda$ |
| 3) abar_a_la_alabarda$ | 9) a$ |
| 4) ar_a_la_alabarda$ | 6) a_la_alabarda$ |
| 5) _a_la_alabarda$ | 3) abar_a_la_alabarda$ |
| 6) a_la_alabarda$ | 8) alabarda$ |
| 7) la_alabarda$ | 4) ar_a_la_alabarda$ |
| 8) alabarda$ | 7) la_alabarda$ |
| 9) a$ | 2) labar_a_la_alabarda$ |

Notice we do not consider the whole string to be a suffix starting at a phrase boundary. We are interested only in phrase boundaries such that occurrences of patterns can cross the boundary with a non-empty prefix to the left of the boundary. The Patricia trees for the reversed phrases and suffixes are shown in Figure 3. Notice that, since "la_alabarda$" and "labar_a_la_alabarda$" share both their first and second characters, the second character 'a' is omitted from the edge from the root to its rightmost child in the Patrica trie for the suffixes; we indicate this with a *.

For each position $i \geq 1$ in $P$, we find the lexicographic range of the reverses of phrases ending with $P[1..i]$, and the lexicographic range of suffixes of $S$ starting

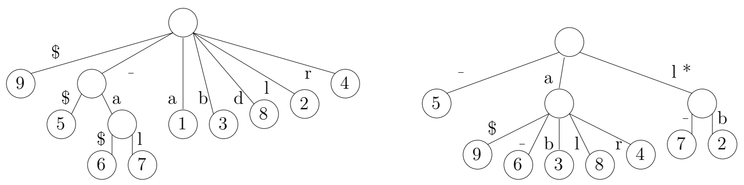
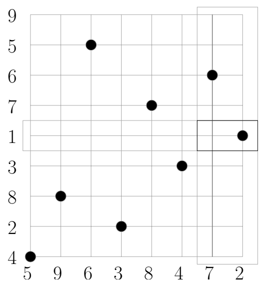


**Fig. 3.** The Patricia trees for the reversed phrases (on the left) in the LZ77 parse of "alabar_a_la_alabarda$" and the suffixes starting at phrase boundaries (on the right)

with $P[i+1..m]$ at phrase boundaries. To do this, we first search for the reverse of $P[1..i]$ in the Patricia tree for the reversed phrases and search for $P[i+1..m]$ in the Patricia tree for the suffixes, then use our data structure from Corollary 3 to verify that the characters omitted from the edge labels match those in the corresponding positions in $P$. This takes $\mathcal{O}(m + \log\log n)$ time for each choice of $i$, or $\mathcal{O}\big(m^2 + m\log\log n\big)$ time in total. For example, if $P =$ "ala", then we search for "a" and "la", "al" and "a", and "ala" and the empty suffix. In the first search, we reach the root's third child in the Patricia tree for the reversed phrases, which is the 5th leaf (labelled 1), and the root's third child in the Patricia tree for the suffixes, which is the ancestor of the 7th and 8th leaves (labelled 7 and 2); the second and third searches fail because there are no paths in the first Patricia tree corresponding to "al" or "ala".

Once we have the lexicographic ranges for a choice of $i$, we check whether there are any pairs of consecutive phrase numbers with the first number $j$ from the first range and the second number $j+1$ from the second range. In our example, $1, 2)$ is such a pair, meaning that the first phrase ends with "a" and the second phrase starts with "la", so there is a primary occurrence crossing the first phrase boundary. To be able to find efficiently such pairs, we store a data structure for 2-dimensional range reporting on a $z \times (z-1)$ grid on which we have placed $z-1$ points, with each point $(a, b)$ indicating that the phrase numbers of the lexicographically $a$th reversed phrase and the lexicographically $b$th suffix, are consecutive. Figure 4 shows the grid for our example, on the left, with the four-sided range query on the 5th row and the the 7th and 8th columns that returns the point $(5, 8)$ indicating a primary occurrence overlapping the phrases 1 and 2. Notice that the first row is always empty, since the lexicographically first reversed phrase starts with the end-of-string character $. Kreft and Navarro used a wavelet tree but we use a recent data structure by Chan, Larsen and Pătraşcu [1] that takes $\mathcal{O}(z\log\log z)$ words and answers queries in $\mathcal{O}((1+p)\log\log z)$ time, where $p$ is the number of points reported. With this data structure, we can list all $\mathrm{occ}_1$ primary occurrences in $\mathcal{O}\big(m^2 + (m + occ_1)\log\log n\big)$ time.

Once we have found all the primary occurrences, it is not difficult to recursively find all the secondary occurrences. We build a data structure for 2-sided range reporting on an $n \times n$ grid on which we place a point $(i, j)$ for each phrase's source $S[i..j]$. Since the queries are 2-sided, we can implement this data structure as a predecessor data structure and a range-maximum data structure [19] which together take $\mathcal{O}(z\log n)$ bits and answer queries in $\mathcal{O}(\log\log n + p)$ time [7,11],



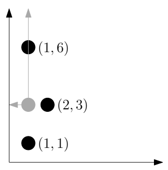


**Fig. 4.** On the left, the grid we use for listing primary occurrences in $S$ = “alabar_a_la_alabarda\$”. The sides of the grid are labelled with the phrase numbers from the leaves in the two Patricia trees. A four-sided range query on the 5th row and 7th and 8th columns that returns the point $(5, 8)$ indicating a primary occurrence overlapping the phrases 1 and 2. On the right, the grid we use for listing secondary occurrences in $S$ = “alabar_a_la_alabarda\$”. The black dots indicate phrase sources and the grey dot indicates a two-sided query to find sources containing $S[1..3]$.

where $p$ is the number of points reported. If a phrase contains a secondary occurrence at a certain position in the phrase, then that phrase's source must also contain an occurrence at the corresponding position. Notice that, by the definition of the parse, the first occurrence of any substring must be primary. It follows that, by querying the data structure with $(a, b)$ for each primary or secondary occurrence $S[a..b]$ we find, we can list all secondary occurrences of $P$ in $\mathcal{O}(\mathrm{occ} \log \log n)$ time. Figure 4 shows the grid for our example, on the right, with black dots indicating phrase sources and the grey dot indicating a two-sided query to find sources containing the primary occurrence $S[1..3]$ of “ala”. This query returns a point $(1, 6)$, so the phrase whose source is $S[1..6]$ — i.e., the 8th — contains a secondary occurrence of “ala”.

**Theorem 4.** *Given a balanced straight-line program for a string $S[1..n]$ whose LZ77 parse consists of $z$ phrases, we can add $\mathcal{O}(z \log \log z)$ words and obtain a compressed self-index for $S$ such that, given a pattern $P[1..m]$, we can list the* occ *occurrences of $P$ in $S$ in $\mathcal{O}\big(m^2 + (m + \mathrm{occ}) \log \log n\big)$ time.*

## 4 Postscript

We recently designed a compressed self-index [12] based on the Relative Lempel-Ziv (RLZ) compression scheme proposed by Kuruppu, Puglisi and Zobel [17]. Given a database of genomes from individuals of the same species, Kuruppu et al. store the first genome $G$ in an FM-index [8] and store the others compressed with a version of LZ77 [23] that allows phrases to be copied only from the first genome. They showed that RLZ compresses well and supports fast extraction but did not show how to support search. Of course, given a pattern $P[1..m]$, we can quickly find all occurrences of $P$ in $G$ by searching in its FM-index. If we store a data structure for 2-sided range reporting like the one described at the end of Section 3, then we can also quickly find all secondary occurrences of $P$ in the rest of the database. We now sketch one way to find primary occurrences.

Let $R$ be the rest of the database and suppose its RLZ parse relative to $G$ consists of $z$ phrases, $d$ of them distinct. Consider each distinct phrase as a meta-character and consider the parse as a string $R'[1..z]$ over the alphabet of meta-characters. Suppose a meta-character $x$ in $R'$ indicates a phrase copied from $G[i..j]$; then we associate with $x$ the non-empty interval corresponding to $G[i..j]$ in the Burrows-Wheeler Transform (BWT) of $G$, and the non-empty interval corresponding to the reverse $(G[i..j])^R$ of $G[i..j]$ in the BWT of $G^R$.

We will refer to two orderings on the alphabet of meta-characters: lexicographic by their phrases (*lex*), and lexicographic by the reverses of their phrases (*r-lex*). Notice that the interval for a string $A$ in the BWT of $G$ contains the intervals of all the meta-characters whose phrases start with $A$, which are consecutive in the *lex* ordering; the interval for $A^R$ in the BWT of $G^R$ contains the intervals of all the meta-characters whose phrases end with $A$, which are consecutive in the *r-lex* ordering. More generally, if the intervals for two strings overlap in a BWT, then one must be contained in the other. We can use this fact to store small data structures such that, given an interval in the BWT of $G$ or $G^R$, we can quickly find the corresponding interval in the *lex* or *r-lex* orderings of the meta-characters.

We build a data structure for 4-sided range reporting on a $d \times z$ grid. We place a point $(a, b)$ on the grid if the lexicographically $b$th suffix of $R'$, considering meta-characters in *lex* order, is preceded by a copy of the $a$th distinct meta-character in the *r-lex* ordering. In addition to the FM-index for $G$ we also store FM-indexes for $G^R$ and $R'$. We can find all primary occurrences of $P$ in $R$ if, for $1 \leq i \leq m$, we

1. use the FM-index for $G^R$ to find the interval for $(P[1..i])^R$ in the BWT of $G^R$;
2. map that interval to the interval in the *r-lex* ordering containing phrases ending with $P[1..i]$;
3. compute the RLZ parse of $P[i+1..m]$ relative to $G$;
4. use the FM-index for $G$ to find the interval for the last phrase $P[j..m]$ of that parse in the BWT of $G$;
5. map that interval to the interval in the *lex* ordering containing phrases starting with $P[j..m]$;
6. use the FM-index for $R'$ to find the interval for the parse of $P[i+1..m]$ in the BWT of $R'$;
7. find all points $(a, b)$ on the $d \times z$ grid with $a$ in the interval for $(P[1..i])^R$ in the BWT of $G^R$, and $b$ in the interval for the parse of $P[i+1..m]$ in the BWT of $R'$.

Notice that, if an occurrence of $P[i+1..m]$ starts at a phrase boundary in $R$'s parse, then the phrases in $P[i+1..m]$'s parse are the same as the subsequent phrases in $R$'s parse, except that the last phrase $P[j..m]$ in $P[i+1..m]$'s parse may be only a prefix of the corresponding phrase in $R$'s parse. When we use the

FM-index for $R'$ to find the interval for $P[i+1..m]$'s parse in the BWT of $R'$, we start with the interval containing the meta-characters that immediately precede in $R'$ meta-characters whose phrases start with $P[j..m]$. If we encounter another phrase in $P[i+1..m]$'s parse that does not occur in $R$'s parse, then we know no occurrence of $P[i+1..m]$ starts at a phrase boundary in $R$'s parse.

Steps 1, 3, 4 and 6 in the process above take time depending linearly on $m$, however, so repeating the whole process for each $i$ between 1 and $m$ takes time depending quadratically on $m$. If we use the FM-index for $G^R$ to search for $(P[1..m])^R$ in $G^R$ then, as a byproduct, we find the intervals for $(P[1..i])^R$ in the BWT of $G^R$ for $1 \leq i \leq m$. Similarly, if we use the FM-index for $G$ to search for $P[1..m]$ in $G$ then we find the intervals for $P[j..m]$ in the BWT of $G$ for $1 \leq j \leq m$. We can therefore avoid repeating Steps 1 and 4.

To avoid repeating Steps 3 and 6, we use 1-dimensional dynamic programming to compute the RLZ parses of all suffixes of $P$ and the corresponding intervals in the BWT of $R'$. Assume that, given the interval for a string $Ac$ in the BWT of $G$ and the length of $A$, we can quickly find the interval for $A$; we will explain later how we do this. We work from right to left in $P$. Assume we have already computed the interval in the BWT of $G$ for the longest prefix $P[i+1..\ell]$ of $P[i+1..m]$ that occurs in $G$ and that, for $i+1 \leq k \leq m$, we have already computed the RLZ parse of $P[k..m]$ relative to $G$ and the interval for that parse in the BWT of $R'$. We will show how to compute the interval in the BWT of $G$ for the longest prefix of $P[i..m]$ that occurs in $G$, the RLZ parse of $P[i..m]$ and the interval for that parse in the BWT of $R'$.

Knowing the interval for $P[i+1..\ell]$ in the BWT of $G$, we can use the FM-index for $G$ to find the interval for $P[i..\ell]$. If this interval is non-empty, then $P[i..\ell]$ is the longest prefix of $P[i..m]$ that occurs in $G$, and the first phrase in the RLZ parse of $P[i..m]$; the rest of the parse is the same as for $P[\ell+1..m]$. We map the interval for $P[i..\ell]$ in the BWT of $G$ to the corresponding interval in the *lex* order. If $\ell = m$, then the interval in the BWT of $R'$ is the one containing the meta-characters that immediately precede in $R'$ meta-characters whose phrases start with $P[i..m]$. If $\ell < m$, then the interval in the *lex* order should be associated with a single meta-character; otherwise, $P[i..\ell]$ does not occur in the parse of $R$ and, thus, the interval in the BWT of $R'$ is empty. Knowing the interval for the parse of $P[\ell+1..m]$ in the BWT of $R'$, we use the FM-index for $R'$ to find the interval for the parse of $P[i..m]$.

If the interval for $P[i..\ell]$ in the BWT of $G$ is empty, then we compute the intervals for $P[i+1..\ell']$ and $P[i..\ell']$, for $\ell'$ decreasing from $\ell-1$ to $i$, until we find a value of $\ell'$ such that the interval for $P[i..\ell']$ is non-empty; we then proceed as described above. We now explain how, given the interval for a string $Ac$ in the BWT of $G$ and the length of $A$, we can quickly find the interval for $A$. In addition to the FM-index for $G$, we store a compressed suffix array (CSA) and compressed longest-common-prefix (LCP) array [9] for $G$ and a previous/next-smaller-value data structure [10] for the LCP. Suppose $[i..j]$ is the interval for $Ac$ in the BWT of $G$. Notice that $\mathrm{LCP}[i]$ and $\mathrm{LCP}[j+1]$ are both at most $|A|$. If $\mathrm{LCP}[i] = |A|$, then we use the P/NSV data structure to find the previous entry LCP$[i']$ in the LCP array that is smaller than $|A|$; otherwise, we set $i' = i$. If LCP$[j] = |A|$, then we use the P/NSV data structure to find the next entry LCP$[j' + 1]$ in the LCP array that is smaller than $|A|$; otherwise, we set $j' = j$. The interval for $A$ in the BWT of $G$ is $[i'..j']$.

We use the CSA by Grossi, Gupta and Vitter [13] which takes $(1/\epsilon) n H_k(G) + \mathcal{O}(n)$ bits and supports access to the suffix array of $G$ in $\mathcal{O}(\log^{\epsilon} n)$ time, where $k \leq \alpha \log_{\sigma} n$, $\sigma$ is the size of the alphabet of $G$ and $\alpha < 1$ and $\epsilon \leq 1$ are constants; see [21]. We note that, using this CSA, we can also reduce the time needed to locate the occurrences of $P$ in $G$. Choosing the right implementations of the various other data structures, we obtain the following result:

**Theorem 5.** *Suppose we are asked to build a compressed self-index for a database of genomes from individuals of the same species. Let $G[1..n]$ be the first genome in the database and suppose the RLZ parse of the rest of the database relative to G consists of $z$ phrases. For any positive constants $\alpha < 1$ and $\epsilon$ and $k \leq \alpha \log_4 n$, we can store the database in $\mathcal{O}(n(H_k(G) + 1) + z(\log n + \log z \log\log z))$ bits such that, given a pattern $P[1..m]$, we can find all* occ *occurrences of $P$ in the database in $\mathcal{O}((m + \mathrm{occ}) \log^{\epsilon} n)$ time.*

Many of the ideas used in this section have been used previously by other authors, notably Chien et al. [3], Hon et al. [14] and Huang et al. [15]. We recently became aware that Do, Jansson, Sadakane and Sung [6] independently proved a theorem similar to Theorem 5 before we did. We have described our own ideas here because they lead to an incomparable time-space tradeoff.

## References

1. Chan, T.M., Larsen, K.G., Pătraşcu, M.: Orthogonal range searching on the RAM, revisited. In: Proceedings of the 27th Symposium on Computational Geometry (SoCG), pp. 1–10 (2011)
2. Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. IEEE Transactions on Information Theory 51(7), 2554–2576 (2005)
3. Chien, Y.-F., Hon, W.-K., Shah, R., Vitter, J.S.: Geometric Burrows-Wheeler Transform: Linking range searching and text indexing. In: Proceedings of the Data Compression Conference (DCC), pp. 252–261 (2008)
4. Claude, F., Navarro, G.: Self-indexed Text Compression Using Straight-Line Programs. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 235–246. Springer, Heidelberg (2009)
5. Claude, F., Navarro, G.: Improved grammar-based self-indexes. Tech. Rep. 1110.4493, arxiv.org (2011)
6. Do, H.H., Jansson, J., Sadakane, K., Sung, W.K.: Indexing strings via textual substitutions from a reference, manuscript
7. van Emde Boas, P.: Preserving order in a forest in less than logarithmic time. In: Proceedings of the 16th Symposium on Foundations of Computer Science (FOCS), pp. 75–84 (1975)
8. Ferragina, P., Manzini, G.: Indexing compressed text. Journal of the ACM 52(4), 552–581 (2005)

9. Fischer, J.: Wee LCP. Information Processing Letters 110(8-9), 317–320 (2010)
10. Fischer, J.: Combined data structure for previous- and next-smaller-values. Theoretical Computer Science 412(22), 2451–2456 (2011)
11. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: Proceedings of the 16th Symposium on Theory of Computing (STOC), pp. 135–143 (1984)
12. Gagie, T., Kärkkäinen, J., Nekrich, Y., Puglisi, S.J.: A compressed self-index for genomic databases. Tech. Rep. 1110.1355, arxiv.org (2011)
13. Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: Proceedings of the 14th Symposium on Discrete Algorithms (SODA), pp. 841–850 (2003)
14. Hon, W.-K., Shah, R., Thankachan, S.V., Vitter, J.S.: On Entropy-Compressed Text Indexing in External Memory. In: Karlgren, J., Tarhio, J., Hyyrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 75–89. Springer, Heidelberg (2009)
15. Huang, S., Lam, T.W., Sung, W.K., Tam, S.L., Yiu, S.M.: Indexing Similar DNA Sequences. In: Chen, B. (ed.) AAIM 2010. LNCS, vol. 6124, pp. 180–190. Springer, Heidelberg (2010)
16. Kreft, S., Navarro, G.: Self-indexing Based on LZ77. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 41–54. Springer, Heidelberg (2011)
17. Kuruppu, S., Puglisi, S.J., Zobel, J.: Relative Lempel-Ziv Compression of Genomes for Large-Scale Storage and Retrieval. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 201–206. Springer, Heidelberg (2010)
18. Maruyama, S., Nakahara, M., Kishiue, N., Sakamoto, H.: ESP-Index: A Compressed Index Based on Edit-Sensitive Parsing. In: Grossi, R., Sebastiani, F., Silvestri, F. (eds.) SPIRE 2011. LNCS, vol. 7024, pp. 398–409. Springer, Heidelberg (2011)
19. McCreight, E.M.: Priority search trees. SIAM Journal on Computing 14(2), 257–276 (1985)
20. Morrison, D.R.: PATRICIA - Practical algorithm to retrieve information coded in alphanumeric. Journal of the ACM 15(4), 514–534 (1968)
21. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Computing Surveys 39(1) (2007)
22. Rytter, W.: Application of Lempel-Ziv factorization to the approximation of grammar-based compression. Theoretical Computer Science 302(1-3), 211–222 (2003)
23. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23(3), 337–343 (1977)

# Learnability of Co-r.e. Classes

Ziyuan Gao[1] and Frank Stephan[2,⋆]

[1] Department of Mathematics,
National University of Singapore, Singapore 117543, Republic of Singapore
ziyuan84@yahoo.com

[2] Department of Mathematics and Department of Computer Science,
National University of Singapore, Singapore 117543, Republic of Singapore
fstephan@comp.nus.edu.sg

**Abstract.** The object of investigation in this paper is the learnability of co-recursively enumerable (co-r.e.) languages based on Gold's [11] original model of inductive inference. In particular, the following learning models are studied: finite learning, explanatory learning, vacillatory learning and behaviourally correct learning. The relative effects of imposing further learning constraints, such as conservativeness and prudence on these various learning models are also investigated. Moreover, an extension of Angluin's [1] characterisation of identifiable indexed families of recursive languages to families of conservatively learnable co-r.e. classes is presented. In this connection, the paper considers the learnability of indexed families of recursive languages, uniformly co-r.e. classes as well as other general classes of co-r.e. languages. A containment hierarchy of co-r.e. learning models is thereby established; while this hierarchy is quite similar to its r.e. analogue, there are some surprising collapses when using a co-r.e. hypothesis space; for example vacillatory learning collapses to explanatory learning.

## 1 Introduction

The model of learning adopted in the present paper is built on that pioneered by Gold [11]. In this setting, a language is conceived as a set of strings over some fixed finite alphabet; a learner is modelled as an algorithmic machine that reads step by step example strings from a pre-assigned language and at each step, based on what it has read so far, outputs a conjecture on the language using a specified nomenclature. The selection of a suitable nomenclature, known formally as a hypothesis space, depends to some extent on the nature of the languages considered: if, for example, a language $L$ is recursive, then a programme for a decision procedure on whether or not a given string belongs to $L$ may be an adequate hypothesis for describing $L$. The learner is said to have successfully learnt a target language if it outputs a correct identification of the language at some point and thenceforth never returns to an incorrect conjecture. The success of a learner in identifying a language $L$ may be contingent on the sort of string

⋆ Work is supported in part by NUS grant R252-000-420-112.

examples presented to it at each step, and also on the order of presentation; the ensuing discussion is confined to the case that only positive data is presented in the form of a text - an infinite sequence of strings that contains all strings of $L$.

Gold's original model has since been generalised in various directions to encompass a large class of learnability notions and systems of learning. One may consider in particular the learnability of co-recursively enumerable classes in the limit. Accordingly, a language is modelled as a co-recursively enumerable class of strings, and a learner is represented by a recursive function mapping the set of finite sequences of natural numbers together with a pause symbol (#) into the set of natural numbers. As a motivation for studying learnability under this new scheme, it may be noted that several psychology experiments [9] have revealed that subjects tasked to identify certain conjunctive concepts generally tend to perform better when receiving positive instances rather than negative instances, even when both types of instances carry the same amount of information. At the same time, it has been observed [14] that the performance of subjects in *disjunctive* concept identification tasks is initially superior when they receive negative instances to that when they receive positive instances.

If the human mind is modelled as a recursive learning machine, and the extension of a concept is always assumed to be r.e., it is possible to interpret the learnability of languages with respect to a co-r.e. hypothesis space as a model for concept identification based on negative examples. On this view, it is hoped that the results in this paper may offer some insight on whether the human processing of negative instances differs from positive instances with regard to learnability of concepts, and whether there are precise settings under which the two kinds of data presentations yield identical learnability results. As was pointed out earlier, the choice of a hypothesis space may be a decisive factor in determining the learnability of a class. We shall take a fixed universal numbering of all co-recursively enumerable sets, whose existence is well-established [20], as a natural hypothesis space for the type of classes considered. This work sits between the setting of uniformly recursive languages considered by Angluin, Lange and Zeugmann [1,21,15,17] and the inference using limiting recursive programs by Case, Jain and Sharma [7] or correction grammars by Carlucci, Case and Jain [5].

The present paper proceeds along two main lines of investigation: the classification of learning criteria according to their relative strengths, as well as the intrinsic characterisation of classes that are learnable under particular constraints. In regard to the latter objective, the paper focusses on characterising classes that are conservatively learnable with respect to a co-r.e. hypothesis space. The main contribution, Theorem 12, formulates this characterisation based on the notion of a *tell-tale set*, originally introduced by Angluin [1]: a finite set $D$ is a tell-tale set for some language $L$ if $D$ is a subset of $L$, and there are no languages $L'$ that are properly contained in $L$ and which also contain $D$. The purpose of a tell-tale set is to provide sufficient evidence for a plausible conjecture based on a finite number of instances of the target language. As a model for learning scientific theories, the restriction to a co-r.e. hypothesis space of the learner is a natural realisation of Popper's [19] proposed criterion that a statement of an empirically scientific nature should be falsifiable. In fact, conservative learners using co-enumerable hypotheses spaces explicitly follow Popper's principle of scientific discovery: they revise hypotheses when they are proven to be false and they issue hypotheses such that they can be refuted whenever they are wrong, at least as long as the learner has only to deal with target sets from the class to be learnt.

More generally, learnability with respect to a co-r.e. hypothesis space appears to be quite a stringent criterion for a scientific law to be learnt recursively, as a learner is required to formulate in the limit a general rule enumerating counterexamples based on the positive data it has seen so far. The subsequent results obtained provide evidence that under fairly natural conditions, learnability using co-enumerable indices is indeed a sharper criterion than learnability using enumerable indices.

**Notation 1.** The notation and terminology from recursion theory adopted in this paper follow [20] in the main. The abbreviation r.e. shall be used for the term "recursively enumerable." Note that it suffices to consider r.e. and co-r.e. languages of natural numbers, since Cantor's pairing function permits a coding of all strings over some fixed finite or countable alphabet into the natural numbers; hence the r.e. languages range over subsets of the set $\mathbb{N} = \{0, 1, 2, \ldots\}$. A universal numbering of all partial-recursive functions is fixed as $\varphi_0, \varphi_1, \varphi_2, \ldots$. Given a set $S$, $\overline{S}$ denotes the complement of $S$, and $S^*$ denotes the set of all finite sequences of elements from $S$. $\overline{W}_0, \overline{W}_1, \overline{W}_2, \ldots$ is a universal numbering of all co-r.e. sets, where $W_e$ is the domain of $\varphi_e$. $\langle x, y\rangle$ denotes Cantor's pairing function, given by $\langle x, y\rangle = \frac{1}{2}(x+y)(x+y+1)+y$. $W_{e,s}$ is an approximation to $W_e$; without loss of generality, $W_{e,s} \subseteq \{0, 1, \ldots, s\}$, and $\{\langle e, x, s\rangle : x \in W_{e,s}\}$ is primitive recursive; $W_e = \bigcup_s W_{e,s}$ and $W_{e,s} \subseteq W_{e,s+1}$ for all $s$. $\varphi_e(x)\uparrow$ means that $\varphi_e(x)$ remains undefined; $\varphi_{e,s}(x)\downarrow$ means that $\varphi_e(x)$ is defined, and that the computation of $\varphi_e(x)$ halts within $s$ steps. $\mathbb{K}$ denotes the diagonal halting problem, $\{e : \varphi_e(e)\downarrow\}$. For any two sets $A$ and $B$, $A \oplus B = \{2x : x \in A\} \cup \{2y+1 : y \in B\}$. Analogously, $A \oplus B \oplus C = \{3x : x \in A\} \cup \{3y+1 : y \in B\} \cup \{3z+2 : z \in C\}$.

For any $\sigma, \tau \in (\mathbb{N} \cup \{\#\})^*, \sigma \preceq \tau$ if and only if $\sigma = \tau$ or $\tau$ is an extension of $\sigma$, $\sigma \prec \tau$ if and only if $\sigma$ is a proper prefix of $\tau$, and $\sigma(n)$ denotes the element in the $n$th position of $\sigma$, starting from $n = 0$. $Pref(\sigma)$ denotes the sequence of all prefixes, in lexicographic ordering, of $\sigma$. Given a number $a$ and some fixed $n \geq 1$, denote by $a^n$ the finite sequence $a \ldots a$, where $a$ occurs $n$ times. $a^0$ denotes the empty string. The concatenation of two strings $\sigma$ and $\tau$ shall be denoted by $\sigma\tau$, and occasionally by $\sigma \circ \tau$.

**Definition 2.** Let $\{L_i\}_{i\in\mathbb{N}}$ be a family of languages.

i. $\{L_i\}_{i\in\mathbb{N}}$ is *uniformly recursive* if the set $\{\langle i, x\rangle : x \in L_i\}$ is recursive.
ii. $\{L_i\}_{i\in\mathbb{N}}$ is *uniformly recursively enumerable* if the set $\{\langle i, x\rangle : x \in L_i\}$ is recursively enumerable.
iii. $\{L_i\}_{i\in\mathbb{N}}$ is *uniformly co-recursively enumerable* if the set $\{\langle i, x\rangle : x \notin L_i\}$ is recursively enumerable.

**Definition 3.** A class $\mathcal{L}$ of recursive languages is said to be an *indexed family of recursive languages* if there exists a uniformly recursive family $\{L_i : i \in \mathbb{N}\}$ such that $\mathcal{L} = \{L_i : i \in \mathbb{N}\}$. $\mathcal{L}$ is said to be an *indexed family of r.e. (co-r.e.) languages* if there exists a uniformly r.e. (co-r.e.) family $\{L_i : i \in \mathbb{N}\}$ such that $\mathcal{L} = \{L_i : i \in \mathbb{N}\}$.

## 2 Learnability

Let $\mathcal{C}$ be a class of recursive, recursively enumerable or co-recursively enumerable sets. A *text* $T_L$ for some $L$ in $\mathcal{C}$ is a map $T_L : \mathbb{N} \to L \cup \{\#\}$, whose range contains every element of $L$. $T[n]$ denotes the string $T(0)T(1)\ldots T(n)$. A learner is a recursive function $M : (\mathbb{N} \cup \{\#\})^* \to \mathbb{N}$. The two main learning criteria studied in this paper are *explanatory* and *behaviourally correct* learning. In general, the subscript of the symbol for a learning criterion indicates the nature of the hypothesis space used; "r.e." and "co-r.e." would refer to r.e. and co-r.e. hypothesis spaces respectively, while "rec" would refer to a recursive hypothesis space.

i. [11] $M$ is said to *explanatorily* ($Ex_{co-r.e.}$) learn $\mathcal{C}$ if, for each $L$ in $\mathcal{C}$, and any corresponding text $T_L$ for $L$, there is a number $n$ for which $L = \overline{W}_{M(T_L[j])}$ whenever $j \geq n$, and for any $k \geq j$, $M(T_L[k]) = M(T_L[j])$.
ii. [8] $M$ is said to *behaviourally correctly* ($BC_{co-r.e.}$) learn $\mathcal{C}$ if, for each $L$ in $\mathcal{C}$, and any corresponding text $T_L$ for $L$, there is a number $n$ for which $L = \overline{W}_{M(T_L[j])}$ whenever $j \geq n$.

The next three definitions restrict the way the learner handles hypotheses.

**Definition 4.** i. [18] A recursive learner $M$ is said to be *prudent* (*Prud*) if it learns the class $\{\overline{W}_{M(\sigma)} : \sigma \in (\mathbb{N} \cup \{\#\})^*\}$. In other words, a prudent learner $M$ learns every set it conjectures.
ii. [1] A recursive learner $M$ is said to *conservatively* ($Cnsv_{co-r.e.}$) learn $\mathcal{C}$ if it $Ex_{co-r.e.}$ learns $\mathcal{C}$ and if, given any two finite sequences $\sigma, \tau \in (\mathbb{N} \cup \{\#\})^*$ such that $M(\sigma) \neq M(\sigma\tau)$, there is a number $x$ with $x \in \text{range}(\sigma\tau) - \overline{W}_{M(\sigma)}$.
iii. [15] Let $\{L_i\}_{i \in \mathbb{N}}$ be a family of languages. $\{L_i\}_{i \in \mathbb{N}}$ is said to be *class-preservingly learnable* if there is a recursive learner $M$ that learns $\{L_i\}_{i \in \mathbb{N}}$ such that $\{L_i\}_{i \in \mathbb{N}} = \{\overline{W}_{M(\sigma)} : \sigma \in (\mathbb{N} \cup \{\#\})^*\}$.

**Remark 5.** Throughout this paper, we often consider combinations of various learning contraints with different criteria; whenever there is no risk of ambiguity or contradiction, the relevant notations for the criteria are concatenated. For example, the criterion of prudent and conservative learning with respect to the hypothesis space $\{\overline{W}_0, \overline{W}_1, \overline{W}_2, \ldots\}$ is denoted by $PrudCnsv_{co-r.e.}$.

## 3 Characterisations of Conservative Learning

Angluin's [1] theorem furnishes a necessary and sufficient condition for an indexed family of recursive languages to be explanatorily learnable in the limit.

This is formally stated in terms of the notion of a finite *tell-tale set*, the main role of which is to aid a learner in steering clear of overgeneralisation; that is, conjecturing a set that properly contains the target language. Overgeneralising at some stage during a sequence of guesses is a potential pitfall for the learner, as it will not observe any counter-examples to its current conjecture. However, by ensuring that the data revealed eventually contain every member of the tell-tale set corresponding to its last conjecture, the learner can avoid being caught in such a situation.

**Definition 6 (Angluin [1]).** Let $\{L_i\}_{i\in\mathbb{N}}$ be a family of languages. A uniformly r.e. family of finite sets $\{H_i\}_{i\in\mathbb{N}}$ is said to be a *family of tell-tale sets* for $\{L_i\}_{i\in\mathbb{N}}$ if, for every $i, j \in \mathbb{N}$,

- $H_i \subseteq L_i$;
- if $H_i \subseteq L_j \subseteq L_i$, then $L_j = L_i$.

For each language $L_i$, $H_i$ is said to be a *tell-tale set* for $L_i$. A class of languages for which a uniformly r.e. family of tell-tale sets exists is said to satisfy *Angluin's tell-tale condition.*

The necessity of Angluin's tell-tale condition for the learnability of indexed families follows from a more general property that learnable languages possess - the existence of locking sequences. This is stated below for the explanatory learning of co-r.e. sets, but the definition extends in a natural way to other kinds of learning criteria, as well as to r.e. sets.

**Definition 7 (Blum and Blum [4]).** Let $M$ be a recursive learner and $L$ a co-r.e. set learned by $M$. A finite sequence $\sigma$ in $(L\cup\{\#\})^*$ is said to be a *locking sequence* for $L$ if $\overline{W}_{M(\sigma)} = L$ and for all $\tau \in (L \cup \{\#\})^*, M(\sigma\tau) = M(\sigma)$.

Using the concept of a locking sequence, one can prove Angluin's theorem for indexed families of recursive languages:

**Theorem 8 (Angluin [1]).** *An indexed family of recursive languages $\mathcal{L} = \{L_i\}_{i\in\mathbb{N}}$ is explanatorily learnable if and only it it satisfies Angluin's tell-tale condition.*

In seeking to bridge the disconnect between two research areas, one in indexed families of recursive languages, and the other in general classes of r.e. languages, de Jongh and Kanazawa [13] later generalised Angluin's tell-tale condition to characterise indexed families of r.e. languages. In [1], Angluin also introduced the notion of conservativeness and observed that it is an important learnability property of indexed families. Here a learner is called *conservative* if and only if $M(\sigma\tau) \neq M(\sigma)$ only for such $\sigma, \tau$ where the range of $\sigma\tau$ contains some elements outside the hypothesis issued by $M(\sigma)$. Note that with this definition, every conservative learner is automatically explanatory. Zeugmann, Lange and Kapur [16] showed how to generalise Angluin's tell-tale condition to characterise conservative learnability of indexed families of recursive languages; this was later extended by de Jongh and Kanazawa [13] to the case of indexed families of r.e. languages. A family of finite sets $\{H_i\}_{i\in\mathbb{N}}$ is *uniformly recursively generable* if there is a total effective procedure $g$, which, on every input $j$, generates all elements of $H_j$ and stops. Conservative learnability of an indexed family of recursive languages $\mathcal{L} = \{L_i\}_{i\in\mathbb{N}}$ can be realised if there exists a family of tell-tale sets $\{H_i\}_{i\in\mathbb{N}}$ for $\mathcal{L}$ which is uniformly recursively generable. More precisely, one has the following theorem.

**Theorem 9 (Lange, Zeugmann and Kapur [16]).** *Let $\mathcal{L}$ be an indexed family of non-empty recursive languages. Then $\mathcal{L}$ is conservatively learnable if and only if there is a hypothesis space $\mathcal{G} = \{G_i\}_{i\in\mathbb{N}}$ and a uniformly recursively generable family $\{H_i\}_{i\in\mathbb{N}}$ of finite non-empty sets such that*

– *range*$(\mathcal{L}) \subseteq$ *range*$(\mathcal{G})$;
– *for all $i \in \mathbb{N}$, $H_i \subseteq G_i$;*
– *for all $i, j \in \mathbb{N}$ with $H_i \subseteq G_j$ it holds that $G_j \not\subset G_i$.*

Theorem 9 has a natural analogue when the learner is restricted to a co-r.e. hypothesis space, as will be demonstrated later. The next series of results give characterisations of conservative co-r.e. learning under various learning constraints. The first theorem states that for $Ex_{co-r.e.}$-learning, Angluin's tell-tale characterisation has a counterpart with respect to class-preserving hypothesis spaces.

**Theorem 10.** *Assume that a class $\mathcal{C}$ is uniformly co-r.e. and contains the empty set. Then $\mathcal{C}$ is conservatively $Ex_{co-r.e.}$ learnable using a class-preserving hypothesis space if and only if it has a uniformly co-r.e. indexing $L_0, L_1, \ldots$ and there is a uniformly r.e. family $H_0, H_1, \ldots$ of finite sets such that for all $i, j$ with $H_i \cap L_i \subseteq L_j \subseteq L_i$ it holds that $L_i = L_j$.*

One has indeed to take a new numbering for Angluin's tell-tale condition to be implemented. The following example is class-preservingly conservatively learnable but does not satisfy the tell-tale condition.

**Example 11.** Consider the following indexing:

$$\begin{aligned} L_0 &= \emptyset, \\ L_{2n-1} &= \{0, 1, 2, \ldots, n\} \oplus \{n\} \oplus \emptyset, \\ L_{2n} &= \{0, 1, 2, \ldots, n\} \oplus \{n\} \oplus \{x : x > |W_n|\}. \end{aligned}$$

This family is conservatively learnable, but no finite family of $H_0, H_1, \ldots$ as in Theorem 10 exists; hence, the hypothesis space has to be adequately chosen such as to achieve the existence of the tell-tale family.

It may be reasonable to inquire next whether all $Cnsv_{co-r.e.}$ learnable, but not necessarily class-preservingly learnable classes have a similar tell-tale characterisation. This is the main content of the following result, which pieces together some presently known equivalent characterisations of $Cnsv_{co-r.e.}$ learnable classes. It shows, moreover, that a conservative co-r.e. learner must also necessarily be prudent. $\{L_0, L_1, L_2, \ldots\}$ *covers* $\mathcal{C}$ if and only if $\mathcal{C} \subseteq \{L_0, L_1, L_2, \ldots\}$.

**Theorem 12.** *Let $\mathcal{C}$ be a class of co-r.e. sets. The following statements are equivalent:*
*(i) $\mathcal{C}$ is $Cnsv_{co-r.e.}$ learnable;*
*(ii) $\mathcal{C}$ is $PrudCnsv_{co-r.e.}$ learnable;*
*(iii) There is a uniformly co-r.e. family $L_0, L_1, L_2, \ldots$ covering $\mathcal{C}$ and a uniformly recursively generable family $H_0, H_1, H_2, \ldots$ of the finite sets such that for all $n, m$ it holds that $H_n \subseteq L_n$ and $H_n \subseteq L_m \subseteq L_n \Rightarrow L_n = L_m$;*
*(iv) There is a uniformly co-r.e. family $L_0, L_1, L_2, \ldots$ covering $\mathcal{C}$ and a uniformly r.e. family of finite sets $H_0, H_1, H_2, \ldots$ such that for all $n, m$ it holds that $H_n \subseteq L_n$ and $H_n \subseteq L_m \subseteq L_n \Rightarrow L_n = L_m$.*

**Sketch.** The proof of the implication (i)⇒(iii) shall be demonstrated. Suppose that $M$ is a $Cnsv_{co-r.e.}$ learner of $\mathcal{C}$. Let $\tau_0, \tau_1, \tau_2, \ldots$ be an enumeration of all sequences in $(\mathbb{N} \cup \{\#\})^*$ which satisfy the following conditions:

- A number $x$ occurs in $\tau_n$ if and only if it occurs in $\tau_n$ exactly in position $x$;
- For all $x < |\tau_n|$ such that $x \notin \text{range}(\tau_n)$, $x \in W_{M(\tau_n)}$;
- For all $\sigma \prec \tau_n$, $M(\sigma) \neq M(\tau_n)$.

Now define a uniformly co-r.e. family of sets $\{L_n\}_{n \in \mathbb{N}}$ by

$$L_n = \begin{cases} \overline{W}_{M(\tau_n)} & \text{if } W_{M(\tau_n)} \cap \text{range}(\tau_n) = \emptyset; \\ \text{range}(\tau_n) - \{\#\} & \text{if } W_{M(\tau_n)} \cap \text{range}(\tau_n) \neq \emptyset. \end{cases}$$

Furthermore, put $H_n = \text{range}(\tau_n) - \{\#\}$ for all $n$; by this definition, $\{H_n\}_{n \in \mathbb{N}}$ is a uniformly recursive family of finite sets. For each $L \in \mathcal{C}$, let $T_L$ be the text for $L$ such that for all $n$, $T_L(n) = n$ if $n \in L$, and $T_L(n) = \#$ otherwise. Since $M$ learns $L$ when fed with the text $T_L$, there is a least $n$ for which $\overline{W}_{M(T_L[n])} = L$ and $W_{M(T_L[n])} \cap \text{range}(T_L[n]) = \emptyset$; moreover, by the construction of $T_L$, $T_L[n] \in \{\tau_0, \tau_1, \tau_2, \ldots\}$. Hence $\mathcal{C} \subseteq \{L_n\}_{n \in \mathbb{N}}$. If $\text{range}(\tau_n) \cap W_{M(\tau_n)} = \emptyset$, then $L_n = \overline{W}_{M(\tau_n)}$, and so $H_n = \text{range}(\tau_n) - \{\#\} \subseteq L_n$. If $\text{range}(\tau_n) \cap W_{M(\tau_n)} \neq \emptyset$, then $L_n = \text{range}(\tau_n) - \{\#\}$, and therefore $H_n \subseteq L_n$ still holds.

It remains to prove the tell-tale characterisation of $\mathcal{C}$. Suppose that for some $i, j, H_i = \text{range}(\tau_i) - \{\#\} \subseteq L_j \subseteq L_i$. It shall be shown that $\tau_i = \tau_j$, and hence $L_i = L_j$, by eliminating in turn the following cases (a), (b) and (c). (a) Suppose that $\tau_i \prec \tau_j$. By the third condition in the construction of $\{\tau_n\}_{n \in \mathbb{N}}$, $M(\tau_i) \neq M(\tau_j)$, and so by the conservativeness of $M$, there is an $x \in \text{range}(\tau_j) \cap W_{M(\tau_i)}$, and as $\text{range}(\tau_j) - \{\#\} \subseteq L_j$, this implies that $L_j \not\subseteq L_i$, a contradiction. (b) Suppose that $\tau_j \prec \tau_i$. Again, $M(\tau_i) \neq M(\tau_j)$, and so by the conservativeness of $M$, there is an $x \in \text{range}(\tau_i) \cap W_{M(\tau_j)}$. Hence $\text{range}(\tau_i) - \{\#\} \not\subseteq L_j$, a contradiction. (c) Suppose that $\tau_i(x) \neq \tau_j(x)$ for some $x < \min\{|\tau_i|, |\tau_j|\}$. Let $e$ be the least such $x$. If $\tau_i(e) = \#$ and $\tau_j(e) = e$, then by the second condition in the construction of $\{\tau_n\}_{n \in \mathbb{N}}$, $e \in W_{M(\tau_i)}$. Further, if $L_i = \text{range}(\tau_i) - \{\#\}$, then one has $e \in L_j - L_i$, contrary to the condition that $L_j \subseteq L_i$. If $L_i = \overline{W}_{M(\tau_i)}$, then again $e \in L_j - L_i$, giving rise to the same contradiction. On the other hand, if $\tau_i(e) = e$ and $\tau_j(e) = \#$, then $e \in \text{range}(\tau_i) - L_j$, contradicting the condition that $\text{range}(\tau_i) - \{\#\} \subseteq L_j$. This completes the case distinction and establishes the required implication. □

To complete the picture, the following example shows that in general, the uniformly co-r.e. family $\{L_0, L_1, L_2, \ldots\}$ covering $\mathcal{C}$ in statements (iii) and (iv) of Theorem 12 may not be equal to $\mathcal{C}$, and so an exact Angluin tell-tale condition cannot be obtained.

**Example 13.** Consider the class $\mathcal{C} = \{D : D \text{ is finite and } D \cap \mathbb{K} = \emptyset\}$. The class $\mathcal{C}$ is $PrudCnsv_{co-r.e.}$ learnable, but there is no uniformly co-r.e. family $L_0, L_1, L_2, \ldots$ and no uniformly r.e. family of finite sets $H_0, H_1, H_2, \ldots$ with $\mathcal{C} = \{L_0, L_1, L_2, \ldots\}$ such that for all $n, m$, $H_n \subseteq L_n$, and whenever $H_n \subseteq L_m \subseteq L_n$, it holds that $L_m = L_n$.

Baliga, Case and Jain [2] proved that an indexed family is behaviourally correctly learnable using some other uniformly r.e. hypothesis space if and only if for every set $L_e$ in the hypothesis space there is a finite set $H_e \subseteq L_e$ such that if $H_e \subseteq L_d \subseteq L_e$ for some index $d$, then $L_d = L_e$. By contrast, the following example shows that some weak analogue of this result fails; Theorem 15 below implies that also the direct analogue of this result (using uniformly recursive families) fails.

**Example 14.** Let $L_e = \{\langle x, y\rangle : \varphi_e(x) = y \vee \exists z \leq x[\varphi_e(z)\uparrow]\}$ and let $\mathcal{C}$ be the class of all sets $L_e$. Then $\mathcal{C}$ is uniformly co-r.e. and satisfies Angluin's tell-tale condition non-effectively, but it is not $BC_{co-r.e.}$ learnable.

## 4 Indexed Families of Recursive Sets

The present section establishes some results on the learnability of families of recursive sets with either a uniformly r.e. numbering or a uniformly co-r.e. numbering. First, it is shown that when a given class of recursive sets has a uniformly r.e. numbering and is $BC_{co-r.e.}$ learnable, then it is already $Ex_{rec}$ learnable. Moreover, Theorem 24 in the next section will establish the existence of a uniformly r.e. class of recursive sets which is finitely learnable using r.e. indices but not $BC_{co-r.e.}$ learnable.

**Theorem 15.** *Let $\mathcal{C}$ be a uniformly r.e. class of recursive sets. If $\mathcal{C}$ is $BC_{co-r.e.}$ learnable, then it is $Ex_{rec}$ learnable.*

**Proof.** Let $A_0, A_1, \ldots$ be a uniformly recursive enumeration of the class and let $M$ be a $BC_{co-r.e.}$ learner for it. For each number $n$, let $\sigma_{n,0}, \sigma_{n,1}, \sigma_{n,2}, \ldots$ be an enumeration of all finite sequences in $A_n^*$. Then one can find for every $n$ in the limit a sequence $\sigma_n = \lim_{t\to\infty} \sigma_{n,t}$ such that $\sigma_n \in A_n^*$ and $M$ outputs on every sequence $\sigma_n\tau$ with $\tau \in A_n^*$ an index $M(\sigma_n\tau)$ of a co-r.e. set such that $W_{M(\sigma_n\tau)} \subseteq \overline{A}_n$. One can find a recursive function $f$ such that $f(n,t)$ is a recursive index for $A_n$ if $\sigma_{n,t} = \sigma_n$, and $f(n,t)$ is a recursive index for a finite set if $\sigma_{n,u} \neq \sigma_{n,t}$ for some $u > t$.

Now, on input with range $D$ and length $s$, the new learner $N$ searches for the first pair $(n,t)$ such that (1) $\sigma_{n,t} = \sigma_{n,t+s}$, (2) range$(\sigma_{n,t}) \subseteq D$, and (3) the set described by $f(n,t)$ contains $D$, and $N$ conjectures $f(n,t)$ if this pair is found in time $s$ and $f(s,s)$ otherwise. Note that condition (3) can be checked as $f(n,t)$ produces a recursive index. Furthermore, if the learner converges to $f(n,t)$, then $(n,t)$ satisfies $\sigma_{n,t} = \sigma_n$ as otherwise $(n,t)$ would not have fulfilled requirement (1) of the above list when $s$ is sufficiently large. Hence the data must contain range$(\sigma_n)$ and furthermore the data must be contained in $A_n$. □

It is an immediate consequence of Theorem 15 that an indexed family which is $BC_{co-r.e.}$ learnable must also be $Ex_{co-r.e.}$ and $Ex_{r.e.}$ learnable; this, in turn, implies by Theorem 8 that a uniformly recursive family of sets is $BC_{co-r.e.}$ learnable if and only if it satisfies Angluin's tell-tale condition. The next theorem sums up these observations, and asserts in addition that Angluin's criterion implies $Cnsv_{co-r.e.}$ learnability. In the context of concept identification, this result seems to affirm empirical findings that human learning by means of positive as opposed to negative instances is indeed superior when the extensions of concepts to be learnt are uniformly recursive.

**Theorem 16.** *Let $\mathcal{C} = \{L_e\}_{e \in \mathbb{N}}$ be a uniformly recursive family of sets. Then the following conditions are equivalent:*

*(i) $\mathcal{C}$ is $Cnsv_{co-r.e.}$ learnable;*
*(ii) $\mathcal{C}$ is $BC_{co-r.e.}$ learnable;*
*(iii) $\mathcal{C}$ is $Ex_{r.e.}$ learnable;*
*(iv) $\mathcal{C}$ satisfies Angluin's tell-tale condition.*

## 5 General Classes of r.e. Sets, Co-r.e. Sets, and Separations

This section summarizes results dealing with the two main learning criteria, namely $Ex_{co-r.e.}$ and $BC_{co-r.e.}$ learning. On the one hand, prudent, explanatory and behaviourally correct learnability form a strict hierarchy of learning notions; on the other hand, vacillatory and explanatory learnability using co-r.e. indices turn out to be equivalent. The following theorem states that *vacillatory learning*, in which the learner is permitted to vacillate between finitely many different indices, is still just as powerful as $Ex_{co-r.e.}$ learning.

**Definition 17 (Case [6]).** $M$ is said to *vacillatorily* ($Vac_{co-r.e.}$) learn $\mathcal{C}$ if it $BC_{co-r.e.}$ learns $\mathcal{C}$ and outputs on every text $T_L$ for each $L$ in $\mathcal{C}$ only finitely many different indices $i_0, i_1, \ldots, i_n$.

**Theorem 18.** *If a class is vacillatorily learnable using co-r.e. indices, then the class is explanatorily learnable using co-r.e. indices.*

However, for uniformly co-r.e. families of sets, $PrudBC_{co-r.e.}$ learnability is less restrictive than $Ex_{co-r.e.}$ learnability. This result is parallel to the usual case of learning using r.e. indices.

**Theorem 19.** *There is a uniformly co-r.e. class $\mathcal{C}$ which is $PrudBC_{co-r.e.}$ learnable but not $Ex_{co-r.e.}$ learnable.*

While Fulk [10] showed that every $Ex_{r.e.}$ learnable class is prudently $Ex_{r.e.}$ learnable and Jain, Stephan and Ye [12] showed that every $BC_{r.e.}$ learnable class is prudently $BC_{r.e.}$ learnable, the next result shows that prudence does not need to hold for co-r.e. learning, even if every member of the class is recursive.

**Theorem 20.** *There is a class $\mathcal{C}$ of recursive sets which is $Ex_{co-r.e.}$ learnable but not $PrudBC_{co-r.e.}$ learnable.*

**Sketch.** For each partial-recursive $\{0,1\}$-valued function $f$, let $L_f = \{\langle x,y\rangle : y \in \{0,1\} \wedge f(x)\downarrow \Rightarrow y = f(x)\}$. Next, define the class

$$\mathcal{C} = \{L_f : \exists e(\forall n > e)[f(e)\uparrow \wedge \varphi_e(n)\downarrow = f(n)\downarrow \in \{0,1\}]\}.$$

Then $\mathcal{C}$ is $Ex_{co-r.e.}$ learnable but not $PrudBC_{co-r.e.}$ learnable. □

A consequence of Theorem 12 is that every $Cnsv_{co-r.e.}$ learnable class is also $PrudEx_{co-r.e.}$ learnable; the next result shows that $Cnsv_{co-r.e.}$ learnability is in fact more restrictive than $PrudEx_{co-r.e.}$ learnability.

**Theorem 21.** *There is a uniformly co-r.e. class $\mathcal{C}$ which is $PrudEx_{co-r.e.}$ learnable but not $Cnsv_{co-r.e.}$ learnable.*

**Sketch.** Let $M_0, M_1, M_2, \ldots$ be a numbering of all partial-recursive learners, and $\sigma_0, \sigma_1, \sigma_2, \ldots$ be an enumeration of all elements of $\mathbb{N}^*$. Build a uniformly co-r.e. family $\{L_i\}_{i\in\mathbb{N}}$ as follows. With $\eta_0 = e$, set $\overline{L}_{2e,0} = \{\tau \in \mathbb{N}^* : |\tau| > 0 \wedge \tau(0) \neq \eta_0(0)\}$. At stage $s+1$, using $\eta_s$, search for the first $\sigma \in \mathbb{N}^*$ such that $\eta_s \prec \sigma$, and either $M_{e,s}(Pref(\eta_s))\downarrow \neq M_{e,s}(Pref(\sigma))\downarrow$, or $\sigma \in W_{M_{e,s}(\eta_s),j}$ holds for some $j < s+1$. Let the computational time of this search be bounded by $s+1$. If such a $\sigma$ is found, let $\eta_{s+1} = \sigma$ and set $\overline{L}_{2e,s+1} = \{\tau \in \mathbb{N}^* : \exists k[k < \max\{|\tau|, |\eta_{s+1}|\} \wedge \tau(k) \neq \eta_{s+1}(k)]\}$. Otherwise, if no such $\sigma$ is found, set $\eta_{s+1} = \eta_s$ and $\overline{L}_{2e,s+1} = \overline{L}_{2e,s}$. Let $\overline{L}_{2e} = \bigcup_{s\in\mathbb{N}} \overline{L}_{2e,s}$. Since, at each stage $s+1$, $\overline{L}_{2e,s} \subseteq \overline{L}_{2e,s+1}$, the set $L_{2e}$ is co-r.e., and consists of all strings which are comparable with the longest common prefix of the strings in $L_{2e,s}$ at some stage $s$. Further, set $L_{2e+1} = \{\eta \in \mathbb{N}^* : \eta \preceq \sigma_e \vee \sigma_e \preceq \eta\}$ for all $e$.

It is shown that no $M_e$ is a conservative learner of $\{L_0, L_1, L_2, \ldots\}$. Suppose that $L_{2e}$ is an infinite chain $e \prec \tau_0 \prec \tau_1 \prec \tau_2 \prec \ldots$ of comparable strings; it follows that $M_e$ fails to learn $L_{2e}$, as there are infinitely many stages at which $M_e$ will either make a mind change, or conjecture a set whose complement enumerates elements of $L_{2e}$, when fed with the text $T$ such that $T[i] = e \circ \tau_0 \circ \tau_1 \circ \tau_2 \ldots \circ \tau_{i-1}$ for all $i \geq 1$. By contrast, if $L_{2e} = \{\eta \in \mathbb{N}^* : \eta \preceq \sigma \vee \sigma \preceq \eta\}$ for some $\sigma \in \mathbb{N}^*$, then $L_{2e} \subseteq \overline{W}_{M_e(Pref(\sigma))}$. Further, if $L_{2e}$ is not a proper subset of $\overline{W}_{M_e(Pref(\sigma))}$, then the set $H = \{\eta \in \mathbb{N}^* : \eta \preceq \sigma \circ 0 \vee \sigma \circ 0 \preceq \eta\}$ is a member of $\{L_0, L_1, L_2, \ldots\}$ that must be a proper subset of $\overline{W}_{M_e(Pref(\sigma))}$, and so $M_e$ cannot conservatively learn $H$ when fed with a text that starts with $Pref(\sigma)$. □

The final results separate the learnability of classes of recursive sets with respect to whether r.e. or co-r.e. hypothesis spaces are used. These results are a bit parallel to prior results on learning correction grammars [5].

**Definition 22 (Bārzdiņš and Freivalds [3]).** $M$ is said to *finitely* ($Fin_{co-r.e.}$) learn $\mathcal{C}$ if, for each $L$ in $\mathcal{C}$ and any text $T_L$ for $L$, there are an index $i$ and a number $n$ such that $\overline{W_i} = L$, $M(T_L[k]) = i$ for all $k \geq n$ and $M(T_L[k]) = ?$ for all $k < n$.

**Theorem 23.** *There exists a uniformly co-r.e. class of recursive sets which is $Fin_{co-r.e.}$ learnable but not $BC_{r.e.}$ learnable.*

**Theorem 24.** *There is a uniformly r.e. class of recursive sets which is $Fin_{r.e.}$ learnable but not $BC_{co-r.e.}$ learnable.*

**Sketch.** Let $L_n = \{n\} \oplus \{m : W_{n,m} \subset W_n \vee (m > 0 \wedge W_{n,m} = W_{n,m-1})\}$ and $\mathcal{C} = \{L_0, L_1, \ldots\}$; this family is uniformly r.e., finitely learnable with respect to r.e. indices, but is not $BC_{co-r.e.}$ learnable. □

Thus, while $BC_{r.e.}$ and $Fin_{co-r.e.}$ are incomparable learning notions, it follows from Theorem 15 that the class of all uniformly r.e. sets of recursive languages which are $Ex_{rec}$ learnable strictly subsumes all uniformly r.e. families of $BC_{co-r.e.}$ learnable recursive languages. At present, however, it is still unknown whether or not there is a uniformly co-r.e. class of recursive languages which is $Ex_{r.e.}$ learnable but not $BC_{co-r.e.}$ learnable.

## 6 Conclusion

Co-r.e. languages are a natural counterpart to the r.e. ones and the current work studies the learnability of these languages. This framework of learning may be viewed as an approximate model for concept identification based on negative instances. Theorem 12 shows that for co-r.e. languages, Angluin's tell-tale condition characterises conservative learnability and not explanatory learnability. Furthermore, the characterisation works only with some class-comprising hypothesis space and not the given one. Later, Theorem 16 shows that for indexed families of recursive sets, the criteria $Ex_{rec}$ $Cnsv_{co-r.e.}$ and $BC_{co-r.e.}$ coincide. While for $Ex_{r.e.}$ and $BC_{r.e.}$ learning, prudence can be achieved, there is an $Ex_{co-r.e.}$ learnable class which does not even have a prudent $BC_{co-r.e.}$ learner.

## References

1. Angluin, D.: Inductive inference of formal languages from positive data. Information and Control 45(2), 117–135 (1980)
2. Baliga, G., Case, J., Jain, S.: The synthesis of language learners. Information and Computation 152, 16–43 (1999)
3. Bārzdiņš, J., Freivalds, R.: On the prediction of general recursive functions. Soviet Mathematical Doklady 13, 1224–1228 (1972)
4. Blum, L., Blum, M.: Towards a mathematical theory of inductive inference. Information and Control 28, 125–155 (1975)
5. Carlucci, L., Case, J., Jain, S.: Learning correction grammars. The Journal of Symbolic Logic 74, 489–516 (2009)

6. Case, J.: The power of vacillation in language learning. SIAM Journal on Computing 28(6), 1941–1969 (1999)
7. Case, J., Jain, S., Sharma, A.: On learning limiting programs. International Journal of Foundations of Computer Science 3, 93–115 (1992)
8. Case, J., Lynes, C.: Machine Inductive Inference and Language Identification. In: Nielsen, M., Schmidt, E.M. (eds.) ICALP 1982. LNCS, vol. 140, pp. 107–115. Springer, Heidelberg (1982)
9. Freibergs, V., Tulving, E.: The effect of practice on utilization of information from positive and negative instances in concept identification. Canadian Journal of Psychology 15(2), 101–106 (1961)
10. Fulk, M.A.: Prudence and other conditions on formal language learning. Information and Computation 85(1), 1–11 (1990)
11. Gold, E.M.: Language identification in the limit. Information and Control 10, 447–474 (1967)
12. Jain, S., Stephan, F., Ye, N.: Prescribed learning of r.e. classes. Theoretical Computer Science 410(19), 1796–1806 (2009)
13. Jongh, D.D., Kanazawa, M.: Angluin's theorem for indexed families of r.e. sets and applications. In: COLT, pp. 193–204 (1996)
14. Krebs, M.J., Lovelace, E.A.: Disjunctive concept identification: stimulus complexity and positive versus negative instances. Journal of Verbal Learning and Verbal Behaviour 9, 653–657 (1970)
15. Lange, S., Zeugmann, T.: Set-driven and rearrangement-independent learning of recursive languages. Mathematical Systems Theory 29(6), 599–634 (1996)
16. Lange, S., Zeugmann, T., Kapur, S.: Characterizations of monotonic and dual monotonic language learning. Information and Computation 120(2), 155–173 (1995)
17. Lange, S., Zeugmann, T., Zilles, S.: Learning indexed families of recursive languages from positive data: a survey. Theoretical Computer Science 397(1-3), 194–232 (2008)
18. Osherson, D., Stob, M., Weinstein, S.: Learning strategies. Information and Control 53, 32–51 (1982)
19. Popper, K.R.: Conjectures and refutations: the growth of scientific knowledge. Routledge and Kegan Paul, London (1972)
20. Rogers Jr., H.: Theory of recursive functions and effective computability. MIT Press, Cambridge (1987)
21. Zeugmann, T., Lange, S.: A Guided Tour Across the Boundaries of Learning Recursive Languages. In: Lange, S., Jantke, K.P. (eds.) GOSLER 1994. LNCS, vol. 961, pp. 190–258. Springer, Heidelberg (1995)

# Two-Way Automata Making Choices Only at the Endmarkers

Viliam Geffert[1,⋆], Bruno Guillon[2], and Giovanni Pighizzini[3]

[1] Department of Computer Science, P. J. Šafárik University, Košice, Slovakia
`viliam.geffert@upjs.sk`

[2] Université Nice-Sophia Antipolis and École Normale Supérieure de Lyon, France
`guillon.bruno+cs@gmail.com`

[3] Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, Italy
`pighizzini@dico.unimi.it`

**Abstract.** The question of the state-size cost for simulation of two-way nondeterministic automata (2NFAs) by two-way deterministic automata (2DFAs) was raised in 1978 and, despite many attempts, it is still open. Subsequently, the problem was attacked by restricting the power of 2DFAs (e.g., using a restricted input head movement) to the degree for which it was already possible to derive some exponential gaps between the weaker model and the standard 2NFAs. Here we use an opposite approach, increasing the power of 2DFAs to the degree for which it is still possible to obtain a subexponential conversion from the stronger model to the standard 2DFAs. In particular, it turns out that subexponential conversion is possible for two-way automata that make nondeterministic choices only when the input head scans one of the input tape endmarkers. However, there is no restriction on the input head movement. This implies that an exponential gap between 2NFAs and 2DFAs can be obtained only for unrestricted 2NFAs using capabilities beyond the proposed new model.

As an additional bonus, conversion into a machine for the complement of the original language is polynomial in this model. The same holds for making such machines self-verifying, halting, or unambiguous. Finally, any superpolynomial lower bound for the simulation of such machines by standard 2DFAs would imply $L \neq NL$. In the same way, the alternating version of these machines is related to $L \overset{?}{=} NL \overset{?}{=} P$, the classical computational complexity problems.

**Keywords:** Two-way Automata, Descriptional Complexity, Regular Languages.

⋆ Supported by the Slovak Grant Agency for Science (VEGA 1/0479/12) under contract "Combinatorial Structures and Complexity of Algorithms", and by the Slovak Research and Development Agency (APVV-0035-10) under contract "Algorithms, Automata, and Discrete Data Structures".

## 1 Introduction

The cost, in terms of states, of the simulation of two-way nondeterministic automata (2NFAs, for short) by two-way deterministic automata (2DFAs) is one of the most important and challenging open problems in automata theory and, in general, in theoretical computer science. This problem was proposed in 1978 by Sakoda and Sipser [19], who conjectured that the cost is exponential. However, in spite of all effort, exponential gaps were proved only between 2NFAs and some restricted *weaker versions of* 2DFA*s*.

In 1980, Sipser proved that if the resulting machine is required to be *sweeping* (deterministic and reversing the direction of its input head only at the *endmarkers*, two special symbols used to mark the left and right ends of the input), the simulation of a 2NFA is indeed exponential [22]. However, Berman and Micali [1,18] proved independently that this does not solve the general problem: in fact the simulation of unrestricted 2DFAs by sweeping 2DFAs also requires an exponential number of states. The Sipser's result was generalized by Hromkovič and Schnitger [11], who considered *oblivious machines* (following the same trajectory of input head movements along all inputs of equal length) and, recently, by Kapoutsis [15], considering 2DFAs with the number of input head reversals that is sublinear in the length of the input. However, even the last condition gives a machine provably less succinct than unrestricted 2DFAs, and hence the general problem remains open.

Starting from 2003 with a paper by Geffert *et al.* [7], a different kind of restriction has been investigated: the subclass or regular languages using *a single-letter input alphabet*. Even under this restriction, the problem of Sakoda and Sipser looks difficult, since it is connected with $\mathrm{L} \stackrel{?}{=} \mathrm{NL}$, an open question in complexity theory. (L and NL denote the respective classes of languages accepted in deterministic and nondeterministic logarithmic space.) First, in [7], a new normal form was obtained for unary automata, in which all nondeterministic choices and input head reversals take place only at the endmarkers. Moreover, the state-size cost of the conversion into this normal form is only *linear*. This normal form is a starting point for several other properties of unary 2NFAs. First, in the same paper, each $n$-state unary 2NFA is simulated by an equivalent 2DFA with $O(n^{\lceil \log_2(n+1)+3 \rceil})$ states, which gives a subexponential but still superpolynomial upper bound. It is not known whether this simulation is tight. However, a positive answer would imply the separation between the classes L and NL. In fact, under assumption that L = NL, each unary 2NFA with $n$ states can be simulated by a 2DFA with a number of states polynomial in $n$ [9]. After a minor modification (without assuming L = NL), this gives that each unary 2NFA can be made unambiguous, keeping the number of the states polynomial. (For further connections between two-way automata and logarithmic space, we address the reader to [2,16].)

Along these lines of investigation, in [8], the problem of the complementation for unary 2NFAs has been considered, by proving that each $n$-state 2NFA accepting a unary language $L$ can be replaced by a 2NFA with $O(n^8)$ states accepting the complement of $L$. The proof combines the above normal form for unary 2NFAs with inductive counting arguments.

Kapoutsis [13] considered the complementation in the case general input alphabets, but restricting the input head reversals. He showed that the complementation of *sweeping* 2NFAs (with the input head reversals only at the endmarkers) requires exponentially many states, thus emphasizing a relevant difference with the unary case.

In this paper, we use a different approach. Instead of *restricting the power of* 2DFA*s* to the degree for which it is already possible to derive an exponential gap between the weaker model and the standard 2NFAs, we *increase the power of* 2DFA*s*, towards 2NFAs, to the degree for which it is still possible to obtain a subexponential conversion from the stronger model to the standard 2DFAs. Such new stronger model then clearly shows that, in order to prove an exponential gap between 2NFAs and 2DFAs, one must use capabilities not allowed in the proposed new model. More precisely, in our new model, we neither restrict the cardinality of input alphabets, nor put any constraint on the head movement, i.e., head reversals can take place at any input position. On the other hand, we permit nondeterministic choices only when the input head is scanning one of the endmarkers. We shall call such machine a *two-way outer-nondeterministic finite automaton* (2ONFA).

It turns out that this machine has its natural counterpart also in the case of *two-way alternating finite automata* (2AFAs), which is a *two-way outer-alternating finite automaton* (2OAFA), making universal and existential choices only at the endmarkers. (For recent results on 2AFAs, see [14,6].)

We show that several results obtained for *unary* 2NFAs can be extended to 2ONFAs, and some of them even to the alternating version, 2OAFAs, with *any* input alphabet. In particular, we prove the following:

– Each $n$-state 2ONFA can be simulated by a *halting* two-way *self-verifying* automaton (2SVFA) [5] with $O(n^8)$ many states. This fact has two important implications:
  - The complementation of 2ONFAs can be done by using a polynomial number of states. Note the contrast with the above mentioned case of sweeping 2NFAs studied in [13].
  - Each 2ONFA can be simulated by a *halting* 2ONFA using a polynomial number of states.
– Each $n$-state 2ONFA can be simulated by a 2DFA with $O(n^{\log_2 n+6})$ states.
– If L = NL, then each $n$-state 2ONFA can be simulated by a 2DFA with a number of states polynomial in $n$. Hence, a superpolynomial lower bound for the simulation of 2ONFAs by 2DFAs would imply L $\neq$ NL. (Unlike in [2], there are no restrictions on the length of potential witness inputs.)
– Each $n$-state 2ONFA can be simulated by an *unambiguous* 2ONFA with a polynomial number of states.
– If L = P, then each $n$-state 2OAFA can be simulated by a 2DFA with a number of states polynomial in $n$, with the same consequences as presented

for L $\stackrel{?}{=}$ NL. (P denotes, as usual, the class of languages recognizable by deterministic Turing machines in polynomial time.)

– Similarly, if NL = P, we get the corresponding polynomial conversion from 2OAFAs to 2NFAs.

These results are obtained by generalizing the constructions given in [7,8,9] for the unary case. However, here we do not have a normal form for simplifying the automata, by restricting input head reversals to the endmarkers. Our generalization rely on a different tool, presented in the first part of the paper. Basically, we extend some techniques developed originally for deterministic devices [21,8] to machines with nondeterminism at the endmarkers. This permits us to check the existence of certain computation paths, including infinite loops, by the use of a linear number of states.

Due to the lack of space, in this version of the paper the proofs are omitted or just outlined. See `http://arxiv.org/abs/1110.1263` for an extended version.

## 2 Preliminaries

Let us start by briefly recalling some basic definitions from automata theory. For a detailed exposition, we refer the reader to [10]. Given a set $S$, $|S|$ denotes its cardinality and $2^S$ the family of all its subsets.

A *two-way nondeterministic finite automaton* (2NFA, for short) is defined as a quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_{\text{I}}, F)$, in which $Q$ is a finite set of states, $\Sigma$ is a finite input alphabet, $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q\times\{-1,0,+1\}}$ is a transition function, where $\vdash, \dashv \notin \Sigma$ are two special symbols, called the left and the right endmarkers, respectively, $q_{\text{I}} \in Q$ is an initial state, and $F \subseteq Q$ is a set of final states. The input is stored onto the input tape surrounded by the two endmarkers, the left endmarker being at the position zero. In one move, $\mathcal{A}$ reads an input symbol, changes its state, and moves the input head one position forward, backward, or keeps it stationary depending on whether $\delta$ returns $+1$, $-1$, or $0$, respectively. The machine accepts the input, if there exists a computation path from the initial state $q_{\text{I}}$ with the head on the left endmarker to some final state $q \in F$. The language accepted by $\mathcal{A}$, denoted by $L(\mathcal{A})$, consists of all input strings that are accepted. $\mathcal{A}$ is said to be *halting* if each computation ends in a finite number of steps. Observing that if an accepting computation visits the same endmarker two times in the same state then there exists a shorter accepting computation on same input, we immediately get the following lemma, which will be used in the proofs of some of our results:

**Lemma 1.** *If a* 2NFA $\mathcal{A}$ *with* $n$ *states accepts an input* $w$*, then it also accept* $w$ *with a computation that visits the left (right) endmarker at most* $n$ *times.*

Throughout the paper, given a 2NFA $\mathcal{A}$, we will call *computation segment* (or just segment) from $p$ to $q$ on $w$, each computation path on an input $w$ that starts at the left endmarker in the state $p$, ends at the left endmarker in the state $q$ and never visits the same endmarker in the meantime.

A *sequence of* $t \geq 0$ *segments* from a state $p$ to a state $q$ on input $w$ is a sequence of segments such that there are states $p_0, p_1, \ldots, p_t$, with $p_0 = p$, $p_t = q$, and the $i$th segment is from $p_{i-1}$ to $p_i$, $i = 1, \ldots, t$.

A 2NFA $\mathcal{A}$ is said to be *deterministic* (2DFA), whenever $|\delta(q, \sigma)| \leq 1$, for any $q \in Q$ and $\sigma \in \Sigma \cup \{\vdash, \dashv\}$, $\mathcal{A}$ is called *unambiguous* (2UFA), if there exists at most one accepting computation path for each input. A two-way *self-verifying* automaton (2SVFA) $\mathcal{A}$ is a 2NFA which, besides the set of accepting states $F \subseteq Q$, is equipped also with a disjoint set of rejecting states $F^r \subseteq Q$. For each input $w \in L(\mathcal{A})$ there exists one computation path halting in a state $q \in F$, and no path may halt in a state $q \in F^r$. Conversely, for $w \notin L(\mathcal{A})$ there exists one computation path halting in a state $q \in F^r$, and no path may halt in a state $q \in F$. Note that some computation paths of a 2SVFA may end with a "don't-know" answer, by ending in one state not belonging to $F \cup F^r$, or entering into an infinite loop.

An automaton working over a single letter alphabet is called *unary*.

A *two-way outer-nondeterministic finite automaton* (2ONFA, for short) is a 2NFA $\mathcal{A} = (Q, \Sigma, \delta, q_{\text{I}}, F)$ that can take nondeterministic decisions only when the input head is scanning one of the two endmarkers, i.e., for each $q \in Q$, $a \in \Sigma$, $|\delta(q, a)| \leq 1$. Actually, with a linear increasing in the number of the states, we can further restrict the use of the nondeterminism to the left endmarker only and obtain other simplifications:

**Lemma 2.** *For any* $n$*-state* 2ONFA $\mathcal{A} = (Q, \Sigma, \delta, q_{\text{I}}, F)$ *there exists an equivalent* 2ONFA $\mathcal{A}'$ *with no more than* $3n$ *states that satisfies the following properties:*

1. *nondeterministic choices are taken only when the input head is scanning the left endmarker,*
2. *there is a unique accepting state* $q_{\text{F}}$ *and this state is also halting,*
3. $q_{\text{F}}$ *is reachable only at the left endmarker by stationary moves,*
4. *stationary moves can occur only at the left endmarker to enter the state* $q_{\text{F}}$*.*

We assume that the reader is familiar with the standard Turing machine model and the basic facts from space and complexity theory, in particular with the classes L, NL, and P. For more details, see e.g. [10,23].

## 3 The Subroutine Reach

This section is devoted to develop a tool which will be fundamental in the proof of our results. Given a 2ONFA $\mathcal{A}$ with $n$ states, we show the existence of a subroutine Reach that receives as parameters two states $q', q''$ of $\mathcal{A}$ and decides whether or not $\mathcal{A}$ has a computation segment from $q'$ to $q''$ on an input string $w$. This subroutine can be implemented using a deterministic finite state control with $O(n)$ states whose input $w$ is stored on a two-way tape.

At a first glance, we could try to compute Reach$(q', q'')$ by initializing the automaton $\mathcal{A}$ in the state $q'$ with the input head at the left endmarker and by stopping its computation as soon as the input head, in one of the following steps, reaches again the left endmarker, then testing whether or not the state so reached is $q''$. However, this approach presents two problems: first of all, the original automaton $\mathcal{A}$ could enter into a infinite loop, never coming back to the left endmarker; second, the first move from the state $q'$ on the left endmarker can be nondeterministic.

In order to solve the first problem, we adapt the construction given in [8] to transform each $n$-state 2DFA into an equivalent halting $4n$-state 2DFA, which, it turns, was a refinement of the construction obtained by Sipser [21] to make space bounded Turing machines halting. We give a brief outline. For each $w \in \Sigma^*$, a deterministic machine accepts $w$ if and only if there is a "backward" path, following the history of the computation in reverse, from the unique accepting configuration $c_f$ to the unique initial configuration $c_0$. In our setting, a "configuration" is a pair $(s, i)$, where $s \in Q$ is a state and $i \in \{0, \ldots, |w| + 1\}$ is an input head position.

Consider the graph whose nodes represent configurations and edges computation steps. If the machine under consideration is deterministic and the accepting configuration is also halting, then no backward path starting from $c_f$ can cycle (hence, it is of finite length). Thus, the component of the graph containing the accepting configuration $c_f$ is a tree rooted at this configuration, with backward paths branching to all possible predecessors of $c_f$. Thus, the equivalent halting machine can perform a depth-first search of this tree in order to detect whether the initial configuration $c_0$ belongs to the predecessors of $c_f$. If this is the case, then the simulator accepts. On the other hand, if all the tree is examined without reaching $c_0$, this means that there are no paths from $c_0$ to $c_f$ and so $w$ is not in the language. Hence, the simulator rejects.

We adapt such a procedure by choosing $c_0 = (q', 0)$ and $c_f = (q'', 0)$, where $q'$ and $q''$ are the two parameters. Furthermore, since we are interested to detect the existence of *just one segment* from $q'$ to $q''$, we do not consider the transitions on the left endmarker from states different than $q'$. However, the remaining transition, i.e., that from configuration $c_0$, can be nondeterministic. Thus, we need further modifications. Suppose $\delta(q', \vdash) = \{(q_1, +1), \ldots, (q_k, +1)\}$. Hence, there is a segment from $c_0$ to $c_f$ if and only if there is a path from $c_j = (q_j, 1)$ to $c_f$, for some $j \in \{1, \ldots, k\}$, visiting the left endmarker only in $c_f$. Hence, the backward search can be done starting from $c_f$, without considering all the transitions of the original automaton from the left endmarker, stopping and accepting when one of the $c_j$'s is reached, or rejecting when all the tree has been visited without reaching any of the $c_j$'s.

To do that, our procedure needs to detect when the head position of the original 2ONFA $\mathcal{A}$ is scanning the first "real" input symbol, i.e., that immediately to the right of the left endmarker, in cell number 1. By a closer look to the simulation in [8] and to its implementation it is possible to show how this can be done. In particular, this construction can be implemented by a deterministic finite state control with $4n - 3$ states (not counting the space needed to store the parameters $q', q''$, which will be used in a "read-only" way). This gives us the following result which will be a fundamental tool in the next sections:

**Lemma 3.** *For each* 2ONFA $\mathcal{A}$ *with* $n$ *states (in the form of Lemma 2), it is possible to construct a deterministic finite state control with* $4n-3$ *states that given an input string* $w \in \Sigma^*$ *stored on a two-way tape and two states* $q', q''$ *of* $\mathcal{A}$ *decides whether or not the automaton* $\mathcal{A}$ *has a computation segment from* $q'$ *to* $q''$ *on* $w$*.*

## 4 Simulation by Halting Self-verifying Automata

In this section we prove that for each $n$-state 2ONFA $\mathcal{A}$ accepting a language $L$ there exists an equivalent *halting* 2SVFA using a polynomial number of states and making nondeterministic choices only when the input head is scanning the left endmarker. As a consequence, we can derive halting 2ONFAs with polynomial many states that accept $L$ and the complement of $L$.

Even in this case, our starting point is a proof given in [8] for the unary case, which was based on the well known technique of the *inductive counting*. However, there are deep differences. In particular, the proof in [8] uses a normal form for unary 2NFAs in which a computation is a sequence of deterministic traversals of the input from one endmarker to the opposite one. In this normal form there are no parts of computations starting and ending at the same endmarker without visiting the other one in the meantime (these parts are usually called *U-turns*). Furthermore, the only possible infinite loops involve the endmarkers and can be easily avoided using Lemma 1. The simulation inductively counts, for increasing values of $t$, how many states are reachable from the initial state in $t$ traversals of the input and, as a side effect of this counting procedure, also lists these states. In this way, after a certain number of traversals, all the states which are reachable at the endmarkers have been listed and, hence, it is possible to decide whether or not the input was accepted by the original machine.

In the case we are considering, we do not have such a kind of normal form. Hence, a computation can present traversals of the input from one endmarker to the opposite one as well as $U$-turns. Furthermore, a computation can reject by entering into a infinite loop: in this case infinite loops which never visits the endmarkers are also possible. To overcome the first problem, instead of traversals, our inductive counting procedure considers computation segments: for increasing values of $t$, it counts how many states are reachable from the initial state in a computation consisting of $t$ segments, i.e., visiting the left endmarker $t+1$ times, and, at the same time, the procedure lists these states. Since, as stated in Lemma 1, each accepted input has an accepting computation which visits the left endmarker at most $n$ times, it is enough to consider computations consisting of at most $n-1$ segments. This avoid infinite loops involving the endmarkers.

The remaining infinite loops are removed using the subroutine REACH, discussed in Section 3 and another subroutine tREACH which with parameters $q \in Q$ and $t \geq 0$ verifies the existence of a sequence of $t$ segments from the initial state $q_{\rm I}$ to $q$ on the input under consideration. In negative case, tREACH aborts the simulation in the "don't-know" state $q_?$, otherwise the subroutine returns to the main simulation in a different state. This subroutine is nondeterministic. It

**Algorithm 1.** *Simulation of* 2ONFA*s by* 2SVFA*s*

```
1  $m' := 1$
2  for $t := 0$ to $|Q| - 2$ do
3      $m := m'$; $m' := 0$
4      foreach $q' \in Q$ do
5          for $i := 1$ to $m$ do
6              $q :=$ a nondeterministically chosen state
7              if ($i > 1$ and $q \leq q_{\text{prev}}$) then halt in $q_?$
8              $q_{\text{prev}} := q$
9              if tREACH$(q, t)$ and REACH$(q_{\text{prev}}, q')$ then
                               // possible side effect of tREACH: abort in $q_?$
10                 if $q' = q_{\text{F}}$ then halt in $q_{\text{yes}}$
11                 $m' := m' + 1$
12                 break
13 halt in $q_{\text{no}}$
```

can also halt in $q_?$, due to a wrong sequence of nondeterministic guesses. However, if there exists a sequence of $t$ segments from $q_{\text{I}}$ to $q$, the subroutine has at least one computation ending in a state other than $q_?$. The subroutine tREACH is implemented by iterating $t$ times a nondeterministic variant of REACH which, given a parameter $q''$, returns a nondeterministic chosen state $q'$ such that $\mathcal{A}$ has a computation segment form $q'$ to $q''$.

The inductive counting procedure is given in Algorithm 1. At each iteration of the main loop (line 2), from the number of states reachable at the left endmarker by all computation paths with exactly $t$ segments (stored in the variable $m$), the number of states reachable upon completing one more segment is counted in the variable $m'$. A linear ordering $\leq$ is assumed on the set of states. Boolean operators are evaluated in a "lazy" way. This implies that the condition $q \leq q_{\text{prev}}$ is evaluated only for $i > 1$, hence after assigning a value to $q_{\text{prev}}$.

We can prove that: (i) if the input is accepted by $\mathcal{A}$, at least one computation path halts in the state $q_{\text{yes}}$, and no path halts in $q_{\text{no}}$, (ii) if the input is rejected, at least one path halts in $q_{\text{no}}$, and no path halts in $q_{\text{yes}}$, (iii) due to wrong sequences of nondeterministic guesses, some computation paths halt in the "don't-know" state $q_?$, but no path can get into an infinite loop.

Our implementation produces an halting automaton with $O(n^8)$ states. Furthermore, in the main algorithm and in the subroutines all the nondeterministic choices are taken with the input head scanning the left endmarker. Hence:

**Theorem 4.** *Each $n$-state* 2ONFA *can be simulated by an equivalent halting $O(n^8)$-state* 2SVFA *making nondeterministic choices only when the input head is scanning the left endmarker.*

**Corollary 5.** *For each $n$-state* 2ONFA $\mathcal{A}$ *there exist an equivalent halting* 2ONFA $\mathcal{A}'$ *with $O(n^8)$ states and a* 2ONFA $\mathcal{A}''$ *with $O(n^8)$ states accepting the complement of the language accepted by $\mathcal{A}$.*

**Algorithm 2.** *Recursive function* Reachable$(q, p, t)$

```
14 if $t = 1$ then return ($q = p$ or Reach$(q, p)$)
15 else
16     foreach state $r \in Q$ do
17         if Reachable$(q, r, \lceil t/2 \rceil)$ then
18             if Reachable$(r, p, \lceil t/2 \rceil)$ then return true
19     return false
```

## 5 Subexponential Deterministic Simulation

In this section, we prove that each 2onfa with $n$ states can be simulated by an equivalent 2dfa with $O(n^{\log_2 n+6})$ states, i.e., with a subexpontential, but still superpolynomial, number of states. In the authors' knowledge this is the first case of a model using nondeterminism and an unrestricted alphabet, having a subexponential simulation by 2dfas.

This result generalizes a result proved for the unary case in [7]. The new "ingredient" in our version is the subroutine Reach presented in Section 3. So we give a very short presentation, addressing the reader to [7] for further details.

Let $\mathcal{A}$ be a 2onfa with $n$ states in the form of Lemma 2. The 2dfa simulating $\mathcal{A}$ implements a recursive function, called Reachable, based on the well known divide-and-conquere technique (see Algorithm 2). The function receives three parameters: two states $q$ and $p$ and an integer $t \geq 1$, and returns a boolean. On these parameters, Reachable$(q, p, t)$ returns true if and only if on the input $w$ under consideration the automaton $\mathcal{A}$ has a sequence of *at most* $t$ segments from the state $q$ to the state $p$. Hence, according to Lemma 1, to decide whether or not $w$ is accepted by $\mathcal{A}$, we call Reachable$(q_{\text{I}}, q_{\text{F}}, n-1)$.

**Theorem 6.** *Each $n$-state* 2onfa *can be simulated by an equivalent* 2dfa *with $O(n^{\log_2 n+6})$ states.*

It is natural to wonder if the upper bound stated in Theorem 6 is optimal. We remind the reader that the best known lower bound for the number of the states of 2dfas simulating $n$-state 2nfas is $O(n^2)$ [4]. In the next section we will show that the optimality of the upper bound in Theorem 6 or any other superpolynomial state lower bound for the simulation of 2onfas by 2dfas would imply the separation between deterministic and nondeterministic logarithmic space, hence solving a longstanding open problem in structural complexity.

## 6 Conditional and Unambiguous Simulations

In this section we discuss how to reduce the language accepted by a given 2onfa to the *graph accessibility problem* (GAP), i.e., the problem of deciding whether a directed graph contains a path connecting two designated vertices. This problem is well known to be complete for NL. As a consequence of this reduction we will prove that the equality between the classes L and NL would imply a polynomial simulation of 2ONFAs by 2DFAs. Furthermore, we also prove that each 2ONFA can be made unambiguous with a polynomial increasing in the number of the states.

Let us start to present our reduction. As for the results in Sections 4 and 5, it is obtained by combining a technique developed for the unary case [9], with the use of the subroutine REACH presented in Section 3.

Given an $n$-state 2ONFA $\mathcal{A}$ in the form of Lemma 2, with each input $w \in \Sigma^*$ we associate the directed graph $G(w) = (Q, E(w))$, where $Q$ is the set of states of $\mathcal{A}$ and $E(w) = \{(p,q) \in Q \times Q \mid \text{REACH}(p,q) \text{ is true}\}$, i.e., it is the set of pairs $(p,q)$ of states such that $\mathcal{A}$ on input $w$ has a segment from $p$ to $q$. It should be clear that $w$ is accepted by $\mathcal{A}$ if and only if the graph $G(w)$ contains a path from vertex $q_\text{I}$, the initial state, to vertex $q_\text{F}$, the accepting state. Hence, this defines a reduction from the language accepted by $\mathcal{A}$ to GAP. As mentioned before, GAP is complete for NL under logarithmic space reductions [20]. Using this property and the reduction above described we are able to prove the following:[1]

**Theorem 7.** *If* L = NL *then each $n$-state* 2ONFA *can be simulated by a* 2DFA *with a number of states polynomial in $n$.*

A modification of the construction used to prove Theorem 7, allows us to obtain the next simulation, which does not require the condition L = NL:

**Theorem 8.** *Each $n$-state* 2ONFA *can be simulated by an* unambiguous 2ONFA *with a number of states polynomial in $n$.*

## 7 The Alternating Case

In this section we briefly discuss an extension of the technique used in Section 6, to the case of automata with *alternations* [3], recently considered in [14,6].

A *two-way alternating automaton* (2AFA, for short) is defined as a quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_\text{I}, F)$, exactly as a 2NFA. However, the set $Q$ is partitioned in two sets $Q_\exists$ and $Q_\forall$, the sets of *existential* and *universal* states, respectively. The acceptance of an input string $w$ is witnessed by an *accepting computation tree*. Even for 2AFAs, we can restrict the use of the nondeterminism as we did for 2NFAs, considering *outer-alternating finite automata* (2OAFAs). In these models, each configuration scanning an ordinary input symbol can have at most one successor, namely (existential or universal) nondeterministic choices can be taken only when the head is scanning one of the endmarkers. Actually, we can further restrict the use of the nondeterminism only at the left endmarker, proving a normal form similar to that of Lemma 2.

Now, we consider the *alternating graph accessibility problem* (AGAP, for short), an alternating version of GAP. The instance of the problem is an *alternating* direct graph, i.e., a graph $G = (V, E)$ with a partition of $V$ in two sets $V_\exists$ and $V_\forall$, and two designated vertices $s$ and $t$. The question is if the predicate

[1] The result in Theorem 7 is presented, in a different context, also in [17].

$apath(s,t)$ is true, where, for $x,y \in V$, $apath(x,y)$ holds true if and only if either (i) $x = y$, or (ii) $x \in V_\exists$ and exists $z \in V$ with $(x,z) \in E$ such that $apath(z,y)$ is true, or (iii) $x \in V_\forall$ and for all $z \in V$, $(x,z) \in E$ implies that $apath(z,y)$ is true. This problem is known to be complete for the class P, with respect to logarithmic space reductions [12]. As is Section 6, we can reduce the language accepted by a given 2OAFA $\mathcal{A}$ to AGAP, by associating with each input string $w$ the graph $G(w) = (Q, E(w))$, where $(p,q) \in E(w)$ if and only if $\mathcal{A}$ has a computation segment from $p$ to $q$ on input $w$. (The extension of the notion of computation segment to 2AFA is obvious.) Since the subroutine REACH presented in Section 3 depends only on the transition function of the given automaton $\mathcal{A}$ and not on the acceptance condition, we can use it to detect segments even in the case of outer 2AFAs. Using the fact that AGAP is complete for P we can prove:

**Theorem 9.** *If* L = P *then each* 2OAFA *can be simulated by a* 2DFA *with a polynomial number of states. If* NL = P *then each* 2OAFA *can be simulated by a* 2NFA *with a polynomial number of states.*

## 8 Concluding Remarks

In this paper we generalized, in a unified framework, some results previously obtained for unary 2NFAs to machines with arbitrary input alphabets, but making nondeterministic choices only at the input tape endmarkers. Among others, we have shown that any superpolynomial lower bound for the simulation of such machines by standard 2DFAs would imply L $\neq$ NL.We also related the alternating version of such machines to L $\overset{?}{=}$ NL $\overset{?}{=}$ P, the classical computational complexity open problems. Comparing our results with those obtained for other restricted models of two-way automata we observe that:

– Actually, unary 2NFAs can use only a restricted form of nondeterminism. In fact, we can restrict their nondeterminism to the endmarkers without increasing significantly their size [7].
– In the general case, the possibility of reversing the input head movement at any input position does not seem so powerful as the possibility of making nondeterministic decisions at any input position. (Compare our polynomial upper bound for the complementation of 2ONFAs with the exponential lower bound for the complementation of sweeping 2NFAs in [13].)
– However, in the deterministic case, the possibility of reversing the input head at any input position can make automata exponentially smaller than machines reversing the input head only at the endmarkers [1,18].

Our contructions mainly rely on the fact that for 2ONFAs a polynomial control is enough to keep information about nondeterministic choices. The same does not seem to be feasible for unrestricted 2NFAs. It would be interesting to see if the results proved in this paper could not be extended to a model using the nondeterminism in a less restricted way than the one considered here. In general, we believe that the study of restricted forms of nondeterminism in finite automata deserves further investigation.

## References

1. Berman, P.: A note on Sweeping Automata. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 91–97. Springer, Heidelberg (1980)
2. Berman, P., Lingas, A.: On the complexity of regular languages in terms of finite automata. Tech. Rep. 304, Polish Academy of Sciences (1977)
3. Chandra, A.K., Kozen, D., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981)
4. Chrobak, M.: Finite automata and unary languages. Theoretical Computer Science 47, 149–158 (1986); Errata: ibid 302, 497–498
5. Duris, P., Hromkovič, J., Rolim, J.D.P., Schnitger, G.: Las Vegas Versus Determinism for One-way Communication Complexity, Finite Automata, and Polynomial-time Computations. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 117–128. Springer, Heidelberg (1997)
6. Geffert, V.: An alternating hierarchy for finite automata. In: Non-Classical Models of Automata and Applications (NCMA 2011), pp. 15–36 (2011)
7. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic unary automata into simpler automata. Theor. Comput. Sci. 295, 189–203 (2003)
8. Geffert, V., Mereghetti, C., Pighizzini, G.: Complementing two-way finite automata. Inf. Comput. 205(8), 1173–1187 (2007)
9. Geffert, V., Pighizzini, G.: Two-way unary automata versus logarithmic space. Inf. Comput. 209(7), 1016–1025 (2011)
10. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
11. Hromkovič, J., Schnitger, G.: Nondeterminism Versus Determinism for Two-way Finite Automata: Generalizations of Sipser's Separation. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 439–451. Springer, Heidelberg (2003)
12. Immerman, N.: Number of quantifiers is better than number of tape cells. Journal of Computer and System Sciences 22(3), 384–406 (1981)
13. Kapoutsis, C.A.: Small Sweeping 2NFAs Are Not Closed Under Complement. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006, Part I. LNCS, vol. 4051, pp. 144–156. Springer, Heidelberg (2006)
14. Kapoutsis, C.A.: Size Complexity of Two-Way Finite Automata. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 47–66. Springer, Heidelberg (2009)
15. Kapoutsis, C.A.: Nondeterminism Is Essential in Small 2FAs with Few Reversals. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 198–209. Springer, Heidelberg (2011)
16. Kapoutsis, C.A.: Two-Way Automata versus Logarithmic Space. In: Kulikov, A.S., Vereshchagin, N.K. (eds.) CSR 2011. LNCS, vol. 6651, pp. 359–372. Springer, Heidelberg (2011)
17. Kapoutsis, C.A., Pighizzini, G.: Two-way automata characterizations of L/poly versus NL (2011) (submitted manuscript)
18. Micali, S.: Two-way deterministic finite automata are exponentially more succinct than sweeping automata. Information Processing Letters 12(2), 103–105 (1981)
19. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two way finite automata. In: STOC, pp. 275–286. ACM (1978)
20. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. J. Comput. Syst. Sci. 4(2), 177–192 (1970)

21. Sipser, M.: Halting space-bounded computations. Theor. Comput. Sci. 10, 335–338 (1980)
22. Sipser, M.: Lower bounds on the size of sweeping automata. J. Comput. Syst. Sci. 21(2), 195–202 (1980)
23. Szepietowski, A.: Turing Machines with Sublogarithmic Space. LNCS, vol. 843. Springer, Heidelberg (1994)

# Polynomial-Time Algorithms for Learning Typed Pattern Languages*

Michael Geilke[1] and Sandra Zilles[2]

[1] Fachbereich Informatik, Technische Universität Kaiserslautern
D-67653 Kaiserslautern, Germany
geilke.michael@gmail.com

[2] Department of Computer Science, University of Regina
Regina, SK, Canada S4S 0A2
zilles@cs.uregina.ca

**Abstract.** This article proposes polynomial-time algorithms for learning typed pattern languages—formal languages that are generated by patterns consisting of terminal symbols and typed variables. A string is generated by a typed pattern by substituting all variables with strings of terminal symbols that belong to the corresponding types.

The algorithms presented consitute non-trivial generalizations of Lange and Wiehagen's efficient algorithm for learning patterns in which variables are not typed. This is achieved by defining *type witnesses* to impose structural conditions on the types used in the patterns. It is shown that Lange and Wiehagen's algorithm implicitly uses a special case of type witnesses. Moreover, the type witnesses for a typed pattern form characteristic sets whose size is linear in the length of the pattern; our algorithm, when processing any set of positive data containing such a characteristic set, will always generate a typed pattern equivalent to the target pattern. Thus our algorithms are of relevance to the area of grammatical inference, in which such characteristic sets are typically studied.

## 1 Introduction

Since Dana Angluin [1] introduced the pattern languages, they have been a popular object of study in the area of algorithmic learning theory and formal language theory. A pattern is a string consisting of terminal and variable symbols; its language is the set of all words that can be generated when replacing variables with non-empty strings of terminal symbols. Part of the reason for their popularity is that patterns provide an intuitive and simple way of representing rather complex but structured languages; moreover they can be thought of as a model for text mining applications.

It is known that the class of all pattern languages is learnable from positive data (i.e., if the learning algorithm sees only words contained in the target pattern language) in Gold's model of identification in the limit [7,1]. The

* This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

first polynomial-time algorithm for learning pattern languages was presented by Lange and Wiehagen [10].

Unfortunately, most interesting text mining applications require a pattern model in which the variables are typed, i.e., the set of strings that can be substituted for a variable in a pattern is restricted. This is only natural since database entries are typically typed as well. Koshiba [9] hence introduced the model of typed pattern languages, in which each variable in a pattern has an associated type that defines the set of allowed substitutions for the variable. After Koshiba's initial results on learning typed pattern languages, recent progress was made on learning relational pattern languages (a generalization of typed pattern languages) [6]. In particular, learnability of relational and thus of typed pattern languages from positive data was shown in variations of Gold's learning model that at least partly address the issue of efficiency in terms of the amount of data presented to the learning algorithm [6].

The present paper focuses on the problem of learning typed pattern languages in polynomial time, in particular on generalizing Lange and Wiehagen's algorithm to the case of typed pattern languages. This is not only of interest for both practical and theoretical reasons, but it also constitutes a non-trivial task. Lange and Wiehagen's algorithm exploits only the shortest words contained in the given data and crucially relies on the fact that these words are formed when variables are replaced by strings of length one. Moreover, it relies on the fact that there are two distinct strings of length one that can be replaced for each variable (see Section 2.2 for more background on Lange and Wiehagen's algorithm). Neither property can be guaranteed when dealing with typed patterns.

We hence introduce the notion of *type witness* to impose certain structural conditions on the types used in the patterns. We demonstrate that

– Lange and Wiehagen's algorithm implicitly uses a special kind of type witness according to our definition,
– type witnesses allow for learning the corresponding typed pattern languages in polynomial time, using a generalization of Lange and Wiehagen's algorithm,
– type witnesses for a typed pattern form characteristic sets [3,8] whose size is linear in the length of the pattern; our algorithm, when processing any set of positive data containing such a characteristic set, will always generate a typed pattern equivalent to the target pattern.[1]

Some of our results also address the important question of polynomial-time learning with a *consistent* algorithm that always hypothesizes patterns that generate all of the positive input data seen [4]. Neither Lange and Wiehagen's algorithm nor our proposed algorithms have this property in general, but we identify special cases in which consistent behaviour can be achieved without sacrificing runtime efficiency.

This article is the first one to design polynomial-time algorithms for learning typed pattern languages. It contributes to the state-of-the-art as follows:

[1] Characteristic sets play an important role in the area of grammatical inference.

- Our generalized study of Lange and Wiehagen's algorithm provides new theoretical insights into the difficulties of efficiently learning patterns in application-relevant scenarios, and it proposes new algorithmic solutions.
- The definition of type witnesses is likely to be of value for further studies on learning typed pattern languages and their generalizations, relational pattern languages.
- The connection to the use of characteristic sets bridges gaps between the inductive inference community (in which pattern languages have mostly been studied so far) and the grammatical inference community.

## 2 Preliminaries and Background

Languages are defined with respect to a non-empty *alphabet* $\Sigma$. A *word* $w$ is a finite, possibly empty, sequence of symbols from $\Sigma$ the length of which is denoted by $|w|$. $\epsilon$ refers to the empty word, *i.e.*, the word of length 0. The set of all words over $\Sigma$ is denoted by $\Sigma^*$, and the set of all non-empty words over $\Sigma$ by $\Sigma^+$; hence $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$. A *language* $L$ is a subset of $\Sigma^*$. By $w_1 \circ w_2$ we denote the concatenation of two words $w_1$ and $w_2$ (where, for ease of presentation, we allow $w_1$ and/or $w_2$ to be written as $\sigma \in \Sigma$ rather than a word $(\sigma)$ of length 1). In what follows, we always assume $\Sigma$ to be a finite set of cardinality at least 2.

For any $w, v \in \Sigma^*$, we say that $v$ is a *prefix* of $w$ (*suffix* of $w$, resp.), if $w = v \circ w'$ ($w = w' \circ v$, resp.) for some $w' \in \Sigma^*$. A prefix (suffix, resp.) of $w$ is called *proper* if it is distinct from $w$. If $A \subseteq \Sigma^*$ and $w \in \Sigma^*$, we write $w \sqsubseteq A$ ($w \sqsubset A$, resp.) if $w$ is a prefix (proper prefix, resp.) of some word in $A$.

For $A \subseteq \Sigma^*$ and $w \in \Sigma^*$, we call $w$ *$A$-decomposable*, if $w$ can be written as $w = w_1 \circ \cdots \circ w_m$ for some $m \geq 1$, where $w_1, \ldots, w_m \in A$. $(w_1, \ldots w_m)$ is then called an *$A$-decomposition* of $w$; the length of this $A$-decomposition is $m$.

### 2.1 Pattern Languages and Typed Pattern Languages

A class of languages that has been studied in the formal language theory community as well as in the learning theory community is Angluin's class of non-erasing pattern languages [1], defined as follows. Let $X = \{x_1, x_2, \ldots\}$ be a countable set of symbols called variables; we require that $X$ be disjoint from $\Sigma$. A *pattern* is a non-empty finite string over $\Sigma \cup X$. The set of all patterns over $\Sigma \cup X$ will be denoted by $Pat_\Sigma$. A *substitution* is a string homomorphism $\theta : Pat_\Sigma \to \Sigma^+$ that is the identity when restricted to $\Sigma$. The set of all substitutions with respect to $\Sigma$ is denoted by $\Theta_\Sigma$. The *(non-erasing) language* $L(p)$ of a pattern $p \in (\Sigma \cup X)^+$ is defined by $L(p) = \{w \in \Sigma^+ \mid \exists \theta \in \Theta_\Sigma\ [\theta(p) = w]\}$, *i.e.*, it consists of all words that result from substituting all variables in $p$ by non-empty words over $\Sigma$. The class $\mathcal{L}_\Sigma$ of (non-erasing) pattern languages is defined by $\mathcal{L}_\Sigma = \{L(p) \mid p \in Pat_\Sigma\}$.

Thus patterns constitute a simple and intuitive means of describing formal languages of a particular kind of structure. From a practical point of view, pattern languages are interesting because in applications one can think of a set of database entries as being generated by a fixed underlying pattern.

Shinohara [13] defined erasing pattern languages by allowing substitutions by the empty word, but in this article we focus on non-erasing languages exclusively. Wright [14] studied a subclass of erasing pattern languages under restricted substitutions, leading to Koshiba's typed pattern languages [9]. In Koshiba's model, each variable $x$ in a pattern $p$ is assigned a particular *type* $T(x)$, which is a decidable subset of $\Sigma^+$. The words generated by $p$, when respecting the types, are formed by substituting any variable $x$ in $p$ only with words from $T(x)$, resulting in the typed pattern language $L(p, T)$.

**Definition 1.** *A* typed pattern *over $\Sigma$ is a pair $(p, T)$, where $p$ is a pattern over $\Sigma$ and $T : X \to \{S \subseteq \Sigma^+ \mid S \text{ is decidable}\}$ is a mapping that associates each variable with a recursive language. For each $x \in X$, $T(x)$ is called the* type *of $x$. A $T$-typed substitution is a string homomorphism $\theta : Pat_\Sigma \to \Sigma^+$ that is the identity when restricted to $\Sigma$ and that fulfills $\theta(x) \in T(x)$ for all $x \in X$. The set of all $T$-typed substitutions with respect to $\Sigma$ is denoted by $\Theta_{\Sigma,T}$. The language of a typed pattern $(p, T)$, denoted by $L(p, T)$, is defined by $L(p, T) = \{w \in \Sigma^+ \mid \exists\theta \in \Theta_{\Sigma,T}\ [\theta(p) = w]\}$.*

Types make pattern languages more expressive and more suitable for applications. For example, a system for entering bibliographic data as described by Shinohara [13] might use patterns like $p =$ `author:` $x_1$ `title:` $x_2$ `year:` $x_3$. One would expect $x_3$ to be substituted only by certain two or four digit integers—a property that becomes expressible when using types.

For the sake of simplicity, we assume that the range of $T$ is finite. However, all of the results we present below generalize very easily to the case that the range of $T$ is infinite, under minor conditions explained in Section 5.1.

**Definition 2.** *Let $\mathcal{T}$ be a finite set of decidable subsets of $\Sigma^+$. The class of all $\mathcal{T}$-typed pattern languages is the class of all languages $L(p, T)$ where $(p, T)$ is a typed pattern and $T(x) \in \mathcal{T}$ for all $x \in X$.*

## 2.2 Learnability

In Gold's model of learning in the limit from positive data [7], a class of languages is learnable if there is a learner that "identifies" every language in the class from any of its texts, where a text for a language $L$ is an infinite sequence $\tau(0), \tau(1), \tau(2), \ldots$ of words such that $\{\tau(i) \mid i \in \mathbb{N}\} = L$.

**Definition 3 (Gold [7]).** *Let $\mathcal{L}$ be a class of languages. $\mathcal{L}$ is learnable in the limit from positive data if there is a hypothesis space $\{L_i \mid i \in \mathbb{N}\} \supseteq \mathcal{L}$ and a partial recursive mapping $\mathcal{A}$ such that, for any $L \in \mathcal{L}$ and any text $(\tau(i))_{i\in\mathbb{N}}$ for $L$, $\mathcal{A}(\tau(0), \ldots, \tau(n))$ is defined for all $n$ and there is a $j \in \mathbb{N}$ with $L_j = L$ and $\mathcal{A}(\tau(0), \ldots, \tau(n)) = j$ for all but finitely many $n$.*

Below we will focus on *polynomial-time learning*, i.e., learning by an algorithm that computes its hypotheses in time polynomial in the length of its input.[2]

[2] This notion of efficient learning does not refer to the number of text examples needed before convergence; for a critical treatment of notions of efficiency in learning, the reader is referred to the work by Pitt [11] and Case and Kötzing [5].

Angluin showed that $\mathcal{L}_\Sigma$ is learnable [1]. Most intuitive learning algorithms, including the one proposed by Angluin, conduct a large number of membership tests ("does word $w$ belong to the language $L(p)$?") in order to identify pattern languages from text. Since the membership problem for non-erasing pattern languages is NP-complete [1], polynomial-time learning cannot be guaranteed by this kind of algorithm.

Lange and Wiehagen [10] designed an algorithm that learns $\mathcal{L}_\Sigma$ in polynomial time. At any point in the learning process, their algorithm uses only the shortest words presented in the text so far to build a hypothesis pattern $p_h$; all other words in the text are ignored. For any $i$, if a symbol $\sigma \in \Sigma$ occurs in the $i$-th position in all the shortest words presented so far, the algorithm sets the $i$-th position of $p_h$ to $\sigma$. Otherwise, the $i$-th position of $p_h$ is a variable. If both position $i$ and position $j$ in $p_h$ contain a variable and the symbol in position $i$ of any shortest word $w$ in the given text segment equals the symbol in position $j$ of $w$, then the variables in positions $i$ and $j$ in $p_h$ are chosen to be identical. For example, if the shortest words collected are

$$\begin{array}{cccccc} a & a & a & bb & a & ab \\ a & a & b & bb & a & ab \\ a & b & a & bb & b & ab \end{array}$$

then the pattern $p_h$ hypothesized by the algorithm is $ax_1x_2bbx_1ab$.

This algorithm does not work for erasing pattern languages; in fact, for $|\Sigma| \in \{2,3,4\}$, the class of all erasing pattern languages is not learnable at all [12], not even by an inefficient learning algorithm. Moreover, in general, neither erasing nor non-erasing typed pattern languages are learnable (cf. [9]), let alone polynomial-time learnable. The main contribution of this paper is the design of polynomial-time algorithms for learning interesting subclasses of (non-erasing) typed pattern languages, where certain structural properties of the potential target patterns are assumed. In order to define appropriate structural properties, we introduce the notion of *type witness* below.

Unfortunately, our above definition of polynomial-time learning is of no consequence. Pitt [11] showed that by repeating previous hypotheses until the input has grown long enough to afford computing a new hypothesis, a polynomial-time learner can be constructed from any learner successful according to Definition 3. But Lange and Wiehagen's algorithm $\mathcal{A}$ learns efficiently in a much stronger sense: every non-erasing pattern language $L$ possesses a finite subset $S_L$ of size polynomial in the length of any pattern $p$ with $L = L(p)$ such that $\mathcal{A}$, on input of any text segment for $L$ containing at least all words in $S_L$, returns a correct hypothesis for $L$. Such a subset $S_L$ is what de la Higuera calls a characteristic set of polynomial size [8,3], which, for learning from positive data, corresponds to a tell-tale set [2]. Deviating slightly from de la Higuera's model, we define:

**Definition 4 (cf. de la Higuera [8]).** *Let $\mathcal{L}$ be a class of languages and $\mathcal{H}$ a set of representations such that for each $H \in \mathcal{H}$ there is exactly one $L_H \in \mathcal{L}$ associated with $H$. Let $||H||$ denote the size of a representation $H \in \mathcal{H}$. $\mathcal{H}$ is polynomially learnable from positive data if there exist two polynomials $\chi$ and $\xi$ and an algorithm $\mathcal{A}$ such that the following conditions are fulfilled.*

1. *Given any finite set $S \subseteq \Sigma^*$, $\mathcal{A}$ returns some $H \in \mathcal{H}$ in time at most $\chi(|S|)$.*
2. *For any $H \in \mathcal{H}$ there is a finite set $S_H \subseteq L_H$ with $|S_H| \leq \xi(||H||)$ such that, on input of any set $S$ with $S_H \subseteq S \subseteq L_H$, $\mathcal{A}$ returns some $H' \in \mathcal{H}$ with $L_{H'} = L_H$. $S_H$ is called a characteristic set for $H$ with respect to $\mathcal{H}$.*

De la Higuera additionally requires $\mathcal{A}$ to be consistent [4], i.e., to produce only hypotheses that contain the input set $S$. We drop this condition in Definition 4. Obviously, if $\mathcal{L}$ has an effectively enumerable representation set $\mathcal{H}$ that is polynomially learnable from positive data according to Definition 4, then $\mathcal{L}$ is also learnable in the limit from positive data. For example, the set of all patterns may serve as a representation set for the set of all pattern languages.

A characteristic set $S_p$ for a pattern $p$, for each variable occurring in the target pattern $p$, contains two shortest words in which that variable is replaced differently and all other variables are replaced by some fixed $\sigma \in \Sigma$:

**Proposition 5.** *Let $p \in Pat_\Sigma$. Then there is a subset $S_p$ of $L(p)$ with $|S_p| \leq 2|p|$ such that $S_p$ is a characteristic set with respect to $Pat_\Sigma$ for Lange and Wiehagen's algorithm.*

*Proof.* Let $\sigma_1, \sigma_2 \in \Sigma$, $\sigma_1 \neq \sigma_2$. Let $S_p$ be a subset of $L(p)$ consisting of two words $v_1(x) = \theta_1^x(p)$ and $v_2(x) = \theta_2^x(p)$ per variable $x$ occurring in $p$, where

$$\theta_\ell^x(x_i) = \begin{cases} \sigma_\ell\,, & \text{if } x_i = x\,, \\ \sigma_1\,, & \text{if } x_i \neq x\,, \end{cases}$$

for $\ell \in \{1, 2\}$. Clearly, $|S_p| \leq 2|p|$ and $S_p$ contains only shortest words from $L(p)$.

Lange and Wiehagen's algorithm on input $S$, $S_p \subseteq S \subseteq L(p)$, can identify the positions in which variables reside (because in each such position in the words in $S_p$ both $\sigma_1$ and $\sigma_2$ occur) and the pairs of positions in which two distinct variables reside (because in these pairs of positions at least one word in $S_p$ has distinct entries). Thus $S_p$ is a characteristic set for $p$ with respect to $Pat_\Sigma$. □

## 3 Type Witnesses

The key to Lange and Wiehagen's algorithm is that a position in a shortest word of a non-erasing pattern language corresponds to the same position in the underlying pattern. For typed pattern languages, this is no longer the case, since the types might not contain any words of length one, and thus the shortest words of the language might be longer than the underlying pattern itself.

The shortest words in Lange and Wiehagen's case are those that are generated by substitutions replacing a variable with a word of length 1. Since, for $\Sigma \geq 2$, there are at least 2 distinct words of length 1, a learner can eventually figure out which positions of the shortest words in the given text correspond to variables and which variable positions correspond to repeated variables. We generalize this idea by using *type witness* words instead of words of length 1. These words will allow the learner to recognize repetitions of variables. Just as there are 2 distinct words of length 1, we need 2 distinct type witness words per type. For the learner to be able to distinguish types, we require the type witness words to belong only to their corresponding type. This is formalized as follows.

**Definition 6.** *Let $\mathcal{T}$ be a set of subsets of $\Sigma^+$. Let $\omega_1, \omega_2 : \mathcal{T} \to \Sigma^+$ be mappings and $W = \bigcup_{t \in \mathcal{T}} \{\omega_1(t), \omega_2(t)\}$. $(\omega_1, \omega_2)$ is a* type witness *for $\mathcal{T}$ if the following properties are fulfilled.*

1. *$\omega_1(t) \neq \omega_2(t)$ and $\{\omega_1(t), \omega_2(t)\} \subseteq t \setminus \bigcup_{t' \in \mathcal{T} \setminus \{t\}} t'$ for all $t \in \mathcal{T}$,*
2. *If $w \sqsubseteq \alpha \circ w_1 \circ \cdots \circ w_m$ for $w_1, \ldots, w_m \in \bigcup_{t \in \mathcal{T}} t$, $\alpha \in \Sigma^*$, and $w \in W$, then $w \sqsubseteq \alpha \circ w_1$.*
3. *If $w \in W$ and $w' \in \bigcup_{t \in \mathcal{T}} t$ then $w'$ is not a proper suffix of $w$.*
4. *If $w, w' \in W$ then $w'$ is not a proper prefix of $w$.*

Conditions 2 through 4 in this definition are included for technical reasons that will become obvious in the results proven below. In Lange and Wiehagen's case, there was only one type in $\mathcal{T}$, namely $\Sigma^+$. As type witnesses, two distinct words of length 1 in $\Sigma^+$ were used. This agrees with all conditions in Definition 6.

**Remark:** For some applications, Definition 6 may be inappropriate. However, there are other options to define type witnesses to yield sufficient conditions for learnability. We see Definition 6 as one non-trivial example of creating sufficient conditions for the design of non-trivial efficient learning algorithms for typed pattern languages.

Since the length of a witness word no longer has to be equal to 1, it might be impossible for the learner to identify which parts of a word in the text correspond to constants in the underlying target pattern. This obstacle is avoided if type witnesses are *short* in the sense of the following definition.

**Definition 7.** *Let $\mathcal{T}$ be a set of subsets of $\Sigma^+$. Let $(\omega_1, \omega_2)$ be a type witness for $\mathcal{T}$. $(\omega_1, \omega_2)$ is a* short type witness *for $\mathcal{T}$ if, for all $t \in T$ and all $w \in t$,*

$$|\omega_1(t)| = |\omega_2(t)| \leq |w| \, .$$

The following example demonstrates how short type witnesses can be defined for Lange and Wiehagen's case of learning non-erasing pattern languages and for a text mining case in which variables can be either of type "positive integer" or of type "float" or of type "string."

*Example 8.* 1. Let $\Sigma = \{a, b\}$, $t = \Sigma^+$, $\mathcal{T} = \{t\}$. Let $\omega_1(t) = a$ and $\omega_2(t) = b$. Then $(\omega_1, \omega_2)$ is a short type witness for $\mathcal{T}$.

2. Let $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ where $\Sigma_1 = \{0, 1, \ldots, 9\}$, $\Sigma_2 = \{`.', `,', \text{SPACE}\}$, $\Sigma_3 = \{a, b, \ldots, z\}$. Let $\mathcal{T} = \{t_1, t_2, t_3\}$, where $t_1 = \{w \in \Sigma_1^+ \mid w$ does not start with $0\}$, $t_2 = \{w \circ . \circ w' \mid w, w' \in \Sigma_1^+\}$, and $t_3 = \{w \circ \sigma \circ w' \mid w, w' \in \Sigma^*,\ \sigma \in \Sigma_3,\ w$ does not contain any substring $\alpha.\sigma'$ with $\alpha \in t_1$ and $\sigma' \in \Sigma \setminus \{\text{SPACE}\}\} \setminus \{1, 2\}$. Define $(\omega_1(t_1), \omega_2(t_1)) = (1, 2)$, $(\omega_1(t_2), \omega_2(t_2)) = (3.0, 4.0)$, $(\omega_1(t_3), \omega_2(t_3)) = (a, b)$. Then $(\omega_1, \omega_2)$ is a short type witness for $\mathcal{T}$.

## 4 Polynomial-Time Learning of Typed Pattern Languages

In what follows, polynomial learning of typed pattern languages implicitly uses a representation class in which each language is represented by a pattern $p$ and a table assigning a type name to each variable occurring in $p$.

Our first theorem states that the class of all non-erasing typed pattern languages generated by terminal-free patterns is polynomially learnable from positive data, if the underlying type collection has a type witness. By terminal-free patterns we mean patterns consisting only of variables.

**Theorem 9.** *Let $\mathcal{T}$ be a finite set of decidable subsets of $\Sigma^+$ that has a type witness. Then the class of all non-erasing $\mathcal{T}$-typed pattern languages that are generated by terminal-free patterns is polynomially learnable from positive data.*

To prove this theorem, we first prove some helpful lemmas.

**Lemma 10.** *Let $\mathcal{T}$ be a set of subsets of $\Sigma^+$. Let $(\omega_1, \omega_2)$ be a type witness for $\mathcal{T}$, $W = \bigcup_{t\in\mathcal{T}}\{\omega_1(t), \omega_2(t)\}$. Let $\alpha, \alpha' \in W$ and $\beta_1, \ldots, \beta_m \in \bigcup_{t\in\mathcal{T}} t$ be such that $\alpha' \sqsubseteq \alpha \circ \beta_1 \circ \cdots \circ \beta_m$. Then $\alpha = \alpha'$. In particular, every $W$-decomposable word has a unique $W$-decomposition.*

*Proof.* Since $\alpha' \sqsubseteq \alpha \circ \beta_1 \circ \cdots \circ \beta_m$, where $\alpha, \beta_1, \ldots, \beta_m \in \bigcup_{t\in\mathcal{T}} t$, and $\alpha' \in W$, Definition 6.2 implies $\alpha' \sqsubseteq \alpha$. Definition 6.4 then yields $\alpha = \alpha'$. □

Lemma 10 helps to prove that the unique $W$-decomposition of a $W$-decomposable word $w \in \Sigma^+$ can be found in time polynomial in the length of $w$ (for a fixed collection of polynomial-time decidable types with type witnesses).

**Lemma 11.** *Let $\mathcal{T}$ be a set of decidable subsets of $\Sigma^+$. Let $(\omega_1, \omega_2)$ be a type witness for $\mathcal{T}$, $W = \bigcup_{t\in\mathcal{T}}\{\omega_1(t), \omega_2(t)\}$. There is an algorithm that, given a $(\bigcup_{t\in\mathcal{T}} t)$-decomposable word $w \in \Sigma^+$, runs in time polynomial in $|w|$, returns the $W$-decomposition of $w$ if $w$ is $W$-decomposable and returns "no" otherwise.*

*Proof.* Because of Lemma 10, it suffices to traverse $w$ in steps in the following way. In the $i$-th step ($i \geq 1$), one picks any word $w_i \in W$ such that $w_1 \circ \cdots \circ w_{i-1} \circ w_i \sqsubseteq w$. If, after some number $m$ of steps, no more such word $w_{m+1}$ is found, one returns $(w_1, \ldots, w_m)$ in case $w_1 \circ \cdots \circ w_m = w$ and returns "no" otherwise. Correctness and efficiency are evident. □

In order to study $W$-decomposable words generated by a terminal-free pattern, we introduce some more notation.

**Definition 12.** *Let $\mathcal{T}$ be a set of subsets of $\Sigma^+$. Let $(\omega_1, \omega_2)$ be a type witness for $\mathcal{T}$, $W = \bigcup_{t\in\mathcal{T}}\{\omega_1(t), \omega_2(t)\}$. For any $\mathcal{T}$-typed pattern $(p, T)$, define*

$$L_W(p, T) = \{w \in \Sigma^+ \mid \exists \theta \in \Theta_{\Sigma,T}\ [\theta(p) = w \text{ and } \theta(x) \in W \text{ for all } x \in X]\}\,.$$

**Lemma 13.** *Let $\mathcal{T}$ be a set of subsets of $\Sigma^+$. Let $(\omega_1, \omega_2)$ be a type witness for $\mathcal{T}$, $W = \bigcup_{t\in\mathcal{T}}\{\omega_1(t), \omega_2(t)\}$. Let $(p, T)$ be a terminal-free $\mathcal{T}$-typed pattern, $|p| = m$. Let $w \in L_W(p, T)$. Then the following statements are fulfilled.*

1. *$w$ has a $W$-decomposition of length $m$.*
2. *If $w' \in L(p,T) \setminus L_W(p,T)$ and $w'$ is $W$-decomposable, then the $W$-decomposition of $w'$ has length greater than $m$.*

*Proof sketch.* Statement 1 is straightforward to show: since $p$ is terminal-free, if $w \in L_W(p,T)$ then $w$ is obviously $W$-decomposable into $m$ words.

To prove Statement 2, let $p = y_1 \dots y_m$, where the $y_i$ are (not necessarily distinct) variables. Let $\theta \in \Theta_{\Sigma,T}$ be such that $w' = \theta(p) = \theta(y_1) \circ \cdots \circ \theta(y_m)$. Let $(v_1, \dots, v_n)$ be the unique $W$-decomposition of $w'$. We need to show that $n > m$. Assume $n \leq m$. The rest of the proof can be sketched as follows.

First, repeated application of Property 2 in Definition 6 yields $v_1 \circ \cdots \circ v_i \sqsubseteq \theta(y_1) \circ \cdots \circ \theta(y_i)$ for $1 \leq i \leq n$, which implies $n = m$ and $v_1 \circ \cdots \circ v_n = \theta(y_1) \circ \cdots \circ \theta(y_n)$.

Second, $v_1 \circ \cdots \circ v_i \sqsubseteq \theta(y_1) \circ \cdots \circ \theta(y_i)$ for $1 \leq i \leq n$ and $v_1 \circ \cdots \circ v_n = \theta(y_1) \circ \cdots \circ \theta(y_n)$ implies that either (i) $\theta(y_i) = v_i$ for all $i \in \{1, \dots, n\}$ or (ii) $\theta(y_i)$ is a proper suffix of $v_i$ for some $i \in \{1, \dots, n\}$. (i) would contradict the choice of $w' \notin L_W(p,T)$, (ii) would contradict Property 3 in Definition 6. Since the assumption $n \leq m$ leads to a contradiction, we have $n > m$. □

Lemma 13 is crucial for our proof of Theorem 9. It further implies the existence of normal forms for terminal-free $\mathcal{T}$-typed patterns, for any type collection $\mathcal{T}$ that has a type witness. In particular, if two terminal-free $\mathcal{T}$-typed patterns generate the same language, they are equal modulo renaming of variables.

*Proof of Theorem 9.* Let $\mathcal{T}$ be a finite set of decidable sets of words. Let $(\omega_1, \omega_2)$ be a type witness for $\mathcal{T}$ and let $W = \bigcup_{t \in \mathcal{T}} \{\omega_1(t), \omega_2(t)\}$. The required learner $\mathcal{A}$, given a set $S \subseteq \Sigma^+$, obeys the following algorithm.

**Algorithm 1.**

1. *Compute the set $C$ of all $W$-decomposable words in $S$ with the shortest $W$-decompositions. Let $m$ be the length of such a shortest decomposition.*
2. *Initialize $p = y_1 \dots y_m$. For each $i \in \{1, \dots, m\}$, let $T(y_i)$ be the type $t \in \mathcal{T}$ for which the $i$-th word in the $W$-decomposition of any $z \in C$ is contained in $\{\omega_1(t), \omega_2(t)\}$.*
3. *For each $i \in \{1, \dots, m\}$ and each $j \in \{i, \dots, m\}$, if $T(y_i) = T(y_j)$ and, for all $z \in S$, the $i$-th word equals the $j$-th word in the $W$-decomposition of $z$, then replace $y_i$ and $y_j$ in $p$ by $x_i$.*
4. *Replace all remaining $y_i$ in $P$ by $x_i$. $T(x_i) = T(y_i)$ for each $i \in \{1, \dots, m\}$.*
5. *Return $(p, T)$.*

Step 1 can be done in polynomial time according to Lemma 11. Step 2 can be executed since $\mathcal{T}$ is finite and because of Definition 6. Steps 3 and 4 can obviously be executed in polynomial time.

Assume the target pattern is $(p^*, T^*)$. Note that $L_W(p^*, T^*)$ is finite. Let $S^*$ be a subset of $L_W(p^*, T^*)$ consisting of two words $v_1(x) = \theta_1^x(p^*)$ and $v_2(x) = \theta_2^x(p^*)$ per variable $x$ occurring in $p^*$ where, for $\ell \in \{1, 2\}$,

$$\theta_\ell^x(x_i) = \begin{cases} \omega_\ell(T^*(x)), & \text{if } x_i = x, \\ \omega_1(T^*(x_i)), & \text{if } x_i \neq x. \end{cases}$$

If $S^* \subseteq S$, it is not hard to verify, using Lemma 13, that there is a terminal-free pattern $(p, T)$ with $L(p^*, T^*) = L(p, T)$, such that $\mathcal{A}$ on input $S$ returns $(p, T)$.

Finally, note that the size of the characteristic set $S^*$ defined above is polynomial in the length of $p^*$. □

Theorem 9 can be generalized to the case of all non-erasing typed pattern languages if we require the underlying type witnesses to be short.

**Theorem 14.** *Let $\mathcal{T}$ be a finite set of decidable subsets of $\Sigma^+$ that has a short type witness. Then the class of all non-erasing $\mathcal{T}$-typed pattern languages is polynomially learnable from positive data.*

*Proof sketch.* The proof is based on the easily verifiable fact that, for any target pattern $(p, T)$, the set $L_W(p, T)$ is contained in the set of shortest words in $L(p, T)$, which is obviously finite. The required learner $\mathcal{A}$, given a finite set $S \subseteq \Sigma^+$, obeys the following algorithm.

**Algorithm 2.**

1. *Compute the set $C$ of all shortest words in $S$, where $m$ is their length.*
2. *Initialize $p = y_1 \dots y_m$. For each $i \in \{1, \dots, m\}$, if there is a $\sigma \in \Sigma$ such that the $i$-th position of each word in $C$ is $\sigma$, change the $i$-th position of $p$ to $\sigma$ and mark the $i$-th position in $p$ "constant."*
3. *Remove all words from $C$ for which any of the maximal substrings corresponding to non-constant positions in $p$ are not $W$-decomposable.*
4. *Apply Algorithm 1 to the left-to-right concatenations of maximal substrings of words in $C$ corresponding to non-constant positions in $p$. Replace the non-constant parts in $p$ with the corresponding parts of the pattern obtained from Algorithm 1.*
5. *Return $(p, T)$.*

It is not hard to verify that this algorithm runs in polynomial time.

Assume the non-erasing target pattern is $(p^*, T^*)$. Let $S^*$ be the subset of all shortest words in $L(p^*, T^*)$ that is defined by analogy with the set $S^*$ used in the proof of Theorem 9. The size of $S^*$ is polynomial in $|p^*|$. If $S^* \subseteq S$, one can prove that there is a pattern $(p, T)$ with $L(p^*, T^*) = L(p, T)$, such that $\mathcal{A}$ on input $S$ returns $(p, T)$. Details are omitted due to space constraints. □

## 5 Extensions of the Presented Results

This section addresses several possible ways of strengthening the above results.

### 5.1 Learning Typed Pattern Languages over Infinitely Many Types

For some applications, it might be required to model the learning problem using an infinite set of types, i.e., an infinite set $\mathcal{T}$. Our learnability results immediately transfer to this case if the type witness is equipped with two appropriate mappings that (i) assign type witness words to a type index number and (ii) find a type index for each witness word.

**Theorem 15.** *Let $\mathcal{T}$ be a set of decidable subsets of $\Sigma^+$ that has a short type witness $(\omega_1, \omega_2)$ fulfilling the following property.*

> *There is an efficiently computable one-to-one mapping* $\text{witness} : \mathbb{N} \to (\Sigma^+)^2$ *and an efficiently computable mapping* $\text{typeIndex}\colon \bigcup_{t\in\mathcal{T}} \{\omega_1(t), \omega_2(t)\} \to \mathbb{N}$ *such that, for all $t \in \mathcal{T}$ and $w \in \{\omega_1(t), \omega_2(t)\}$,* $\text{witness}(\text{typeIndex}(w)) = (\omega_1(t), \omega_2(t))$.

*Then the class of all non-erasing $\mathcal{T}$-typed pattern languages is polynomially learnable from positive data.*

We get an analogous generalization of Theorem 9 for terminal-free patterns when dropping the shortness constraint on the type witness.

## 5.2 Consistent Learning of Typed Pattern Languages

Algorithms 1 and 2 presented above, just like Lange and Wiehagen's algorithm, ignore most of the input words and build a hypothesis based just on a special subset of the given words. A side-effect is that the hypotheses returned on any set not containing a characteristic set may be inconsistent with the given data.

One way of making our learning algorithms consistent would be to test for each input word whether it is generated by the pattern that the algorithm would normally return. If not, then $(x_1, T)$ with $T(x_1) = \Sigma^+$ could be returned instead. However, there is no known polynomial-time algorithm deciding membership for (typed) pattern languages, even if all types are polynomial-time decidable: the membership problem for non-erasing pattern languages is NP-complete [1].[3]

However, the membership problem for non-erasing pattern languages is in P, when restricted to patterns containing a bounded number of distinct variables [1]. This was generalized to typed pattern languages with finitely many polynomial-time decidable types [6]. A similar result holds for all typed pattern languages with finitely many polynomial-time decidable types when bounding the length of words from above [6]. These results imply the following theorem.

**Theorem 16.** *Let $\mathcal{T}$ be a finite set of polynomial-time decidable subsets of $\Sigma^+$ that has a short type witness $(\omega_1, \omega_2)$. Let $k \in \mathbb{N}$. Then the following holds.*

1. *The class of all non-erasing $\mathcal{T}$-typed pattern languages that are generated by typed patterns containing at most $k$ distinct variables is polynomially learnable from positive data by a consistent learning algorithm.*
2. *The class of all non-erasing $\mathcal{T}$-typed pattern languages with shortest words of length at most $k$ is polynomially learnable from positive data by a consistent learning algorithm, if the input to the learning algorithm is restricted to sets of words of length at most $c \cdot k$ for some constant $c$.*

A detailed proof is omitted due to space constraints. Again we get an analogous generalization of Theorem 9 for terminal-free patterns when dropping the shortness constraint on the type witness.

[3] The membership problem for (non-erasing) pattern languages is to decide, given a pattern $p$ and a word $w$, whether or not $w \in L(p)$.

## 6 Conclusion

We generalized the method of Lange and Wiehagen's polynomial-time algorithm for learning pattern languages to be applicable to typed pattern languages. This required the introduction of type witnesses, structurally restricting the class of typed pattern languages for which we can prove polynomial-time learnability. We established a connection to the study of characteristic sets, though in general without being able to guarantee consistent learning algorithms. Our preliminary considerations in Section 5.2 are a first step towards future work on the problem of achieving consistency without sacrificing runtime efficiency.

Type witnesses are likely to be of further use for the study of efficient algorithms for the identification of (typed or relational) patterns. In particular, specific application scenarios will allow for modeling specific sets of type witnesses; in all such cases the algorithms we proposed can be a basis for efficient solutions to machine learning and data mining problems.

## References

1. Angluin, D.: Finding patterns common to a set of strings. J. Comput. Syst. Sci. 21, 46–62 (1980)
2. Angluin, D.: Inductive inference of formal languages from positive data. Inform. Control 45, 117–135 (1980)
3. Angluin, D.: Inference of reversible languages. J. ACM 29, 741–765 (1982)
4. Blum, L., Blum, M.: Toward a mathematical theory of inductive inference. Inform. Control 28, 125–155 (1975)
5. Case, J., Kötzing, T.: Difficulties in Forcing Fairness of Polynomial Time Inductive Inference. In: Gavaldà, R., Lugosi, G., Zeugmann, T., Zilles, S. (eds.) ALT 2009. LNCS, vol. 5809, pp. 263–277. Springer, Heidelberg (2009)
6. Geilke, M., Zilles, S.: Learning Relational Patterns. In: Kivinen, J., Szepesvári, C., Ukkonen, E., Zeugmann, T. (eds.) ALT 2011. LNCS, vol. 6925, pp. 84–98. Springer, Heidelberg (2011)
7. Gold, E.M.: Language identification in the limit. Inform. Control 10, 447–474 (1967)
8. de la Higuera, C.: Characteristic sets for polynominal grammatical inference. In: ICGI, pp. 59–71 (1996)
9. Koshiba, T.: Typed Pattern Languages and Their Learnability. In: Vitányi, P.M.B. (ed.) EuroCOLT 1995. LNCS, vol. 904, pp. 367–379. Springer, Heidelberg (1995)
10. Lange, S., Wiehagen, R.: Polynomial-time inference of arbitrary pattern languages. New Generation Comput. 8, 361–370 (1991)
11. Pitt, L.: Inductive Inference, DFAs, and Computational Complexity. In: Jantke, K.P. (ed.) AII 1986. LNCS, vol. 265, pp. 18–44. Springer, Heidelberg (1987)
12. Reidenbach, D.: Discontinuities in pattern inference. Theor. Comput. Sci. 397, 166–193 (2008)
13. Shinohara, T.: Polynomial Time Inference of Extended Regular Pattern Languages. In: Goto, E., Furukawa, K., Nakajima, R., Nakata, I., Yonezawa, A. (eds.) RIMS 1982. LNCS, vol. 147, pp. 115–127. Springer, Heidelberg (1983)
14. Wright, K.: Inductive identification of pattern languages with restricted substitutions. In: COLT, pp. 111–121 (1990)

# Forbidding Sets and Normal Forms for Language Forbidding-Enforcing Systems

Daniela Genova

Department of Mathematics and Statistics
University of North Florida Jacksonville, FL 32224, USA
d.genova@unf.edu

**Abstract.** This paper investigates ways to reduce redundancy in forbidding sets for language forbidding-enforcing systems. A language forbidding set disallows combinations of subwords in a word, while permitting the presence of some parts of these combinations. Since a forbidding set is a potentially infinite set of finite sets of words, finding normal forms for forbidding sets is interesting from a combinatorics on words perspective and important for the theoretical investigation of language fe-systems, the connection between variants of fe-systems, and their applications to molecular computation. This paper shows that the minimal normal forms for forbidding sets defining classes of languages (fe-families) are also normal forms for forbidding sets defining single languages (fe-languages), but not necessarily minimal. Thus, an investigation of minimality and sufficient conditions for fe-languages are presented and it is shown that in special cases they coincide with a minimal normal form for fe-families.

**Keywords:** fe-systems, forbidden words, biomolecular computing, normal forms, formal languages.

## 1 Introduction

Forbidding-enforcing systems (fe-systems) can be viewed, in general, as boundary restrictions imposed on classes of structures that can be defined over any category of objects and morphisms [9]. Abstracting from the non-deterministic behavior of molecules in molecular reactions, A. Ehrenfeucht and G. Rozenberg introduced the forbidding and enforcing paradigm ([2,3,4,16]) as fe-systems that define classes of languages (fe-families) capable of providing means for information processing. These classes of languages were shown to be different than Chomsky's hierarchy [8]. Fe-systems have been proposed in the framework of membrane computing [1], used to model DNA self-assembly [5], and defined on graphs [6]. Detailed discussion of DNA computing models, splicing systems, membrane systems, and DNA self-assembly, can be found in [10,11,12,13,14,15,17]).

This paper investigates ways to simplify forbidding sets for a variant of fe-systems introduced in [7], in which one fe-system is used to define a single language (fe-language fe-system), as opposed to a family of languages (fe-family fe-system) as defined in [4]. Unlike a grammar or an automaton, which generates or accepts a word symbol by symbol, a language fe-system defines a language based on forbidden and enforced subwords. Characterizations of local and factorial languages by fe-systems were presented in [7] and it was shown that such systems can define the solutions to the $k$-colorability problem and model splicing rules. Using one system to define one set of words is motivated by molecular computation, where the result of a computation is a set of molecules (a set of words over a DNA alphabet), restriction enzymes require specific sequences (subwords) to perform a cut, and from DNA involution codes, where subwords are essential to the word structure.

The focus here is on theoretical properties of forbidding sets and the sets of words that they define and shies away from applications. This paper investigates how redundancy of forbidding sets can be avoided, i.e. how by shrinking certain forbidden combinations of subwords or deleting many combinations from the forbidding set, one can find an equivalent reduced set of forbidden combinations of subwords that defines the same set of consistent words (fe-language). From a combinatorics on words perspective, the combinations of forbidden words are non-strict, i.e., some parts of them are allowed, and the number of forbidden combinations may be infinite.

Following the related definitions and examples in Section 2, Section 3 investigates the similarities and differences of the normal forms for forbidding sets for both the fe-family and fe-language fe-systems models and shows that the subword free and subword incomparable minimal normal form proved in [2,16] for fe-families is not necessarily minimal for the single language forbidding set. Minimal language forbidding sets are investigated in Section 4 and Section 5 provides normal forms for strict forbidding sets.

## 2 Language Forbidding-Enforcing Systems

A finite set of symbols (*alphabet*) is denoted by $A$ and the free monoid consisting of all words over $A$ is denoted by $A^*$. A subset of $A^*$ is called a *language*. The *length* of a word $w \in A^*$ is denoted by $|w|$ and $A^m$ is the set of all words of length $m$, whereas $A^{\leqslant m}$ is the set of all words of length at most $m$. The empty word, denoted by $\lambda$ has length 0. The language $A^+$ consists of all words over $A$ with positive length.

The word $y \in A^*$ is a *subword* (*factor*) of $x \in A^*$, if there exist $s, t \in A^*$, such that $x = syt$. The set of subwords of a word $x$ is denoted by $sub(x)$ and the set of subwords of a language $L$ by $sub(L)$, where $sub(L) = \cup_{x \in L} sub(x)$.

When referring to fe-families, this paper uses the definitions and notation from [4]. For more details about properties of fe-systems defining fe-families of languages, the reader is referred to [2,3,4,6,8,16].

The normal forms in this paper relate to the fe-systems model introduced in [7], in which one forbidding-enforcing system defines a single language as opposed to a family of languages. The related definitions are recalled below. Assume that the alphabet $A$ is given.

**Definition 1.** A *forbidding set* $\mathcal{F}$ is a family of finite nonempty subsets of $A^+$; each element of a forbidding set is called a *forbidder.* A word $w$ is *consistent with a forbidder* $F$, denoted by $w\,con\,F$, if and only if, $F \not\subseteq sub\,(w)$. A word $w$ is *consistent with a forbidding set* $\mathcal{F}$, denoted by $w\,con\,\mathcal{F}$, if and only if, $w\,con\,F$ for all $F \in \mathcal{F}$. If $w$ is not consistent with $F$ (resp. $\mathcal{F}$), the notation is $w\,ncon\,F$ (resp. $w\,ncon\,\mathcal{F}$). The language $L(\mathcal{F}) = \{w \mid w\,con\,\mathcal{F}\}$. A language $L$ is a *forbidding language* or *f-language*, if there is a forbidding set $\mathcal{F}$ such that $L = L(\mathcal{F})$. Two forbidding sets $\mathcal{F}$ and $\mathcal{F}'$ are *equivalent*, denoted by $\mathcal{F} \sim \mathcal{F}'$, if and only if, $L(\mathcal{F}) = L(\mathcal{F}')$. A forbidding set $\mathcal{F}$ is called *strict*, if and only if, $|F| = 1$ for every $F \in \mathcal{F}$.

The following forbidding set was discussed in [2,4,6,8,16], where it was used to define a family of languages. The example below is from [7], where the same forbidding set was used to define a single set of words (f-language).

*Example 2.* Assume that $A = \{a, b\}$. Let $\mathcal{F} = \{\{ab, ba\}, \{aa, bb\}\}$. Then $L(\mathcal{F}) = \{a^n, b^n, ab^n, a^nb, ba^n, b^na \mid n \geq 0\}$.

Other examples of f-languages over the same alphabet $A$ include: $L(\mathcal{F}_1) = a^*$ for $\mathcal{F}_1 = \{\{b\}\}$ and if $\mathcal{F}_2 = \{\{bb\}\}$, then $L(\mathcal{F}_2)$ contains words where any two $b$'s are separated by at least one $a$. Note that $a^* \subset L(\mathcal{F}_2)$. Also note that if nothing is forbidden, then everything is allowed, i.e. $L(\mathcal{F}) = A^*$ if and only if $\mathcal{F}$ is empty.

Theorem 1 in [7] establishes a connection between the fe-family defined by a forbidding set and the fe-language defined by the same forbidding set. It states that the union of maximal languages in the fe-family gives the fe-language. The same statement holds, if we replace "maximal languages" by "languages". The following result is used in the proof of Theorem 8.

**Theorem 3.** *Let $\mathcal{F}$ be a forbidding set. Then, $L(\mathcal{F}) = \cup_{L \in \mathcal{L}(\mathcal{F})} L$.*

*Proof.* Let $\mathcal{F}$ be given. Assume $w \in L(\mathcal{F})$. Then, the language $\{w\} \in \mathcal{L}(\mathcal{F})$, otherwise there is a forbidder $F \in \mathcal{F}$, such that $F \subseteq sub\,(\{w\})$ and we have $F \subseteq sub\,(w)$, which contradicts the assumption that $w\,con\,F$. Hence, $w \in \cup_{L \in \mathcal{L}(\mathcal{F})} L$. Therefore, $L(\mathcal{F}) \subseteq \cup_{L \in \mathcal{L}(\mathcal{F})} L$. Conversely, assume $w \in \cup_{L \in \mathcal{L}(\mathcal{F})} L$. Then, there exists a language $K \in \mathcal{L}(\mathcal{F})$ such that $w \in K$. Let $F \in \mathcal{F}$. Since $K\,con\,\mathcal{F}$, it follows that $F \not\subseteq sub\,(K)$. Then, $F \not\subseteq sub\,(w)$, otherwise $F \subseteq sub\,(w)$ and $sub(w) \subseteq sub\,(K)$ imply $F \subseteq sub\,(K)$, a contradiction. Since $w\,con\,F$ for an arbitrary $F \in \mathcal{F}$, we have that $w\,con\,\mathcal{F}$, i.e., $w \in L(\mathcal{F})$. Thus, $\cup_{L \in \mathcal{L}(\mathcal{F})} L \subseteq L(\mathcal{F})$. Consequently, $L(\mathcal{F}) = \cup_{L \in \mathcal{L}(\mathcal{F})} L$. □

The other boundary condition used in the fe-language fe-systems model proposed in [7] is an enforcing set.

**Definition 4.** An *enforcing set* $\mathcal{E}$ is a family of ordered pairs called *enforcers* $(x, Y)$, such that $x \in A^*$ and $Y = \{y_1, \ldots, y_n\}$ where $y_i \in A^+$ for $i = 1, \ldots, n$, $x \in sub\,(y_i)$ and $x \neq y_i$ for every $y_i \in Y$. A word $w$ *satisfies an enforcer* $(x, Y)$ $(w\,sat\,(x, Y))$, if and only if, $w = uxv$ for some $u, v \in A^*$ implies that there exists $y_i \in Y$ and $u_1, u_2, v_1, v_2 \in A^*$ such that $y_i = u_2xv_2$ and $w = u_1u_2xv_2v_1$.

In the case that $x \notin sub(w)$, $w$ is said to satisfy the enforcer trivially. A word $w$ *satisfies an enforcing set* $\mathcal{E}$ ($w \, sat \, \mathcal{E}$), if and only if, $w$ satisfies every enforcer in that set. For an enforcing set $\mathcal{E}$ the set of all words that satisfy it is denoted by $L(\mathcal{E})$. A language $L$ is called an *e-language*, if there exists an enforcing set $\mathcal{E}$ such that $L = L(\mathcal{E})$.

Enforcers in which $x = \lambda$ are called *brute*. In this case, a word from $Y$ has to be a subword of $w$ in order for $w$ to satisfy the enforcer. Note that if $y \in Y$, then $y \, sat \, (x, Y)$. Also, $L(\mathcal{E}) = A^*$ if and only if $\mathcal{E} = \emptyset$. An enforcer $(x, Y)$ is called *strict*, if $|Y| = 1$. Consider the enforcing set $\mathcal{E} = \{(\lambda, \{a\})\} \cup \{(a^i, \{a^{i+1}\}) \mid i \geq 1\}$ over an alphabet $A$ that contains $a$. It consists of strict enforcers only, one of which is brute and requires that $a^i \in sub(w)$ for any $i \geq 1$. Since $L(\mathcal{E})$ can only contain finite words, $L(\mathcal{E}) = \emptyset$.

The idea of a forbidding-enforcing system for families of languages from [4] is preserved for a set of words (fe-language) in [7] and the definition is stated next.

**Definition 5.** A *forbidding-enforcing system* is an ordered pair $(\mathcal{F}, \mathcal{E})$, such that $\mathcal{F}$ is a forbidding set and $\mathcal{E}$ is an enforcing set. The language $L(\mathcal{F}, \mathcal{E})$ defined by this system consists of all words that are consistent with $\mathcal{F}$ and satisfy $\mathcal{E}$, i.e., $L(\mathcal{F}, \mathcal{E}) = L(\mathcal{F}) \cap L(\mathcal{E})$. A language $L$ is called an *fe-language*, if there exists an fe-system $(\mathcal{F}, \mathcal{E})$, such that $L = L(\mathcal{F}, \mathcal{E})$.

Basic properties of fe-language fe-systems are stated in [7] and are reminiscent of fe-family fe-systems properties from [4,16]. For example, Property 7 from Proposition 1 in [7] states that if $\mathcal{F}$ and $\mathcal{F}'$ are two forbidding sets and $\mathcal{E}$ and $\mathcal{E}'$ are two enforcing sets, then $L(\mathcal{F} \cup \mathcal{F}', \mathcal{E} \cup \mathcal{E}') = L(\mathcal{F}, \mathcal{E}) \cap L(\mathcal{F}', \mathcal{E}')$. This property is used in the following example from [7] to define the language in Item 3 as the intersection of the languages in Items 1 and 2.

*Example 6.* Let $A = \{a, b\}$.

1. Let $\mathcal{F} = \{\{ba\}\}$ and $\mathcal{E}_1 = \{(\lambda, \{a\})\} \cup \{(a^i, \{a^{i+1}, a^i b^i\}) \mid i \geq 1\}$. Then, $L_1 = L(\mathcal{F}, \mathcal{E}_1) = \{a^n b^m \mid n \leq m \text{ and } n, m \geq 1\}$.
2. Let $\mathcal{F} = \{\{ba\}\}$ and $\mathcal{E}_2 = \{(\lambda, \{b\})\} \cup \{(b^i, \{b^{i+1}, a^i b^i\}) \mid i \geq 1\}$. Then, $L_2 = L(\mathcal{F}, \mathcal{E}_2) = \{a^n b^m \mid n \geq m \text{ and } n, m \geq 1\}$.
3. Then, $L = L_1 \cap L_2 = \{a^n b^n \mid n \geq 1\} = L(\mathcal{F}, \mathcal{E}_1 \cup \mathcal{E}_2)$.

Since a forbidding (enforcing) set can be empty, an fe-system can be defined using only one of the boundary conditions as constraints, i.e., $L(\emptyset, \mathcal{E}) = A^* \cap L(\mathcal{E}) = L(\mathcal{E})$ and $L(\mathcal{F}, \emptyset) = L(\mathcal{F}) \cap A^* = L(\mathcal{F})$. In this sense, forbidding languages (enforcing languages) are fe-languages.

## 3 Subword-Free and Subword-Incomparable Normal Forms

A forbidding set may be be redundant and in that case it may be reduced by removing some parts of its forbidders or entire forbidders without changing the language that it defines and without changing the family of languages that it defines.

**Definition 7.** A forbidding set $\mathcal{F}$ is called *subword free* if all of its forbidders are subword free and *subword incomparable* if for any two forbidders $F_1, F_2 \in \mathcal{F}$ with $F_1 \neq F_2$, it holds that $sub(F_1) \not\subseteq sub(F_2)$ and $sub(F_2) \not\subseteq sub(F_1)$.

The subword free and subword incomparable normal forms for families of languages were introduced in [2] and discussed in detail in [16]. The authors proved that these two conditions combined (called minimal normal form) define a normal form that is indeed minimal and unique for fe-families. In this section, two questions are investigated: whether the normal forms proved for fe-families forbidding sets are also normal forms for fe-language forbidding sets and if so, whether the minimal and unique normal form for fe-families is minimal and unique for fe-languages, as well. The answer to the first question is affirmative. However, as shown in this section, the analogous normal form for fe-languages is neither minimal nor unique.

Example 5 in [2] shows that since the minimal normal form of the forbidding set $\mathcal{F} = \{\{a^i, b^i, a^i b^i\} \mid i \geq 1\}$ is $\mathcal{F}' = \{\{ab\}\}$, they both define the same family of languages and so, an infinite forbidding set in this case can be reduced to a finite one. Observe that, the same is true for the fe-language model, as well, since $\{a^i, b^i, a^i b^i\} \not\subseteq sub(w)$ for all $i \geq 1$ iff $\{a, b, ab\} \not\subseteq sub(w)$ iff $\{ab\} \not\subseteq sub(w)$. In fact, consider the following.

**Theorem 8.** *Every normal form for forbidding sets for fe-families is also a normal form for forbidding sets for fe-languages.*

*Proof.* Let $\mathcal{F}$ be a forbidding set and $\mathcal{F}'$ be a forbidding set equivalent to it in some normal form for fe-families. Then, $\mathcal{L}(\mathcal{F}) = \mathcal{L}(\mathcal{F}')$. From Theorem 3, $L(\mathcal{F}) = \cup_{L \in \mathcal{L}(\mathcal{F})} L$. By the same theorem, we have that $L(\mathcal{F}') = \cup_{L \in \mathcal{L}(\mathcal{F}')} L$. Thus, $L(\mathcal{F}) = \cup_{L \in \mathcal{L}(\mathcal{F})} L = \cup_{L \in \mathcal{L}(\mathcal{F}')} L = L(\mathcal{F}')$. Hence, $L(\mathcal{F}) = L(\mathcal{F}')$. Therefore, $\mathcal{F} \sim \mathcal{F}'$ for fe-language forbidding sets, as well. $\square$

It follows from the above theorem that the minimal normal form for fe-family forbidding sets is a normal form for fe-language forbidding sets, as well.

**Corollary 9.** *For every language forbidding set there is an equivalent subword free and subword incomparable forbidding set.*

So, given a forbidding set, we can find a fe-language equivalent forbidding set that is subword free and subword incomparable. However, the next example shows that a subword free and subword incomparable forbidding set is not necessarily fe-language minimal.

*Example 10.* Let $A = \{a, b\}$, $\mathcal{F} = \{\{aabb\}, \{bbaa\}, \{bbabaa\}, \{aa, bb, abab\}\}$, and $\mathcal{F}' = \{\{aabb\}, \{bbaa\}, \{bbabaa\}, \{aa, bb\}\}$. Clearly, $\mathcal{F}$ is subword free and subword incomparable. In the fe-families model, this forbidding set is minimal. If we remove a forbidder, the removed forbidder is a language that is not in the old family but it is in the new one, so the obtained forbidding set is not going to be equivalent to $\mathcal{F}$. If we remove a word from a forbidder, say *abab*, the remaining words from that forbidder, i.e., $\{aa, bb\}$ form a language that is consistent with $\mathcal{F}$ but not consistent with the newly obtained forbidding set. In the fe-language model $\mathcal{F}$ is not minimal, since it can be reduced to a smaller forbidding set which defines the same language, i.e., is equivalent to $\mathcal{F}'$. Observe that every word that contains both *aa* and *bb* as subwords and does not contain *abab* as a subword contains either *aabb*, or *bbaa*, or *bbabaa* as a subword. Hence, $w\, con\, \mathcal{F}$ implies $w\, con\, \mathcal{F}'$. The converse is obvious. Therefore, $\mathcal{F} \sim \mathcal{F}'$. Further more, since $\mathcal{F}'$ is not subword incomparable, i.e., $sub(\{aa, bb\}) \subseteq sub(G)$ for every $G \in \mathcal{F}'$ with $G \neq \{aa, bb\}$, $\mathcal{F}'$ can be reduced to $\mathcal{F}''$ where $\mathcal{F}'' = \{\{aa, bb\}\}$. Thus, $\mathcal{F} \sim \mathcal{F}''$.

## 4 Connecting Words and Minimal Normal Form

In [7] connecting words were used to prove that for every forbidding set there exists an equivalent enforcing set that defines the same set of words, i.e. language. In this paper, connecting words are used to investigate the relationship between general forbidding sets and strict ones and to prove some normal forms.

**Definition 11.** Given a finite set of words (a forbidder) $F$, a word $x$ such that $F \subseteq sub(x)$ is called a *connecting word of* $F$. The set of all connecting words of $F$ is called the *connect of* $F$ and denoted by $C(F)$. If $s \in C(F)$ and for no $t \in C(F)$, $t \neq s$ it holds that $t \in sub(s)$, then $s$ is a *minimal connecting word of* $F$. The set of minimal connecting words of $F$ is called the *minimal connect of* $F$ and denoted by $C_{min}(F)$.

*Remark 12.* Note that for a forbidder $F$ and a word $w \in A^*$, either $w\, con\, F$ or $w \in C(F)$. More precisely, $w \in C(F)$ if and only if $F \subseteq sub(w)$ if and only if $w\, ncon\, F$.

Two forbidders $F_1$ and $F_2$ are equivalent if and only if $w\, con\, F_1$ implies $w\, con\, F_2$ and vice versa. Hence, the following remark.

*Remark 13.* Let $\mathcal{F}$ be a forbidding set and $F_1, F_2 \in \mathcal{F}$. Then, $F_1$ and $F_2$ are equivalent if and only if $C(F_1) = C(F_2)$.

Even for a very simple forbidder, such as the one in the next example, the set of minimal connecting words and thus, the set of all connecting words for this forbidder may be infinite.

*Example 14.* Let $A = \{a, b\}$ and consider the forbidder $F = \{aa, bb\}$. Then $aabbabb \in C(F)$, but $aabbabb \notin C_{min}(F)$, since $aabb \in sub(aabbabb)$. However, $aababb \in C_{min}(F)$, since none of its proper subwords is in $C(F)$. In fact, $aa(ba)^i bb \in C_{min}(F)$ for any $i \geq 0$. Moreover, $C_{min}(F) = \{aa(ba)^i bb, bb(ab)^i aa \mid i \geq 0\}$.

The following useful result can be proved directly by the above definition.

**Lemma 15.** *Let $F$ and $F'$ be finite sets of words (forbidders) with $F' \subseteq F$. Then, $C(F) \subseteq C(F')$.*

**Lemma 16.** *Let $\mathcal{F}$ be a forbidding set and $F_1, F_2 \in \mathcal{F}$ such that $C(F_2) \subseteq C(F_1)$. Then $\mathcal{F} \sim (\mathcal{F} \backslash \{F_2\})$.*

*Proof.* Let $\mathcal{F}, F_1$, and $F_2$ be as in the hypothesis of the lemma. Obviously, $L(\mathcal{F}) \subseteq L(\mathcal{F} \backslash \{F_2\})$. Let $w \in L(\mathcal{F} \backslash \{F_2\})$. Since $w \, con \, F_1$, it follows that $w \notin C(F_2)$. Otherwise, since $C(F_2) \subseteq C(F_1)$, we have $w \in C(F_1)$, a contradiction with Remark 12 and so, $w \, con \, F_2$, in view of the same remark. Therefore, $L(\mathcal{F} \backslash \{F_2\}) \subseteq L(\mathcal{F})$. □

The above lemma is generalized below to allow removal of possibly infinitely many forbidders.

**Lemma 17.** *Let $\mathcal{F}$ and $\mathcal{F}'$ be forbidding sets with $\mathcal{F}' \subseteq \mathcal{F}$ such that for each $F \in \mathcal{F}$ there is $F' \in \mathcal{F}'$ such that $C(F) \subseteq C(F')$. Then $\mathcal{F}' \sim \mathcal{F}$.*

*Proof.* Obviously, $L(\mathcal{F}) \subseteq L(\mathcal{F}')$. Let $w \in L(\mathcal{F}')$ and let $F \in F$. Then, there exists $F' \in \mathcal{F}'$ such that $C(F) \subseteq C(F')$. By Remark 12, $w \notin C(F')$, which implies that $w \notin C(F)$. Hence, $w \, con \, F$ and the lemma follows. □

Example 10 shows that some words in a forbidder may be redundant depending not only on other words in the forbidder, but also on the other forbidders.

**Proposition 18.** *Let $\mathcal{F}$ be a forbidding set and $F \in \mathcal{F}$ be a forbidder. Let $x \in F$ be such that for every $w \in C(F \setminus \{x\})$ with $x \notin sub(w)$, there exists $G \in \mathcal{F}$, $G \neq F$ with $w \in C(G)$. Then, $\mathcal{F} \sim (\mathcal{F} \setminus \{F\}) \cup \{F'\}$, where $F' = F \setminus \{x\}$.*

*Proof.* Assume $\mathcal{F}$, $F$, and $x$ are given as in the statement and let $\mathcal{F}' = (\mathcal{F} \setminus \{F\}) \cup \{F'\}$, where $F' = F \setminus \{x\}$. Clearly, $L(\mathcal{F}') \subseteq L(\mathcal{F})$. Assume $w \in L(\mathcal{F})$ and $w \, ncon \, F'$. By Remark 12, $w \in C(F')$. Since $w \, con \, F$, it follows that $x \notin sub(w)$. Then, there exists $G \in \mathcal{F}$, $G \neq F$ such that $w \in C(G)$, which contradicts the assumption that $w \in L(\mathcal{F})$. Hence, $w \, con \, F'$ and $L(\mathcal{F}) \subseteq L(\mathcal{F}')$. Consequently, $\mathcal{F} \sim \mathcal{F}'$. □

Note that the above process is transitive. It follows from the above proposition that if $F$ can be reduced to $F' = F \setminus \{x\}$ and $F'$ can be reduced to some $F'' = F' \setminus \{y\}$, then $F$ can be reduced to $F'' = F \setminus \{y, x\}$. In fact, $F$ may be reduced by a subset $X$, if every element in $X$ satisfies the conditions of the above proposition.

**Definition 19.** A forbidding set is called *connecting free* if for every forbidder $F \in \mathcal{F}$ with $|F| \geq 2$ and every word $x \in F$ there exists $w \in C(F \backslash \{x\})$ with $x \notin sub(w)$ such that $w \, con \, G$ for every $G \in \mathcal{F}$, $G \neq F$.

The forbidding set from Example 2 is connecting free. Observe that if $F = \{ab, ba\}$, and $x = ab$, then there exists $w = ba$ such that $w \in C(F \setminus \{x\})$ and $w \, con \, \{aa, bb\}$. Similarly, for $x = ba$ we can take $w = ab$, for $x = aa$ let $w = bb$, and for $x = bb$ take $w = aa$.

**Proposition 20.** *Every connecting free forbidding set is subword free.*

*Proof.* Suppose a connecting free forbidding set $\mathcal{F}$ is not subword free. This implies that there exists a forbidder $F$ and $x, y \in F$ such that $x \in \mathit{sub}\,(y)$. Then, there is no $w \in C(F \setminus \{x\})$ for which $x \notin \mathit{sub}\,(w)$, since if $y \in \mathit{sub}\,(w)$, so is $x$. Hence, there doesn't exist $w \in C(F \setminus \{x\})$ with $x \notin \mathit{sub}\,(w)$ such that $w\,\mathit{con}\,G$ for every $G \in \mathcal{F}$, $G \neq F$. This contradicts the assumption that $\mathcal{F}$ is connecting free. Hence, the proposition follows. □

Example 10 shows that, in general, the converse of Proposition 20 does not hold, i.e. a subword free forbidding set is not necessarily connecting free. The next proposition states that for subword free and subword incomparable forbidding sets with forbidders consisting of no more than two elements, the converse of Proposition 20 holds.

**Proposition 21.** *Let $\mathcal{F}$ be a subword free and subword incomparable forbidding set with $|F| \leq 2$ for every $F \in \mathcal{F}$. Then, $\mathcal{F}$ is connecting free.*

*Proof.* Let $\mathcal{F}$ satisfy the hypothesis of the statement and let $F \in \mathcal{F}$ with $|F| = 2$. Since $\mathcal{F}$ is subword free, $F = \{x, y\}$ for some $x, y \in A^+$ such that $x \notin \mathit{sub}\,(y)$ and $y \notin \mathit{sub}\,(x)$. Then, $y \in C(F \setminus \{x\})$ is such that $y\,\mathit{con}\,G$ for every $G \in \mathcal{F}$, otherwise there exists $G \in \mathcal{F}$ such that $G \subseteq \mathit{sub}\,(y)$, which implies that $\mathit{sub}\,(G) \subseteq \mathit{sub}\,(y) \subseteq \mathit{sub}\,(F)$ and contradicts the assumption that $\mathcal{F}$ is subword incomparable. Similarly, $x \in C(F \setminus \{y\})$ is such that $x\,\mathit{con}\,G$ for every $G \in \mathcal{F}$. □

The following result states that if in addition to subword free and subword incomparable, a forbidding set is also connecting free, it is minimal in the sense that a removal of only one word from one forbidder changes the forbidding language.

**Lemma 22.** *Let $\mathcal{F}$ be subword incomparable and connecting free. Then, for every $F \in \mathcal{F}$ and every $x \in F$, $L(\mathcal{F}') \subset L(\mathcal{F})$, where $\mathcal{F}' = (\mathcal{F} \setminus \{F\}) \cup \{F'\}$ such that $F' = F \setminus \{x\}$.*

*Proof.* Let $\mathcal{F}$ be subword incomparable and connecting free. Let $F \in \mathcal{F}$ and $x \in F$. Consider $F' = F \setminus \{x\}$ and $\mathcal{F}' = (\mathcal{F} \setminus \{F\}) \cup \{F'\}$. It is obvious that $L(\mathcal{F}') \subseteq L(\mathcal{F})$. Since $\mathcal{F}$ is connecting free, there exists $w \in C(F')$ with $x \notin \mathit{sub}(w)$ such that $w\,\mathit{con}\,G$ for every $G \in \mathcal{F}$, $G \neq F$. Then, $w$ is such that $w \in L(\mathcal{F})$, but $w \notin L(\mathcal{F}')$. Hence, $L(\mathcal{F}') \subset L(\mathcal{F})$. □

The next example shows that even if a forbidding set contains minimal forbidders, it may still contain redundant forbidders.

*Example 23.* Let $A = \{a, b\}$ and consider $\mathcal{F} = \{\{ab\}, \{ba\}, \{aa, bb\}\}$ and let $F = \{aa, bb\}$. From Example 14, we have that $C_{min}(F) = \{aa(ba)^i bb, bb(ab)^i aa \mid i \geq 0\}$. Since $w\,\mathit{ncon}\,F$ implies that one of the words in $C_{min}(F)$ is a subword of $w$, and for every word in $C_{min}(F)$ either $ab$ or $ba$ is a subword of that word, it follows that $w\,\mathit{ncon}\,\{ab\}$ or $w\,\mathit{ncon}\,\{ba\}$. This implies that $w\,\mathit{con}\,\mathcal{F}$ if and only if $w\,\mathit{con}\,(\mathcal{F} \setminus \{F\})$. Hence, $F$ is redundant and can be removed without changing the forbidding language. The forbidding language $L(\mathcal{F}) = a^* \cup b^* = L(\mathcal{F} \setminus \{\{aa, bb\}\})$.

The next proposition is a generalization of Example 23.

**Proposition 24.** *Let $\mathcal{F}$ be a forbidding set and let $F \in \mathcal{F}$ be such that for every $w \in C(F)$ there exists $G \in \mathcal{F}$, $G \neq F$ such that $w \in C(G)$. Then, $\mathcal{F} \sim (\mathcal{F} \setminus \{F\})$.*

*Proof.* Let $\mathcal{F}$ be a forbidding set and let $F \in \mathcal{F}$ be such that for every $w \in C(F)$ there exists $G \in \mathcal{F}$, $G \neq F$ such that $w \in C(G)$. Obviously, $L(\mathcal{F}) \subseteq L(\mathcal{F} \setminus \{F\})$. Conversely, assume that $w \in L(\mathcal{F} \setminus \{F\})$. Suppose that $w \, ncon \, F$. Then, $w \in C(F)$ and so, there exists $G \neq F$ such that $w \in C(G)$. This contradicts the assumption that $w \, con \, G$. Therefore, $w \, con \, F$ and $L(\mathcal{F} \setminus \{F\}) \subseteq L(\mathcal{F})$. $\square$

**Definition 25.** A forbidding set $\mathcal{F}$ is called *connecting reduced* if for every $F \in \mathcal{F}$ there exists $w \in C(F)$, such that $w \notin C(G)$ for every $G \in \mathcal{F}$, $G \neq F$.

**Proposition 26.** *Every connecting reduced forbidding set is subword incomparable.*

*Proof.* Let $\mathcal{F}$ be connecting reduced. Let $F_1, F_2 \in \mathcal{F}$ with $F_1 \neq F_2$. Since $\mathcal{F}$ is connecting reduced, there exists $w \in C(F_2)$, such that $w \notin C(F_1)$, i.e., $F_1 \nsubseteq sub\,(w)$. Moreover, since $w \in C(F_2)$, by Remark 12, we have $F_2 \subseteq sub\,(w)$ and so, $sub\,(F_2) \subseteq sub\,(w)$. Thus, $F_1 \nsubseteq sub\,(F_2)$, otherwise $F_1 \subseteq sub\,(F_2) \subseteq sub\,(w)$, a contradiction. Hence, $sub\,(F_1) \nsubseteq sub\,(F_2)$. Similarly, $sub\,(F_2) \nsubseteq sub\,(F_1)$. Thus, $\mathcal{F}$ is subword incomparable. $\square$

Example 23 shows that the converse of the above proposition does not hold, since this forbidding set is subword incomparable, but not connecting reduced.

**Lemma 27.** *Let $\mathcal{F}$ be a connecting reduced forbidding set. Then, $L(\mathcal{F}) \subset L(\mathcal{F} \setminus \{F\})$ for any $F \in \mathcal{F}$.*

*Proof.* Assume that $\mathcal{F}$ is connecting reduced and let $F \in \mathcal{F}$. Clearly, $L(\mathcal{F}) \subseteq L(\mathcal{F} \setminus \{F\})$. Since $\mathcal{F}$ is connecting reduced, there exists $w \in C(F)$ such that $w \notin C(G)$ for every $G \in \mathcal{F}$, $G \neq F$. Then, $w \in L(\mathcal{F} \setminus \{F\})$, but $w \notin L(\mathcal{F})$. Therefore, $L(\mathcal{F}) \subset L(\mathcal{F} \setminus \{F\})$. $\square$

Example 23 also shows that a connecting free forbidding set is not necessarily connecting reduced.

**Definition 28.** A forbidding set $\mathcal{F}$ is *reduced* if it is both connecting free and connecting reduced.

The forbidding set from Example 2 is reduced.

The next theorem states that every reduced forbidding set is *minimal*, i.e. removal of only one forbidder or only one word in a forbidder, changes the set of words that the forbidding set defines. It follows from Lemmas 22 and 27.

**Theorem 29.** *Every reduced forbidding set is minimal.*

## 5 Normal Forms for Strict Forbidding Sets

**Proposition 30.** *Let $\mathcal{F}$ be a forbidding set. For every $F \in \mathcal{F}$ choose one connecting word $s_F \in C(F)$ and consider $\mathcal{F}' = \{\{s_F\} \mid F \in \mathcal{F}\}$. Then, $L(\mathcal{F}) \subseteq L(\mathcal{F}')$.*

*Proof.* Let $w \in L(\mathcal{F})$ and let $\{s_F\} \in \mathcal{F}'$. Then, $s_F$ is a connecting word for some $F \in \mathcal{F}$. Then $F \not\subseteq \mathit{sub}(w)$ implies that $\{s_F\} \not\subseteq \mathit{sub}(w)$. Thus, $L(\mathcal{F}) \subseteq L(\mathcal{F}')$. □

Note that the converse is not necessarily true even if $\mathcal{F}$ is reduced. For example, consider $A = \{a, b\}$, $\mathcal{F} = \{\{aa, bb\}\}$, and $\mathcal{F}' = \{\{aabb\}\}$. Then $w = bbaa$ is such that $w \in L(\mathcal{F}')$ and $w \notin L(\mathcal{F})$. However, if all minimal connecting words of all forbidders are considered as singleton forbidders, then the converse holds. Moreover, we have the following normal form.

**Theorem 31.** *For every forbidding set there exists an equivalent strict forbidding set.*

*Proof.* Let $\mathcal{F}$ be a forbidding set. For every $F \in \mathcal{F}$ construct the forbidding set $\mathcal{F}_F = \{\{s\} \mid s \in C_{min}(F)\}$ and consider $\mathcal{F}' = \cup_{F\in\mathcal{F}}\mathcal{F}_F$. We show that $\mathcal{F} \sim \mathcal{F}'$. Assume that $w \, con \, \mathcal{F}$. Note that by definition of $\mathcal{F}'$, for every $\{s\} \in \mathcal{F}'$ there exists $F \in \mathcal{F}$ such that $F \subseteq \mathit{sub}(s)$. Since $w \, con \, \mathcal{F}$, we have that $F \not\subseteq \mathit{sub}(w)$. It follows that $\{s\} \not\subseteq \mathit{sub}(w)$. Hence, $L(\mathcal{F}) \subseteq L(\mathcal{F}')$. Conversely, let $w \, con \, \mathcal{F}'$ and let $F \in \mathcal{F}$. Suppose $F \subseteq \mathit{sub}(w)$. Then, $w \in C(F)$ and there is $s \in C_{min}(F)$ such that $s \in \mathit{sub}(w)$, which contradicts the assumption that $w \, con \, \mathcal{F}'$. Hence, $F \not\subseteq \mathit{sub}(w)$ and $L(\mathcal{F}') \subseteq L(\mathcal{F})$. Consequently, $\mathcal{F} \sim \mathcal{F}'$. □

*Remark 32.* Note that the above theorem does not make general forbidding sets obsolete. Example 14 shows that even a simple forbidder $F = \{aa, bb\}$ may have an infinite number of minimal connecting words and replacing a finite number of forbidders with an infinite number of forbidders maybe undesirable.

*Remark 33.* Any strict forbidding set is connecting (subword) free. Also, connecting reduced is equivalent to subword incomparable for such a set.

**Corollary 34.** *For every forbidding set there exists an equivalent minimal strict forbidding set.*

*Proof.* Let $\mathcal{F}$ be given and construct an equivalent $\mathcal{F}'$ as in the proof of Theorem 31. Then, from Corollary 9 for $\mathcal{F}'$ there exists an equivalent connecting reduced (subword incomparable) forbidding set $\mathcal{F}''$. By Lemma 27, $\mathcal{F}''$ is minimal, i.e., $L(\mathcal{F}'') \subset L(\mathcal{F}'' \setminus \{F\})$ for any $F \in \mathcal{F}''$. □

**Lemma 35.** *Let $\mathcal{F}$ be a strict forbidding set and $\mathcal{F}_1$ and $\mathcal{F}_2$ be two minimal strict forbidding sets equivalent to $\mathcal{F}$. Then, $\mathcal{F}_1 = \mathcal{F}_2$.*

*Proof.* Let $\{u\} \in \mathcal{F}_1$ and since $\mathcal{F}_1 \sim \mathcal{F}_2$, $u \notin L(\mathcal{F}_1) = L(\mathcal{F}_2)$. It follows that there exists $\{v\} \in \mathcal{F}_2$ such that $\{v\} \subseteq \mathit{sub}(u)$. Hence, $\mathit{sub}(\{v\}) \subseteq \mathit{sub}(u)$. Similarly, since $v \notin L(\mathcal{F}_1)$, there exists $\{w\} \in \mathcal{F}_1$ such that $\{w\} \subseteq \mathit{sub}(v)$, which implies that $\mathit{sub}(\{w\}) \subseteq \mathit{sub}(v)$. Since both $\{w\}$ and $\{u\}$ are in $\mathcal{F}_1$,

$sub(\{w\}) \subseteq sub(\{u\})$, and $\mathcal{F}_1$ is subword incomparable, it follows that $\{w\} = \{u\}$. Hence, $sub(\{w\}) = sub(\{u\})$ and thus, $sub(\{v\}) = sub(\{u\})$. Since both $v$ and $u$ are subword free, we have that $\{v\} = \{u\}$ and $\{u\} \in \mathcal{F}_2$. Thus, $\mathcal{F}_1 \subseteq \mathcal{F}_2$. Similarly, $\mathcal{F}_2 \subseteq \mathcal{F}_1$. Consequently, $\mathcal{F}_1 = \mathcal{F}_2$. □

The next result shows that if a strict forbidding set is reduced, then it is both minimal and unique.

**Theorem 36.** *For every forbidding set there exists an equivalent unique minimal strict forbidding set.*

*Proof.* Let $\mathcal{F}$ be given and let $\mathcal{F}'$ be the strict forbidding set constructed as in the proof of Theorem 31 consisting of singleton forbidders of all minimal connecting words of the forbidders in $\mathcal{F}$. From Corollary 9 there exists a connecting reduced (subword incomparable) set $\hat{\mathcal{F}}$ equivalent to $\mathcal{F}'$. Then, $\hat{\mathcal{F}}$ is reduced. From Corollary 34, it is minimal and Lemma 35 establishes that it is unique. □

In [7] it was shown that a language is local if and only if it is an f-language. Since the characterization was obtained using strict forbidders, i.e. for every local language $L$ with a set of forbidden words $H = \{h_1, \ldots, h_n\}$ it holds that $L = A^* \backslash A^* H A^*$ if and only if $L = L(\mathcal{F})$, where $\mathcal{F} = \{\{h_1\}, \ldots, \{h_n\}\}$, we have the obvious corollary for local languages. The above theorem shows that the set of forbidden words for $L$ can be reduced.

**Corollary 37.** *For every local language $L$ with a set of forbidden words $H$, there exists a unique minimal set of forbidden words $H'$ such that $L = A^* \backslash A^* H' A^*$.*

## 6 Concluding Remarks

This paper presented an investigation of normal forms for forbidding sets defining fe-languages. It was shown that every normal form for forbidding sets defining fe-families is also a normal form for forbidding sets defining fe-languages. However, such normal forms are not necessarily minimal nor unique for fe-languages as they are known to be for fe-families. Thus, connecting free and connecting reduced forbidding sets were introduced and it was shown that they are minimal for fe-language forbidding sets and coincide with the subword free and subword incomparable normal form for fe-families for strict forbidding sets. Further, investigation of normal forms for enforcing sets is needed to enhance the study of language fe-systems and their applications to molecular computing. The relationship between language fe-systems and graph fe-systems [6] should be investigated further. Some similarity exists between the notion of connecting words and that of connecting graphs that may lead to common properties of subwords and subgraphs.

**Acknowledgement.** This work has been supported in part by a UNF Faculty Development Scholarship Grant.

## References

1. Cavaliere, M., Jonoska, N.: Forbidding and enforcing in membrane computing. Natural Computing 2, 215–228 (2003)
2. Ehrenfeucht, A., Hoogeboom, H.J., Rozenberg, G., van Vugt, N.: Forbidding and enforcing. In: DNA Based Computers V, vol. 54, pp. 195–206. AMS DIMACS, Providence (2000)
3. Ehrenfeucht, A., Hoogeboom, H.J., Rozenberg, G., van Vugt, N.: Sequences of languages in forbidding-enforcing families. Soft Computing 5(2), 121–125 (2001)
4. Ehrenfeucht, A., Rozenberg, G.: Forbidding-enforcing systems. Theoretical Computer Science 292, 611–638 (2003)
5. Franco, G., Jonoska, N.: Forbidding and enforcing conditions in DNA self-assembly of graphs. In: Nanotechnology: Science and Computation, Part I. Natural Computing Series, pp. 105–118 (2006)
6. Genova, D.: Forbidding and Enforcing of Formal Languages, Graphs and Partially Ordered Sets. Ph.D. thesis, University of South Florida (2007)
7. Genova, D.: Defining Languages by Forbidding-Enforcing Systems. In: Löwe, B., Normann, D., Soskov, I., Soskova, A. (eds.) CiE 2011. LNCS, vol. 6735, pp. 92–101. Springer, Heidelberg (2011)
8. Genova, D., Jonoska, N.: Topological properties of forbidding-enforcing systems. Journal of Automata, Languages and Combinatorics 11(4), 375–397 (2006)
9. Genova, D., Jonoska, N.: Defining structures through forbidding and enforcing constraints. Physica B: Condensed Matter 394(2), 306–310 (2007)
10. Head, T.: Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. Bull. Math. Biology 49(6), 737–759 (1987)
11. Head, T., Păun, G., Pixton, D.: Language theory and molecular genetics: generative mechanisms suggested by DNA recombination. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 2, pp. 295–360. Springer, Berlin (1996)
12. Jonoska, N., Sa-Ardyen, P., Seeman, N.C.: Computation by self-assembly of DNA graphs. Journal of Genetic Programming and Evolvable Machines 4(2), 123–138 (2003)
13. Păun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)
14. Păun, G., Rozenberg, G., Salomaa, A.: Computing by splicing. Theoretical Computer Science 168, 321–336 (1996)
15. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing, new computing paradigms. Springer, Heidelberg (1998)
16. van Vugt, N.: Models of Molecular Computing. Ph.D. thesis, Leiden University (2002)
17. Winfree, E., Yang, X., Seeman, N.C.: Universal computation via self-assembly of DNA: Some theory and experiments. In: Landweber, L., Baum, E. (eds.) DNA Computers II. AMS DIMACS series, vol. 44, pp. 191–198 (1998)

# Applying Tree Languages in Proof Theory

Stefan Hetzl[⋆]

Institute of Discrete Mathematics and Geometry
Vienna University of Technology
Wiedner Hauptstraße 8-10, A-1040 Vienna, Austria
hetzl@logic.at

**Abstract.** We introduce a new connection between formal language theory and proof theory. One of the most fundamental proof transformations in a class of formal proofs is shown to correspond exactly to the computation of the language of a certain class of tree grammars. Translations in both directions, from proofs to grammars and from grammars to proofs, are provided. This correspondence allows theoretical as well as practical applications.

## 1 Introduction

Proof theory developed from Hilbert's programme in the foundations of mathematics at the beginning of the 20th century. The fundamental observation of Hilbert was that even though mathematical proofs speak about infinite objects (such as real numbers, real-valued functions, vector spaces of such functions, etc.) they do so by using only a finite amount of symbols and of space. Therefore, considering proofs as mathematical objects in their own right makes them amenable to analysis by mathematical (finitary, discrete) means. Hilbert's original aim was to justify mathematical reasoning by consistency proofs which, by Gödel's second incompleteness theorem, turned out to be too ambitious. However, other kinds of analyses of proofs are possible.

One type of analysis that has received a lot of attention in recent years (see e.g. [15]) is *proof mining*: the extraction of additional mathematical information from existing proofs. Such additional information can often be thought of as concrete values for existential quantifiers. In the most simple situations it can be straightforward to read off such values, for example in case a proof of a statement $\exists x\, \varphi(x)$ starts with "Let us show $\varphi(a)$." for some concrete value $a$. In general the situation is more complicated, consider the following famous example:

**Theorem.** There are $x, y \in \mathbb{R} \setminus \mathbb{Q}$ s.t. $x^y \in \mathbb{Q}$.

*Proof.* If ${\sqrt{2}}^{\sqrt{2}} \in \mathbb{Q}$, let $x = y = \sqrt{2}$ and we are done as $\sqrt{2} \in \mathbb{R} \setminus \mathbb{Q}$. Otherwise ${\sqrt{2}}^{\sqrt{2}} \in \mathbb{R} \setminus \mathbb{Q}$, let $x = {\sqrt{2}}^{\sqrt{2}}, y = \sqrt{2}$ and observe $x^y = {\sqrt{2}}^2 = 2 \in \mathbb{Q}$. $\square$

This proof does not give us any information on whether ${\sqrt{2}}^{\sqrt{2}} \in \mathbb{Q}$ or not.

⋆ This work was supported by a Marie Curie Intra European Fellowship within the 7th European Community Framework Programme and by the projects I-603 N18 and P22028 of the Austrian Science Fund (FWF).

The values of the existential quantifiers $\exists x$ and $\exists y$ are not unique, we have to represent this proof by a *disjunction of instances of the theorem*: if $\varphi(x,y)$ abbreviates "$x \in \mathbb{R} \setminus \mathbb{Q}$ and $y \in \mathbb{R} \setminus \mathbb{Q}$ and $x^y \in \mathbb{Q}$", the above proof only shows that $\varphi(\sqrt{2},\sqrt{2}) \vee \varphi(\sqrt{2}^{\sqrt{2}},\sqrt{2})$ is provable (from suitable basic axioms). This situation is described from a logical point of view by Herbrand's theorem [9,2]. In its simplest form it states that $\exists x\, \varphi(x)$ for a quantifier-free formula $\varphi(x)$ is valid iff there are terms $t_1,\ldots,t_n$ s.t. $\bigvee_{i=1}^n \varphi(t_i)$ is a tautology. Such a disjunction is therefore also called *Herbrand-disjunction*.

It was easy to read off an Herbrand-disjunction from the above proof. The reason is that it contains the instances of its quantifiers in plain sight. In general however proofs use lemmas and values of quantifiers of the theorem depend on objects whose existence is asserted by these lemmas. The proofs of these lemmas may in turn rely on other lemmas and so on. In total, there can be very complicated dependencies between values of quantifiers that have to be unwound in order to obtain concrete values. A proof transformation that carries out this unwinding is *cut-elimination* (the root of this terminology being that the *cut*-rule formalises the use of a lemma). This transformation is, for a number of reasons, of central importance in proof theory. It essentially works by a stepwise reduction of the complexity of the cuts (lemmas), the interested reader is referred to [19].

In this paper we show that for a certain class of proofs, cut-elimination corresponds *exactly* to the computation of the language of a certain tree grammar: we give translations from proofs to grammars and from grammars to proofs s.t. this correspondence holds. The connection point is the observation that *an Herbrand-disjunction is given by a finite set of terms, hence a finite tree language.* A proof with lemmas then corresponds to a tree grammar whose language is an Herbrand-disjunction. Therefore, one can obtain concrete values for existential quantifiers by computing the language of a grammar. In Section 2 we will develop a suitable notion of tree grammar corresponding to rigid tree automata [13,14], in Section 3 we describe how to translate proofs to grammars and in Section 4 how to translate grammars to proofs. Most proofs in this paper will only be sketched, the reader interested in more details is referred to the technical report [10].

## 2 Rigid Tree Languages

A feature which is important for many applications of tree languages but not present in regular tree languages is the ability to carry out equality tests between subterms, for instance to recognise patterns of the form $f(x,x)$. This need has led to the development of several classes of tree automata providing this ability: some allow to specify local equality constraints as side conditions of transition rules by giving term positions explicitly, see [3] for a survey, while others consider global constraints specified via states. An important class of the latter kind are *tree automata with global equalities and disequalities* (TAGED) [5,6,7]. For the purposes of this paper it will turn out to be natural to work with *rigid tree automata* that have been introduced in [13], see also [14]. They are a subclass of TAGED (characterised by having minimal equality and disequality relations).

**Definition 1.** *A* tree automaton *on a signature $\Sigma$ is a tuple $\langle Q, F, \Delta\rangle$ where $Q$ is a finite set of state symbols, $F \subset Q$ is the set of final states and $\Delta$ a set of transition rules of the form: $f(q_1, \dots, q_n) \to q$ where $f \in \Sigma$ and $q, q_1, \dots, q_n \in Q$.*

*A* rigid tree automaton *on $\Sigma$ is a tuple $\langle Q, R, F, \Delta\rangle$ where $\langle Q, F, \Delta\rangle$ is a tree automaton and $R \subseteq Q$ is the set of rigid states.*

As usual a position is a list of natural numbers, $\mathrm{Pos}(t)$ is the set of positions of the term $t$, $\varepsilon$ is the empty (root) position and concatenation of positions $p_1$ and $p_2$ is written as $p_1.p_2$. The subterm of $t$ at a position $p \in \mathrm{Pos}(t)$ is denoted as $t|_p$. We write $\mathrm{Pos}(x, t)$ for the set of positions of the symbol $x$ in $t$ and we write $x \in t$ if $\mathrm{Pos}(x, t) \neq \emptyset$. A run of a tree automaton on a term $t$ is a function $r : \mathrm{Pos}(t) \to Q$ s.t. for all $f \in \Sigma$ and all $p \in \mathrm{Pos}(f, t)$: $f(r(p.1), \dots, r(p.n)) \to r(p) \in \Delta$. A run of a rigid tree automaton on $t$ is a run of the underlying tree automaton satisfying the additional *rigidity condition*: for all $p_1, p_2 \in \mathrm{Pos}(t)$: if $r(p_1) = r(p_2) \in R$ then $t|_{p_1} = t|_{p_2}$. $\mathcal{T}(\Sigma)$ denotes the set of ground terms over a signature $\Sigma$. The language of an automaton $A$ in a state $q$ is denoted as $L(A, q)$ and defined as the set of $t \in \mathcal{T}(\Sigma)$ s.t. there exists a run $r$ on $t$ with $r(\varepsilon) = q$. The language of $A$ is defined as $L(A) = \bigcup_{q\in F} L(A, q)$.

*Example 2.* Let $\Sigma = \{0/0, s/1\}$. A simple pumping argument shows that the language $L = \{f(t, t) \mid t \in \mathcal{T}(\Sigma)\}$ is not regular. On the other hand, $L$ is recognised by the rigid tree automaton $\langle Q, R, F, \Delta\rangle$ where $Q = \{q, q_r, q_f\}$, $R = \{q_r\}$, $F = \{q_f\}$ and $\Delta = \{0 \to q, 0 \to q_r, s(q) \to q, s(q) \to q_r, f(q_r, q_r) \to q_f\}$.

For the proof-theoretic purposes of this paper it is considerably more natural and technically useful to work with grammars instead of automata.

**Definition 3.** *A* regular tree grammar *is a tuple $\langle \alpha, N, \Sigma, P\rangle$ composed of an axiom $\alpha$, a set $N$ of non-terminal symbols with arity* 0 *and $\alpha \in N$, a term signature $\Sigma$ with $\Sigma \cap N = \emptyset$ and a set $P$ of production rules of the form $\beta \to t$ where $\beta \in N$ and $t \in \mathcal{T}(\Sigma \cup N)$.*

*A* rigid tree grammar *is a tuple $\langle \alpha, N, R, \Sigma, P\rangle$ where $\langle \alpha, N, \Sigma, P\rangle$ is a regular tree grammar and $R \subseteq N$ is the set of rigid non-terminal symbols.*

The derivation relation $\to_G$ of a regular tree grammar $G$ is defined for $s, t \in \mathcal{T}(\Sigma \cup N)$ as $s \to_G t$ if there is a production rule $\beta \to u$ and a position $p$ s.t. $s|_p = \beta$ and $t$ is obtained from $s$ by replacing $\beta$ at $p$ by $u$. A derivation of a term $t \in \mathcal{T}(\Sigma)$ in a regular tree grammar is a list of terms $t_1, \dots, t_n \in \mathcal{T}(\Sigma \cup N)$ s.t. $t_1 = \alpha, t_n = t$ and $t_i \to_G t_{i+1}$ for $i = 1, \dots, n - 1$. A derivation of $t$ in a rigid tree grammar is a derivation in the underlying regular tree grammar satisfying the additional *rigidity condition*: if $t_i \to_G t_{i+1}$ and $t_j \to_G t_{j+1}$ are applications of productions rules at positions $p_i, p_j$ with the same left-hand side $\beta \in R$, then $t|_{p_i} = t|_{p_j}$. The language of a tree grammar $L(G)$ is the set of $t \in \mathcal{T}(\Sigma)$ that are derivable in $G$. A production whose left-hand side is $\beta$ will be called $\beta$-production. Let $G = \langle \alpha, N, R, T, P\rangle$ be a rigid tree grammar; a rigid tree grammar $G' = \langle \alpha, N, R, T, P'\rangle$ is called *projection of* $G$ if $P' \subseteq P$ and $P'$ contains at most one $\beta$-production for every $\beta \in R$. A first basic but useful observation about rigid tree grammars is the following

**Lemma 4.** *Let $G$ be a rigid tree grammar and let $t \in L(G)$. Then there is a projection $G'$ of $G$ s.t. $t \in L(G')$.*

*Proof.* Use the rigidity condition, replace subderivations if needed.

In order to establish the connection with the existing literature we quickly sketch a proof of the equivalence of rigid tree grammars and rigid tree automata. The reader interested in the details is referred to [10].

**Definition 5.** *A grammar is called* normalised *if every production rule has the form $\gamma \to a$ or $\gamma \to f(\gamma_1, \ldots, \gamma_n)$ for $a, f \in \Sigma$ and $\gamma, \gamma_1, \ldots, \gamma_n \in N$.*

**Lemma 6.** *If $G$ is a rigid tree grammar, then there is a normalised rigid tree grammar $G^*$ s.t. $L(G) = L(G^*)$.*

*Proof.* In a first phase of normalisation, productions $\beta \to f(t_1, \ldots, t_n)$ are replaced by $\beta \to f(\beta_1, \ldots, \beta_n), \beta_1 \to t_1, \ldots, \beta_n \to t_n$. In the second phase, productions of the form $\beta \to \gamma$ are removed which – due to the rigidity condition – necessitates a different treatment depending on which of $\beta, \gamma$ are rigid.

**Theorem 7.** *A set of terms is language of a rigid tree grammar iff it is language of a rigid tree automaton.*

*Proof.* It is straightforward to translate between normalised rigid tree grammars and rigid tree automata, the result then follows from Lemma 6.

**Definition 8.** *A rigid tree grammar is called* totally rigid *if all non-terminals are rigid.*

Totally rigid tree grammars will simply be written as $\langle \alpha, R, \Sigma, P \rangle$.

**Definition 9.** *Let $G$ be a regular or rigid tree grammar with non-terminals $N$ and productions $P$. Define an order $<^1_G$ on $N$ as $\alpha <^1_G \beta$ if $\alpha \to t \in P$ and $\beta \in t$ and write $<_G$ for the transitive closure of $<^1_G$. $G$ is called* acyclic *if $<_G$ is.*

*Example 10.* The totally rigid grammar $G = \langle \alpha, R, \Sigma, P \rangle$ with $R = \{\alpha, \beta, \gamma\}$, $\Sigma = \{f/1, g/1, g/1, a/0, b/0\}$, and $P = \{\alpha \to h(\beta) | h(\gamma),\ \beta \to f(\gamma) | a,\ \gamma \to g(\beta) | b\}$ is cyclic because $\beta <_G \gamma$ and $\gamma <_G \beta$ but removing $\beta \to f(\gamma)$ or $\gamma \to g(\beta)$ (or both) from the productions yields an acyclic grammar.

Totally rigid acyclic grammars are central for this paper as they correspond to proofs. Furthermore, they allow the following description of their language in terms of substitutions. As usual, a substitution is a mapping from variables to terms which is different from the identity for only a finite number of variables. A substitution is written as $[x_1 \backslash t_1, \ldots, x_n \backslash t_n]$. Application of a substitution $\sigma$ to a term $t$ is written as $t\sigma$.

**Lemma 11.** *If $G$ is totally rigid and acyclic, then up to renaming of the non-terminals $G = \langle \alpha_0, \{\alpha_0, \ldots, \alpha_n\}, \Sigma, P \rangle$ with $L(G) = \{\alpha_0[\alpha_0 \backslash t_0] \cdots [\alpha_n \backslash t_n] \mid \alpha_i \to t_i \in P\}$.*

*Proof.* Acyclicity permits a renaming of non-terminals s.t. $\alpha_i >_P \alpha_j$ implies $i > j$. The result then follows from re-arranging the derivation and Lemma 4.

Consequently, the language $L(G)$ of a totally rigid and acyclic $G$ is finite.

## 3 From Proofs to Tree Languages

We now turn to proof theory. In the seminal article [8], which can be considered the founding work of structural proof theory, Gentzen introduced the sequent calculus and proved the cut-elimination theorem. A sequent is a pair of multisets of formulas written $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ whose intended meaning is the formula $(\bigwedge_{i=1}^{n} A_i) \to (\bigvee_{j=1}^{m} B_j)$.

**Definition 12.** *A proof in the sequent calculus is a tree that starts with sequents of the form $A \vdash A$ for an atomic formula $A$ and is built up using the following rules:*
*The logical rules:*

$$\frac{\Gamma \vdash \Delta, A \quad \Pi \vdash \Lambda, B}{\Gamma, \Pi \vdash \Delta, \Lambda, A \wedge B}\wedge_{\mathrm{r}} \qquad \frac{A, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}\wedge_{\mathrm{l}_1} \qquad \frac{B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}\wedge_{\mathrm{l}_2}$$

$$\frac{A, \Gamma \vdash \Delta \quad B, \Pi \vdash \Lambda}{A \vee B, \Gamma, \Pi \vdash \Delta, \Lambda}\vee_{\mathrm{l}} \qquad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \vee B}\vee_{\mathrm{r}_1} \qquad \frac{\Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \vee B}\vee_{\mathrm{r}_2}$$

$$\frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta}\neg_{\mathrm{l}} \qquad \frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A}\neg_{\mathrm{r}}$$

$$\frac{A[x\backslash t], \Gamma \vdash \Delta}{\forall x\, A, \Gamma \vdash \Delta}\forall_{\mathrm{l}} \quad \frac{\Gamma \vdash \Delta, A[x\backslash \alpha]}{\Gamma \vdash \Delta, \forall x\, A}\forall_{\mathrm{r}} \quad \frac{A[x\backslash \alpha], \Gamma \vdash \Delta}{\exists x\, A, \Gamma \vdash \Delta}\exists_{\mathrm{l}} \quad \frac{\Gamma \vdash \Delta, A[x\backslash t]}{\Gamma \vdash \Delta, \exists x\, A}\exists_{\mathrm{r}}$$

*where $t$ is a term, $\alpha$ is a variable and the quantifier rules are subject to the following conditions:*

1. *$t$ must not contain a bound variable,*
2. *$\alpha$ is called* eigenvariable *and must not occur in $\Gamma \cup \Delta \cup \{A\}$*

*The structural rules weakening, contraction and cut:*

$$\frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\mathrm{w_l} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\mathrm{w_r} \qquad \frac{A, A, \Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\mathrm{c_l} \qquad \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\mathrm{c_r}$$

$$\frac{\Gamma \vdash \Delta, A \quad A, \Pi \vdash \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda}\mathrm{cut}$$

The formula $A$ in an application of the cut-rule is called *cut-formula.* The sequent at the root of a proof is called *end-sequent* of that proof. This calculus is sound and complete for classical first-order logic in the sense that a formula $F$ is valid iff there is a proof whose end-sequent is $\vdash F$. We consider $A \to B$ to be an abbreviation of $\neg A \vee B$ and also allow free use of corresponding rule abbreviations $\to_{\mathrm{l}}$ and $\to_{\mathrm{r}}$ for

$$\frac{\Gamma \vdash \Delta, A \quad B, \Pi \vdash \Lambda}{A \to B, \Gamma, \Pi \vdash \Delta, \Lambda}\to_{\mathrm{l}} \qquad \text{and} \qquad \frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \to B}\to_{\mathrm{r}} \quad .$$

Furthermore, $\to$ is right-associative.

*Example 13.* Define the formulas $A_1 = P(a) \vee P(b)$, $A_2 = \forall x\,(P(x) \to Q(f(x)))$ and $A_3 = \forall x \forall y\,(P(x) \to Q(y) \to R(g(x,y)))$ and the proof $\pi =$

$$
\begin{prooftree}
\AxiomC{$(\pi_1)$}
\noLine
\UnaryInfC{$A_1 \vdash P(a), P(b)$}
\RightLabel{$\exists_\mathrm{r}$}
\UnaryInfC{$A_1 \vdash \exists x P(x), P(b)$}
\RightLabel{$\exists_\mathrm{r}$}
\UnaryInfC{$A_1 \vdash \exists x P(x), \exists x P(x)$}
\RightLabel{$\mathrm{c}_\mathrm{r}$}
\UnaryInfC{$A_1 \vdash \exists x P(x)$}
\AxiomC{$(\pi_2)$}
\noLine
\UnaryInfC{$P(\alpha), A_2 \vdash Q(f(\alpha))$}
\RightLabel{$\exists_\mathrm{r}$}
\UnaryInfC{$P(\alpha), A_2 \vdash \exists x Q(x)$}
\AxiomC{$(\pi_3)$}
\noLine
\UnaryInfC{$P(\alpha), Q(\beta), A_3 \vdash R(g(\alpha,\beta))$}
\RightLabel{$\exists_\mathrm{r}$}
\UnaryInfC{$P(\alpha), Q(\beta), A_3 \vdash \exists x R(x)$}
\RightLabel{$\exists_\mathrm{l}$}
\UnaryInfC{$P(\alpha), \exists x Q(x), A_3 \vdash \exists x R(x)$}
\RightLabel{cut}
\BinaryInfC{$P(\alpha), P(\alpha), A_2, A_3 \vdash \exists x R(x)$}
\RightLabel{$\mathrm{c}_\mathrm{l}$}
\UnaryInfC{$P(\alpha), A_2, A_3 \vdash \exists x R(x)$}
\RightLabel{$\exists_\mathrm{l}$}
\UnaryInfC{$\exists x P(x), A_2, A_3 \vdash \exists x R(x)$}
\RightLabel{cut}
\BinaryInfC{$A_1, A_2, A_3 \vdash \exists x R(x)$}
\end{prooftree}
$$

where the proofs $\pi_1, \pi_2$ and $\pi_3$ are left to the reader. This proof contains two cuts, one whose cut-formula is $\exists x\, P(x)$ and another whose cut-formula is $\exists x\, Q(x)$.

A quantifier in a formula $A$ is called *positive* if it is below an even number of negations in the syntax tree of $A$ and negative if it is under an odd number of negations. Positive universal and negative existential quantifiers are called *strong*, the others are called *weak*. It is straightforward to show that a strong quantifier is introduced by $\forall_\mathrm{r}$ or $\exists_\mathrm{l}$ and a weak quantifier by $\forall_\mathrm{l}$ or $\exists_\mathrm{r}$. A proof is called *regular* if different strong quantifier inferences have different eigenvariables. From now on we will assume – as a convention on variable-naming – that all proofs are regular.

In the above definition some formulas are mentioned explicitly like $A \wedge B$, $A$ and $B$ in the case of $\wedge_\mathrm{r}$. The formula $A \wedge B$ below the rule is called *main formula* and the formulas $A$ and $B$ above it are called *auxiliary formulas* of the rule. Analogous definitions apply to all other rules. One then defines an *ancestor* relation on the formula occurrences in a proof as follows: an auxiliary formula occurrence is ancestor of a main formula occurrence and furthermore the occurrence of a formula in the context $\Gamma, \Delta, \Pi, \Lambda$ above an inference is ancestor of the corresponding occurrence below the inference. For illustration of the ancestor relation see Definitions 14, 17 and Examples 15, 18.

From now on and for the rest of this paper, $T$ will denote a universal theory, i.e. a set of formulas of the form $\forall x_1 \cdots \forall x_n B$ with $B$ quantifier-free. It is a standard result of mathematical logic that every theory has a conservative universal extension which is obtained by Skolemisation, see e.g. [22]. Concentrating on universal theories hence does not significantly restrict (though simplifies technically) the results of this paper.

**Definition 14.** *Let $\pi$ be a proof of $T \vdash \exists x\, A$ with $A$ quantifier-free and $\psi$ a subproof of $\pi$. The* Herbrand-set $\mathrm{H}(\psi, \pi)$ *of $\psi$ w.r.t. $\pi$ is defined as follows. If $\psi$ is an axiom, then* $\mathrm{H}(\psi, \pi) = \emptyset$. *If $\psi$ is of the form*

$$
\begin{prooftree}
\AxiomC{$(\psi')$}
\noLine
\UnaryInfC{$\Pi \vdash \Lambda, A[x \backslash t]$}
\RightLabel{$\exists_\mathrm{r}$}
\UnaryInfC{$\Pi \vdash \Lambda, \exists x\, A$}
\end{prooftree}
$$

*where the main formula $\exists x\, A$ is ancestor of the formula $\exists x\, A$ in the end-sequent, then* $\mathrm{H}(\psi,\pi) = \mathrm{H}(\psi',\pi) \cup \{A[x\backslash t]\}$*. If $\psi$ ends with any other unary inference and $\psi'$ is its immediate subproof then* $\mathrm{H}(\psi,\pi) = \mathrm{H}(\psi',\pi)$*. If $\psi$ ends with a binary inference and $\psi_1, \psi_2$ are its immediate subproofs, then* $\mathrm{H}(\psi,\pi) = \mathrm{H}(\psi_1,\pi) \cup \mathrm{H}(\psi_2,\pi)$*. We write* $\mathrm{H}(\pi)$ *for* $\mathrm{H}(\pi,\pi)$*.*

*Example 15.* The proof $\pi$ of Example 13 has $\mathrm{H}(\pi) = \{R(g(\alpha,\beta))\}$.

*Example 16.* A (suitable) formalisation of the proof discussed in the introduction has the Herbrand-set $\{\varphi(\sqrt{2},\sqrt{2}), \varphi(\sqrt{2}^{\sqrt{2}},\sqrt{2})\}$.

**Definition 17.** *Let $\pi$ be a proof and $Q$ be a quantifier occurrence in $\pi$. Define a set of terms* $\mathrm{t}(Q)$ *associated with $Q$ as follows: if $Q$ occurs in the main formula of a weakening, then* $\mathrm{t}(Q) = \emptyset$*. If $Q$ is introduced by a quantifier inference from a term $t$ or a variable $x$, then* $\mathrm{t}(Q) = \{t\}$ *or* $\mathrm{t}(Q) = \{x\}$ *respectively. If $Q$ occurs in the main formula of a contraction and $Q_1, Q_2$ are the two corresponding quantifiers in the auxiliary formulas of the contraction, then* $\mathrm{t}(Q) = \mathrm{t}(Q_1) \cup \mathrm{t}(Q_2)$*. In all other cases $Q$ has exactly one immediate ancestor $Q'$ and* $\mathrm{t}(Q) = \mathrm{t}(Q')$*.*

*Let $\pi$ be a proof and $c$ be a cut in $\pi$. Write* $\mathrm{Q}(c)$ *for the set of pairs $(Q,Q')$ of quantifier occurrences where $Q$ is a strong occurrence in one occurrence of the cut-formula of $c$ and $Q'$ the corresponding weak occurrence in the other occurrence of the cut-formula. Define the set of base substitutions of $c$ as* $\mathrm{B}(c) = \bigcup_{(Q,Q')\in \mathrm{Q}(c)} \{[x\backslash t] \mid x \in \mathrm{t}(Q), t \in \mathrm{t}(Q')\}$*. For $c_1,\ldots,c_n$ being the cuts in $\pi$ define the base substitutions of $\pi$ as* $\mathrm{B}(\pi) = \bigcup_{i=1}^{n} \mathrm{B}(c_i)$*.*

*Example 18.* The proof $\pi$ of Example 13 has $\mathrm{B}(\pi) = \{[\alpha\backslash a], [\alpha\backslash b], [\beta\backslash f(\alpha)]\}$.

**Definition 19.** *For a proof $\pi$ define the totally rigid tree grammar* $\mathrm{G}(\pi) = \langle \varphi, N, \Sigma, P\rangle$ *by* $N = \{\varphi\} \cup \mathrm{EV}(\pi)$*,* $\Sigma = \Sigma(\pi) \cup \{\wedge,\vee,\neg\}$*, and* $P = \{\varphi \to F \mid F \in \mathrm{H}(\pi)\} \cup \{\alpha \to t \mid [\alpha\backslash t] \in \mathrm{B}(\pi)\}$*, where* $\mathrm{EV}(\pi)$ *is the set of eigenvariables of the proof $\pi$ and $\Sigma(\pi)$ is its first-order signature.*

*Example 20.* The proof $\pi$ of Example 13 has $\mathrm{G}(\pi) = \langle \varphi, N, \Sigma, P\rangle$ where $N = \{\varphi, \alpha, \beta\}$, $\Sigma = \{a, b, f, g, P, Q, R, \wedge, \vee, \neg\}$ and $P = \{\varphi \to R(g(\alpha,\beta)), \beta \to f(\alpha), \alpha \to a, \alpha \to b\}$ hence $L(\mathrm{G}(\pi)) = \{R(g(a,f(a))), R(g(b,f(b)))\}$. The reader is invited to verify that $A_1, A_2, A_3 \vdash L(\mathrm{G}(\pi))$ is provable for $A_1, A_2, A_3$ as in Example 13.

**Definition 21.** *A proof $\pi$ is called* simple *if every cut-formula in $\pi$ contains at most one quantifier.*

We will now restrict our attention to simple proofs. The main result of this paper is that cut-elimination in the class of simple proofs corresponds exactly (in a sense made precise below) to the computation of the language of a totally rigid acyclic tree grammar. While this restriction on proofs substantially decreases the scope of the present analysis, simple proofs are still of considerable interest: they do contain quantified cuts and hence allow the formalisation of some mathematical lemmas and their cut-elimination is of exponential complexity.

The size of a grammar $G$, written as $|G|$, is the number of production rules. The size of a proof $\pi$, written as $|\pi|$, is the number of inferences.

**Theorem 22.** *If $\pi$ is a simple proof of $T \vdash \exists x\, A$ with $A$ quantifier-free, then there is a totally rigid acyclic grammar $G$ with $|G| \leq |\pi|$ and $L(G) \subseteq L(\mathrm{G}(\pi))$ s.t. $T \vdash L(G)$ is provable.*

*Proof.* Only a sketch of the proof is described here, the reader interested in the details is referred to the technical report [10]. We have $|\mathrm{G}(\pi)| \leq |\pi|^2$, the quadratic size being due to quantifiers in cut-formulas which are introduced from a linear number of terms (on their weak side) and a linear number of eigenvariables (on their strong side).

Let $\beta, \gamma$ be two non-terminals of $\mathrm{G}(\pi)$ and write $\beta \sim \gamma$ if there is a strong quantifier occurrence $Q$ in a cut-formula with $\beta, \gamma \in \mathrm{t}(Q)$. The equivalence relation $\sim$ defines a partition of the non-terminals of $\mathrm{G}(\pi)$ into $n$ classes where $n$ is the number of cuts in $\pi$ that contain a quantifier. Define the totally rigid grammar $G$ from $\mathrm{G}(\pi)$ by identifying all non-terminals of the same $\sim$-class. Then $|G| \leq |\pi|$ and $L(G) \subseteq L(\mathrm{G}(\pi))$.

Furthermore, let $d$ be a new ("dummy") constant and, for a given proof $\psi$, write $\mathrm{G}_{\mathrm{nd}}(\psi)$ for $\mathrm{G}(\psi)$ from which all productions of the form $\beta \to d$ for any non-terminal $\beta$ have been removed. By proof-theoretic transformations – in particular the prenexification of [1], see also [4, Theorem VII.4.7], applied to cut-formulas – we obtain a proof $\pi'$ all of whose cuts are of the form $\exists x\, B$ for $B$ quantifier-free and which satisfies $L(\mathrm{G}_{\mathrm{nd}}(\pi')) = L(G)$. The role of the dummy constant is to mark artefacts introduced by prenexification into the grammar as such.

The central part of the proof then consists in applying a suitable procedure for cut-elimination to $\pi'$ and to show that this process is computing $L(\mathrm{G}_{\mathrm{nd}}(\pi'))$ step-by-step from $\mathrm{G}(\pi')$ in the proof: the systematic unfolding of the proof induced by cut-elimination essentially transforms a grammar into its language, the occurrences of the dummy constant are deleted by this process due to their positions in the proof. Finally we obtain a cut-free proof $\pi^*$ with $\mathrm{H}(\pi^*) = L(\mathrm{G}_{\mathrm{nd}}(\pi'))$. This allows to conclude that $T \vdash L(\mathrm{G}_{\mathrm{nd}}(\pi'))$, i.e. $T \vdash L(G)$, is provable.

**Corollary 23.** *If $\pi$ is a simple proof of $T \vdash \exists x\, A$ with $A$ quantifier-free, then $T \vdash L(\mathrm{G}(\pi))$ is provable.*

*Proof.* Append weakenings to the proof of $T \vdash L(G)$ obtained from Theorem 22.

*Example 24.* Applying Corollary 23 to the proof $\pi$ of Example 13 shows that

$$A_1, A_2, A_3 \vdash R(g(a, f(a))), R(g(b, f(b)))$$

is provable. The reader is invited to verify that a standard algorithm for cut-elimination (see e.g. [19]) gives the same result.

Note that Theorem 22 together with Lemma 11 provides an exponential upper bound on the complexity of cut-elimination in simple proofs, more precisely: for every simple proof $\pi$ of $T \vdash \exists x\, A$ with $A$ quantifier-free there are $t_1, \ldots, t_k$ with $k \leq |\pi|^{|\pi|}$ s.t. $T \vdash A[x \backslash t_1], \ldots, A[x \backslash t_k]$. On the other hand, cut-elimination in general is non-elementary [23,17,18] which shows that simplicity is a necessary assumption for Theorem 22.

## 4 From Tree Languages to Proofs

Already the results in [11] show that a simple proof induces an acyclic regular(!) tree grammar whose finite language is an Herbrand-disjunction. So what have we gained from strengthening this result by adding total rigidity? On the one hand, we have gained an exponent: in contrast to the bound $n^n$ obtained in the totally rigid case, there are acyclic regular tree grammars $G_n$ with $2n$ productions and $|L(G_n)| = n^{n^n}$ ($G_n$ builds a tree with depth $n$, branching-degree $n$ and $n$ choices at each leaf).

However there is another – more fundamental – motivation for this result: in this section we show that the compression power of simple proofs corresponds *exactly* to that of totally rigid acyclic grammars. Given such a grammar $G$ we will obtain a simple proof $\pi$ that induces $G$ and whose cut-elimination essentially computes $L(G)$ from $G$. As $L(\mathrm{G}(\pi))$ is a set of formulas but $L(G)$ is a set of terms (which do not necessarily represent formulas), we cannot expect to obtain $\mathrm{G}(\pi) = G$. The closest possible connection is to wrap up the term language of $G$ in some new unary predicate symbol $R_1$. Therefore the proofs constructed in the theorem below have $T \vdash \exists x\, R_1(x)$ as end-sequent. For proofs $\pi$ and $\pi'$ we write $\pi \to \pi'$ if $\pi'$ can be obtained from $\pi$ by applying the standard reduction rules of cut-elimination as in [19].

**Theorem 25.** *For every totally rigid acyclic tree grammar $G = \langle \alpha_1, R, \Sigma, P\rangle$ there is a simple proof $\pi$ with $\mathrm{G}(\pi) = \langle \alpha_0, R \cup \{\alpha_0\}, \Sigma, P \cup \{\alpha_0 \to R_1(\alpha_1)\}\rangle$ and a cut-free proof $\pi'$ with $\pi \to \pi'$ and $\mathrm{H}(\pi') = L(\mathrm{G}(\pi))$.*

*Proof.* By Lemma 11 we can assume that $G = \langle \alpha_1, \{\alpha_1, \dots, \alpha_n\}, \Sigma, P\rangle$ s.t. $\alpha_i$ depends only on $\alpha_j$ with $j > i$. The proof $\pi$ is defined in the language $\Sigma \cup \{R_i \mid 1 \le i \le n\}$ where the $R_i$ are unary predicate symbols with intended interpretation "being reachable from the non-terminal $\alpha_i$". For each rule $\alpha_i \to t$ define the formula

$$\varphi_{\alpha_i \to t} \;=\; \forall x_{i+1} \cdots \forall x_n \;(\; R_{i+1}(x_{i+1}) \to \cdots R_n(x_n) \to R_i(t[\alpha_j \backslash x_j]_{j=i+1}^n) \;).$$

For each non-terminal $\alpha_i$ with rules $\alpha_i \to t_1, \dots, \alpha_i \to t_m$ define the formula

$$\varphi_i \;=\; \bigvee_{j=1}^{m} \varphi_{\alpha_i \to t_j}$$

and the proof $\psi_i =$

$$\dfrac{\cdots \qquad \dfrac{\dfrac{R_i(t_j) \vdash R_i(t_j)}{R_i(t_j) \vdash \exists x\, R_i(x)}\,\exists_{\mathrm{r}}}{\varphi_{\alpha_i \to t_j}, R_{i+1}(\alpha_{i+1}), \dots, R_n(\alpha_n) \vdash \exists x\, R_i(x)}\,\forall_1^*, \to_1^* \qquad \cdots}{\varphi_i, R_{i+1}(\alpha_{i+1}), \dots, R_n(\alpha_n) \vdash \exists x\, R_i(x)}\,\mathrm{c}^*, \vee_1^* \;.$$

Now define proofs $\pi_i : \varphi_1, \dots, \varphi_i, R_{i+1}(\alpha_{i+1}), \dots, R_n(\alpha_n) \vdash \exists x\, R_1(x)$ for $i \in \{0, \dots, n\}$ and $\pi_i' : \varphi_1, \dots, \varphi_i, \exists x\, R_{i+1}(x), R_{i+2}(\alpha_{i+2}), \dots, R_n(\alpha_n) \vdash \exists x\, R_1(x)$

for $i \in \{0, \ldots, n-1\}$ by

$$\pi_i' \;=\; \cfrac{\begin{array}{c}(\pi_i)\\ \varphi_1, \ldots, \varphi_i, R_{i+1}(\alpha_{i+1}), \ldots, R_n(\alpha_n) \vdash \exists x\, R_1(x)\end{array}}{\varphi_1, \ldots, \varphi_i, \exists x\, R_{i+1}(x), R_{i+2}(\alpha_{i+2}), \ldots, R_n(\alpha_n) \vdash \exists x\, R_1(x)}\ \exists_{\mathrm{l}}$$

and

$$\pi_0 \;=\; \cfrac{R_1(\alpha_1) \vdash R_1(\alpha_1)}{R_1(\alpha_1), \ldots, R_n(\alpha_n) \vdash \exists x\, R_1(x)}\ \mathrm{w}_{\mathrm{l}}^*, \exists_{\mathrm{r}}$$

and

$$\pi_{i+1} \;=\; \cfrac{\begin{array}{c}(\psi_{i+1})\\ \varphi_{i+1}, R_{i+2}(\alpha_{i+2}), \ldots, R_n(\alpha_n) \vdash \exists x\, R_{i+1}(x)\end{array} \qquad (\pi_i')}{\varphi_1, \ldots, \varphi_{i+1}, R_{i+2}(\alpha_{i+2}), \ldots, R_n(\alpha_n) \vdash \exists x\, R_1(x)}\ \mathrm{c}_{\mathrm{l}}^*, \mathrm{cut} \quad .$$

Then it is straightforward to verify that $\pi = \pi_n : \varphi_1, \ldots, \varphi_n \vdash \exists x\, R_1(x)$ has the desired grammar. In order to obtain $\pi'$, reduce the cuts in a bottom-up order which for each production rule of an $\alpha_{i+1}$ will create a new copy of $\pi_i$ hence computing the language $L(\mathrm{G}(\pi))$ by expansion from right to left (in the representation of Lemma 11).

## 5 Applications

The above results pave the way for several applications of formal language theory in proof theory to be further explored in future work. First of all, for carrying out concrete analyses of simple proofs one can use rigid tree grammars instead of the more cumbersome cut-elimination to compute values for existential quantifiers. Secondly, standard problems of formal language theory such as membership, intersection, etc. assume a proof-theoretic meaning by allowing to answer whether a given value is obtained from a given proof, what values can be obtained from both of two given proofs, etc.

Furthermore, these results show that the length of a proof with cut (which is notoriously difficult to control) is intimately related to measures such as automatic complexity [21] and automaticity [20], more precisely:

**Corollary 26.** *Let $\exists x\, A$ be a formula and $k \in \mathbb{N}$ s.t. $T \vdash A[x\backslash t_1], \ldots, A[x\backslash t_n]$ implies that every totally rigid acyclic grammar $G$ with $L(G) = \{t_1, \ldots, t_n\}$ has $|G| \geq k$, then every simple proof $\pi$ of $T \vdash \exists x\, A$ has $|\pi| \geq k$.*

*Proof.* Suppose there was a simple proof $\pi_0$ of $T \vdash \exists x\, A$ with $|\pi_0| < k$, then by Theorem 22 there would be a totally rigid acyclic tree grammar $G_0$ with $|G_0| \leq |\pi_0| < k$ s.t. $T \vdash L(G_0)$ would be provable, contradiction.

Via this connection, a lower bound on grammars thus translates to a lower bound on proofs with cut.

Another intriguing perspective is to exploit these results computationally by abbreviating a cut-free proof through the introduction of cuts which are obtained from first computing a small grammar: the cut-free proof of $T \vdash \exists x\, A$

is represented by its Herbrand-disjunction $A[x\backslash t_1], \ldots, A[x\backslash t_n]$ which in turn is represented by the trivial grammar whose axiom is $x$ and whose productions are $x \to t_1, \ldots, x \to t_n$. An analysis of the structure of the $t_i$ can lead to a smaller grammar representing the same language. It then only remains to check whether the grammar can be realised by cut-formulas of a simple proof (which for the case of a single cut is always the case). A first algorithm based on this approach for the case of a single cut can be found in [12].

## 6 Conclusion

We have shown that cut-elimination in proofs where each cut contains at most one quantifier corresponds exactly to the computation of the language of a totally rigid acyclic tree grammar. This work constitutes a proof-of-concept result for a new connection between proof theory and formal language theory arising from exact characterisations of classes of proofs by classes of grammars. In principle, such a result is conceivable for any proof system that possesses an Herbrand-like theorem, i.e. even full higher-order logic as in [16]. The challenge consists in finding an appropriate type of grammars.

**Acknowledgements.** The author would like to thank the anonymous referees for helpful comments and suggestions that led to significant improvements of this paper.

## References

1. Baaz, M., Leitsch, A.: Cut Normal Forms and Proof Complexity. Annals of Pure and Applied Logic 97, 127–177 (1999)
2. Buss, S.R.: On Herbrand's Theorem. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 195–209. Springer, Heidelberg (1995)
3. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata: Techniques and Applications (2007), http://www.grappa.univ-lille3.fr/tata (release October 12, 2007)
4. Cook, S., Nguyen, P.: Logical Foundations of Proof Complexity. Perspectives in Logic. Cambridge University Press (2010)
5. Filiot, E., Talbot, J.-M., Tison, S.: Satisfiability of a Spatial Logic with Tree Variables. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 130–145. Springer, Heidelberg (2007)
6. Filiot, E., Talbot, J.-M., Tison, S.: Tree Automata with Global Constraints. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 314–326. Springer, Heidelberg (2008)
7. Filiot, E., Talbot, J.M., Tison, S.: Tree Automata With Global Constraints. International Journal of Foundations of Computer Science 21(4), 571–596 (2010)
8. Gentzen, G.: Untersuchungen über das logische Schließen. Mathematische Zeitschrift 39, 176–210, 405–431 (1934-1935)
9. Herbrand, J.: Recherches sur la théorie de la démonstration. Ph.D. thesis, Université de Paris (1930)

10. Hetzl, S.: Proofs as Tree Languages (preprint), http://hal.archives-ouvertes.fr/hal-00613713/
11. Hetzl, S.: On the form of witness terms. Archive for Mathematical Logic 49(5), 529–554 (2010)
12. Hetzl, S., Leitsch, A., Weller, D.: Towards Algorithmic Cut-Introduction (submitted)
13. Jacquemard, F., Klay, F., Vacher, C.: Rigid Tree Automata. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 446–457. Springer, Heidelberg (2009)
14. Jacquemard, F., Klay, F., Vacher, C.: Rigid tree automata and applications. Information and Computation 209, 486–512 (2011)
15. Kohlenbach, U.: Applied Proof Theory: Proof Interpretations and their Use in Mathematics. Springer, Heidelberg (2008)
16. Miller, D.: A Compact Representation of Proofs. Studia Logica 46(4), 347–370 (1987)
17. Orevkov, V.P.: Lower bounds for increasing complexity of derivations after cut elimination. Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta 88, 137–161 (1979)
18. Pudlák, P.: The Lengths of Proofs. In: Buss, S. (ed.) Handbook of Proof Theory, pp. 547–637. Elsevier (1998)
19. Schwichtenberg, H., Troelstra, A.S.: Basic Proof Theory. Cambridge University Press (1996)
20. Shallit, J., Breitbart, Y.: Automaticity I: Properties of a Measure of Descriptional Complexity. Journal of Computer and System Sciences 53, 10–25 (1996)
21. Shallit, J., Wang, M.W.: Automatic complexity of strings. Journal of Automata, Languages and Combinatorics 6(4), 537–554 (2001)
22. Shoenfield, J.R.: Mathematical Logic, 2nd edn. AK Peters (2001)
23. Statman, R.: Lower bounds on Herbrand's theorem. Proceedings of the American Mathematical Society 75, 104–107 (1979)

# The Membership Problem for Regular Expressions with Unordered Concatenation and Numerical Constraints

Dag Hovland

Department of Informatics, University of Oslo, Norway
hovland@ifi.uio.no

**Abstract.** We study the membership problem for regular expressions extended with operators for *unordered concatenation* and *numerical constraints*. The unordered concatenation of a set of regular expressions denotes all sequences consisting of exactly one word denoted by each of the expressions. Numerical constraints are an extension of regular expressions used in many applications, e.g. text search (e.g., UNIX grep), document formats (e.g. XML Schema). Regular expressions with unordered concatenation and numerical constraints denote the same languages as the classical regular expressions, but, in certain important cases, exponentially more succinct. We show that the membership problem for regular expressions with unordered concatenation (without numerical constraints) is already NP-hard. We show a polynomial-time algorithm for the membership problem for regular expressions with numerical constraints and unordered concatenation, when restricted to a subclass called *strongly 1-unambiguous*.

**Keywords:** Regular Expressions, Automata, Numerical Constraints, Unordered Concatenation, Interleaving, XML, SGML.

## 1 Introduction

In the ISO standard for the Standard Generalized Markup Language (SGML) [16], the precursor of XML, the operator "&" is used for what in this paper is called *unordered concatenation*, that is, the languages are concatenated, but in any order. For example, $\&(ya, basta)$ denotes $\{yabasta, bastaya\}$. In SGML "&" is infix, but because it is not associative, we find it more convenient to write it prefix. Brüggemann-Klein [4,2] investigates unambiguity of regular expressions extended with such an unordered concatenation operator.

Unordered concatenation is superficially similar to *interleaving*, an extension to regular expressions studied by e.g. Mayer & Stockmeyer in [21]. Interleaving is used to model process-theoretic parallel composition. There is no obvious way to translate the algorithms and the complexity results for interleaving to unordered concatenation.

Numerical constraints allow expressing that a subexpression must be matched a number of times specified by a lower and an upper limit. For example, $(a+b)^{2..3}$

denotes the words of length 2 or 3 consisting only of $a$'s and $b$'s. Numerical constraints are also used in XML Schema, and in addition in applications for text search, e.g. `GNU grep`. The extension has been studied by, among others, Gelade et al.[8], Kilpeläinen & Tuhkanen [18], Hovland [14], and Ghelli et al. [11].

This paper has a theoretical and a more practical motivation. The theoretical motivation is curiosity about the properties of unordered concatenation, especially when used in combination with numerical constraints. Unordered concatenation is intuitive and seems useful for searches and definitions in natural language text. *Membership* is shown to be tractable for the *strongly 1-unambiguous* regular expressions with unordered concatenation *and* numerical constraints. It is interesting to note that for the regular expressions with numerical constraints, the largest known subset where membership can be decided in time linear in the size of the word *and* polynomial in the size of the expression, is the strongly 1-unambiguous subset.

In this paper we will study the regular expressions with unordered concatenation and numerical constraints, and the membership problem for these expressions. In the next section we give a definition of these expressions and their languages, and in Sect. 3 we show that the membership problem is NP-complete already without numerical constraints. The algorithm for membership is based on construction of finite automata with counters, where positions in the term trees of the regular expressions play a central role. Section 4 is therefore devoted to a description of term trees and positions in these trees, while in Sect. 5 we define the finite automata with counters. In Sect. 6 we define strong 1-unambiguity, and state the main theorem. The last section presents some related work and a conclusion.

## 2 Regular Expressions with Unordered Concatenation and Numerical Constraints

Fix an *alphabet* $\Sigma$ of *letters*. Assume $a$, $b$, and $c$ are members of $\Sigma$. $l, l_1, l_2, \ldots$ are used as variables for members of $\Sigma$. Let $\mathbb{N} = \{1, 2, \ldots\}$, $\mathbb{N}_1 = \{2, 3, 4, \ldots\} \cup \{\infty\}$, and $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$.

**Definition 1.** *Given an alphabet $\Sigma$, $\mathcal{R}_\Sigma$ is the set of* regular expressions with unordered concatenation and numerical constraints *over $\Sigma$, defined by the following grammar:*

$$\mathcal{R}_\Sigma ::= \mathcal{R}_\Sigma + \mathcal{R}_\Sigma \mid \mathcal{R}_\Sigma \cdot \mathcal{R}_\Sigma \mid \mathcal{R}_\Sigma^{\mathbb{N}..\mathbb{N}_1} \mid \&(\mathcal{R}_\Sigma, \ldots, \mathcal{R}_\Sigma) \mid \Sigma \mid \epsilon$$

*We only allow $r^{n..u}$ for $n \leq u$. Numerical constraints have the highest precedence, followed by concatenation, choice, and unordered concatenation, which has the least precedence. Parentheses are used, when necessary, to group subexpressions. We use $r, r_1, r_2, \ldots$ as variables for regular expressions. The sign for concatenation, $\cdot$, will often be omitted. A regular expression denoting the empty language is not included, as this is irrelevant to the results in this paper.*

*We use $r^{n..}$ as shorthand for $r^{n..\infty}$, $r^{0..n}$ as shorthand for $r^{1..n} + \epsilon$, $r^+$ as shorthand for $r^{1..\infty}$, $r^*$ as shorthand for $r^{0..}$, and $r^n$ for $r^{n..n}$. We denote the set of letters from $\Sigma$ occurring in $r$ by* $\mathsf{sym}(r)$*.*

The reason the unordered concatenation operator is not binary infix, is that, as we will see below, it is not associative. The *star-free regular expressions with unordered concatenation* are the subset of $\mathcal{R}_\Sigma$ with no numerical constraints, that is, no subexpressions of the form $r^{n..u}$.

We lift concatenation of words to sets of words, such that if $L_1, L_2 \subseteq \Sigma^*$, then $L_1 \cdot L_2 = \{w_1 \cdot w_2 \,|\, w_1 \in L_1 \wedge w_2 \in L_2\}$. Moreover, $\epsilon$ denotes the *empty word* of zero length, such that for all $w \in \Sigma^*$, $\epsilon \cdot w = w \cdot \epsilon = w$. Further, we allow non-negative integers as exponents meaning repeated concatenation, such that for an $L \subseteq \Sigma^*$, we have $L^n = L^{n-1} \cdot L$ for $n > 0$ and $L^0 = \{\epsilon\}$. We define that $n < \infty$ for all numbers $n$. The semantics of unordered concatenation is defined in terms of permutations. By $\mathsf{Perm}(\{1,\ldots,n\})$ we mean the set of permutations of $\{1,\ldots,n\}$. If $\sigma \in \mathsf{Perm}(\{1,\ldots,n\})$, we assume $\sigma = \sigma_1,\ldots,\sigma_n$. For convenience, we recall in Definition 2 the language denoted by a regular expression, and extend it to unordered concatenation and numerical constraints.

**Definition 2 (Language).** *The language $\|r\|$ denoted by a regular expression $r \in \mathcal{R}_\Sigma$, is defined in the following inductive way:*

$$
\begin{aligned}
&\|r_1 + r_2\| = \|r_1\| \cup \|r_2\|\\
&\|r_1 \cdot r_2\| = \|r_1\| \cdot \|r_2\|\\
&\|\&(r_1,\ldots,r_n)\| = \textstyle\bigcup_{\sigma \in \mathsf{Perm}(\{1,\ldots,n\})} \|r_{\sigma_1}\| \cdots \|r_{\sigma_n}\|\\
&\|r^{l..u}\| = \textstyle\bigcup_{l \le i \le u} \|r\|^i\\
&\textit{for } a \in \Sigma \cup \{\epsilon\}, \|a\| = \{a\}
\end{aligned}
$$

Some examples of regular expressions and their languages are: $\|\&(ab,c)\| = \{abc, cab\}$, $\|\&(a,b,c)\| = \{abc, bac, acb, bca, cab, cba\}$, and $\|(a+b)^{1..2}\| = \{a, b, aa, ab, ba, bb\}$. Note that unordered concatenation is not associative, for example: $\|\&(\&(a,b),c)\| = \{abc, bac, cab, cba\} \neq \{abc, acb, bca, cba\} = \|\&(a,\&(b,c))\|$.

## 3 Complexity of Membership under Unordered Concatenation

The *membership*-problem is to decide, given a regular expression with unordered concatenation $r \in \mathcal{R}_\Sigma$, and a word $w \in \Sigma^*$, whether $w \in \|r\|$. This is also called *matching.* For regular expressions with numerical constraints (without unordered concatenation), the membership problem is known to be in P [17]. NP-hardness of membership for regular expressions with interleaving was shown by Ogden et al. [23]. The proof cannot easily be modified to fit the case for unordered concatenation.

It is not hard to show that the membership problem for regular expressions with numerical constraints and unordered concatenation is in NP. The certificate for an instance of the problem, consists in making all the necessary choices in the regular expression, such that one can see that the word is in the language. The size of the certificate is polynomial in the lengths of the word and the regular expression. An explicit construction is given in [15, p.53].

To show that membership is NP-hard, we use a reduction from satisfiability of propositional formulas, first shown NP-hard by Cook [6]. By a result of Tseitin [25] we can assume the formulas are in conjunctive normal form. In the remainder of this section we will not use the numerical constraints. The usage of the exponents in the expressions and words in this section is only a shorthand for repeated concatenation. The alphabet consists of the names of the Boolean variables. Given a formula with $c$ clauses and $v$ variables, we construct a regular expression $r$ which is a unordered concatenation of $c + v$ expressions. The first $c$ expressions in the unordered concatenation each represent a clause. In these *clause-expressions*, disjunction is represented by choice (+), a positive literal is represented by itself, and a negated literal is represented by concatenating the respective letter with itself $c + 1$ times. The last $v$ expressions in the unordered concatenation, one for each variable $x$, are of the following form $((x + \epsilon)^c x^{c^2}) + (x^{c+1} + \epsilon)^c$. The word $w$ that we will check for membership, is ${x_1}^{c^2+c} \cdots {x_v}^{c^2+c}$, assuming the variables are $x_1, \ldots, x_v$.

*Example 3.* For the purpose of an example, let the formula be $(x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee \neg x_5 \vee x_6) \wedge (x_3 \vee \neg x_6)$. Then $v = 6$, $c = 3$ and $\Sigma = \{x_1, x_2, x_3, x_4, x_5, x_6\}$. The regular expression becomes $\&((x_1 + x_2^4 + x_3^4 + x_4), (x_3 + x_5^4 + x_6), (x_3 + x_6^4), r_1, r_2, r_3, r_4, r_5, r_6)$, where each $r_i$ is $((x_i + \epsilon)^3 x_i^9) + (x_i^4 + \epsilon)^3$. The word to check membership in the language of this regular expression becomes $x_1^{12} \cdots {x_6}^{12}$.

It remains to show that the problem instance of the membership problem is only polynomially larger than the propositional formula, and that the word is in the language of the regular expression if and only if the propositional formula is satisfiable. For reasons of space, these proofs must be left out, but they can be found in [15, p.54-57]. The intuition is that in the choices in the last $v$ parts of the regular expressions, the left choice can be used if the corresponding variable can be true, and the right choice if it can be false, and that in the sub-expressions representing the clauses, the chosen subexpression must be true in the formula.

Note that it is enough for NP-hardness with one single top-level unordered concatenation.

## 4 Term Trees, Positions, and Subscripting

In this section we will define notation necessary for the later sections. Given a regular expression $r$, we follow [1] and define the term tree of $r$ as the tree where the root is labeled with the main operator (choice, concatenation, or star) and the subtrees are the term trees of the subexpression(s). If $a \in \Sigma \cup \{\epsilon\}$ the term tree is a single root-node with $a$ as label.

We use $\langle n_1, \ldots, n_k \rangle$, a possibly empty sequence of natural numbers, to denote a position in a term tree. We let $p, q$, including subscripted variants, be variables for such possibly empty sequences of natural numbers. The position of the root is $\langle\rangle$. If $r = r_1 \cdot r_2$ or $r = r_1 + r_2$, and $n_1 \in \{1, 2\}$, the position $\langle n_1, \ldots, n_k\rangle$ in $r$ is the position $\langle n_2, \ldots, n_k\rangle$ in the subtree of child $n_1$, that is, in the term tree of $r_{n_1}$. If $r = {r_1}^*$, the position $\langle 1, n_1, \ldots, n_k\rangle$ in $r$ is the position $\langle n_1, \ldots, n_k\rangle$ in the term tree of $r_1$. Let $\mathsf{pos}(r)$ be the set of positions in $r$.

$p \odot q$ will be used for the concatenation of positions $p$ and $q$. We will also use this notation for concatenating a position with each element in a list of positions, and for concatenating a position with each element of a set of lists of positions.

Whenever concatenating with a position of length one, we will often omit the sign $\odot$ and abbreviate, such that for example $p1 = p \odot \langle 1\rangle$, $2S = \langle 2\rangle \odot S$, $ir = \langle i\rangle \odot r$, etc.

For a position $p$ in $r$ we will denote the subexpression rooted at this position by $r[p]$. Note that $r[\langle\rangle] = r$. We also set $r[\epsilon] = \epsilon$. Furthermore, given $p_1, \ldots, p_n$ in $\mathsf{pos}(r) \cup \{\epsilon\}$, put $r[p_1 \cdot \cdots \cdot p_n] = r[p_1] \cdot \cdots \cdot r[p_n]$. Lastly, we lift $r[]$ to sets of string, such that if $S \subseteq \mathsf{pos}(r)^*$, then $r[S] = \{r[w] \mid w \in S\}$.

The concept of *marked expressions* will be important in this paper. It was first used in a similar context by Brüggemann-Klein & Wood [5]. For any regular expression $r$, let $\mu(r)$ be the marked expression, where every instance of any symbol from $\Sigma$ is substituted by its position in the expression. Note that, e.g., $\mu(b) = \mu(a) = \langle\rangle$, which shows that marking is not injective. Furthermore $\|\mu(r_1 \cdot r_2)\| = 1\|\mu(r_1)\| \cdot 2\|\mu(r_2)\|$, $\|\mu(r_1 + r_2)\| = 1\|\mu(r_1)\| \cup 2\|\mu(r_2)\|$, and $\|\mu(r^*)\| = 1\|\mu(r)^*\|$.

*Example 4.* Consider $\Sigma = \{a, b\}$ and $r = (\&(a^2, b))^{3..4}$. Then $\mu(r) = (\&(\langle 1,1,1\rangle^2, \langle 1,2\rangle))^{3..4}$. The term trees of $r$ and $\mu(r)$ are shown in Fig. 1.

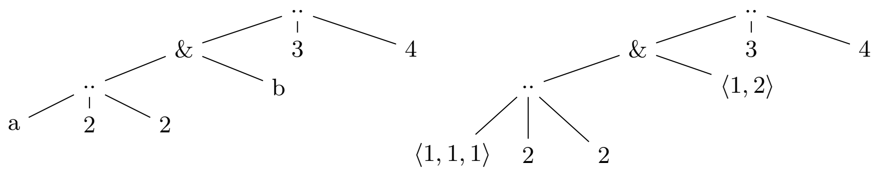


**Fig. 1.** Term trees for $(\&(a^2, b))^{3..4}$ and $\mu((\&(a^2, b))^{3..4})$

## 5 Finite Automata with Counters

In this section we describe the finite automata with counters (FAC). FACs are, of course, based on classical finite automata, but extended with a finite set of *counters*. A configuration of the FAC includes a mapping, called *counter state*, from the counters to the non-negative integers. For subexpressions with numerical constraints we use the counters to keep track of the number of times the subexpression has been matched, and use this to control that the numerical constraints are not violated. For regular expressions with unordered concatenation we use the counters to keep track of which parts of a unordered concatenation have been matched. We keep a counter for every argument in every unordered concatenation. All counters are initially 0. A part of an unordered concatenation can only be used for matching if the corresponding counter is 0, the counter will then be increased to 1. The matching process is only allowed to leave the unordered concatenation when all parts, except those that can match $\epsilon$, have been matched. The counters are then reset to 0.

Let $\mathcal{C}$ be the set of positions of subexpressions we need to keep track of. We model counter states as mappings $\gamma : \mathcal{C} \to \mathbb{N}_0$. Let $\gamma_0$ be the counter state in which all counters are 0. We define an *update instruction* $\psi$ as a partial mapping from $\mathcal{C}$ to $\{\mathsf{inc}, \mathsf{res}, \mathsf{one}\}$ ($\mathsf{inc}$ for *increment*, $\mathsf{res}$ for *reset*, $\mathsf{one}$ for setting to 1). Update instructions $\psi$ define mappings $f_\psi$ between counter states in the following way: If $\psi(p) = \mathsf{inc}$, then $f_\psi(\gamma)(p) = \gamma(p) + 1$, if $\psi(p) = \mathsf{res}$ then $f_\psi(\gamma)(p) = 0$, if $\psi(p) = \mathsf{one}$ then $f_\psi(\gamma)(p) = 1$, and otherwise $f_\psi(\gamma)(p) = \gamma(p)$.

**Definition 5 (Satisfaction of Update Instructions).** *We define a satisfaction relation between update instructions and counter states. Given* $\gamma : \mathcal{C} \to \mathbb{N}_0$, $\psi : \mathcal{C} \to \{\mathsf{inc}, \mathsf{res}, \mathsf{one}\}$, $\mathsf{min} : \mathcal{C} \to \mathbb{N}_0$, *and* $\mathsf{max} : \mathcal{C} \to \mathbb{N}_1$, $\gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi$ *is defined by the following inductive rules:*

$$\begin{array}{lcl}
\gamma \models^{\mathsf{max}}_{\mathsf{min}} \varnothing & & \\
\gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \cup \{p \mapsto \mathsf{inc}\} & \Leftrightarrow & \gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \wedge \gamma(p) < \mathsf{max}(p) \\
\gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \cup \{p \mapsto \mathsf{res}\} & \Leftrightarrow & \gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \wedge \gamma(p) \geq \mathsf{min}(p) \\
\gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \cup \{p \mapsto \mathsf{one}\} & \Leftrightarrow & \gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \wedge \gamma(p) \geq \mathsf{min}(p)
\end{array}$$

The intuition of Definition 5 is that a counter can only be increased if the value is smaller than the maximum, while a value can only be reset if it's value is at least as large as the minimum. Given mappings $\mathsf{max}$ and $\mathsf{min}$, two update instructions are called *overlapping*, if there is a counter state that satisfies both of the update instructions. Overlap can be detected in linear time: For every $p \in \mathcal{C}$ such that $p$ is mapped to $\mathsf{inc}$ by one of the update instructions, and $p$ is mapped to either $\mathsf{res}$ or $\mathsf{one}$ by the other update instruction, it must hold that $\mathsf{min}(p) < \mathsf{max}(p)$.

### 5.1 Finite Automata with Counters

**Definition 6 (Finite Automata with Counters).** *A finite automaton with counters (FAC) is a tuple* $(\Sigma, Q, \mathcal{C}, \mathcal{A}, \Phi, \mathsf{min}, \mathsf{max}, q^I, \mathcal{F})$. *The members of the tuple are described below:*

- $\Sigma$ *is a finite, non-empty set (the alphabet).*
- $Q$ *and* $\mathcal{C}$ *are finite sets of* states *and* counters, *respectively.*
- $q^I \in Q$ *is the* initial state.
- $\mathcal{A} : Q - \{q^I\} \to \Sigma$ *maps each non-initial state to the letter which is matched when entering the state.*
- $\Phi$ *maps each state to a set of pairs of a state and an update instruction.* $\Phi : Q \to \wp(Q \times (\mathcal{C} \to \{\mathsf{inc}, \mathsf{res}, \mathsf{one}\}))$.
- $\mathsf{min} : \mathcal{C} \to \mathbb{N}_0$ *and* $\mathsf{max} : \mathcal{C} \to \mathbb{N}_1$ *are the counter-conditions.*
- $\mathcal{F} \subset Q \times (\mathcal{C} \to \{\mathsf{res}\})$ *describes the* final configurations *(See Definition 7).*

Running or executing an FAC is defined in terms of *transitions* between *configurations*. The configurations of an FAC are pairs, where the first element is a member of $Q$, and the second element is a counter state.

**Definition 7 (Configuration of an FAC).** *A* configuration *of an FAC is a pair* $(q, \gamma)$*, where* $q \in Q$ *is the current state and* $\gamma : \mathcal{C} \to \mathbb{N}_0$ *is the counter state. A configuration* $(q, \gamma)$ *is final, if there is* $(q, \psi) \in \mathcal{F}$ *such that* $\gamma \models_{\mathsf{min}}^{\mathsf{max}} \psi$*.*

Intuitively, the first member of each of the pairs mapped to by $\Phi$, is the state that can be entered, and the second member describes the changes to the current configuration of the automaton in this step. Thus, $\Phi$ and $\mathcal{A}$ together describe the possible transitions of the automaton. This is formalized as the transition function $\delta$.

**Definition 8 (Transition Function of an FAC).** *For an FAC* $(\Sigma, Q, \mathcal{C}, \mathcal{A}, \Phi, q^I, \mathcal{F})$*, the transition function* $\delta$ *is defined for any configuration* $(q, \gamma)$ *and letter* $l$ *by* $\delta((q, \gamma), l) = \{(p, f_\psi(\gamma)) \mid \mathcal{A}(p) = l, (p, \psi) \in \Phi(q), \gamma \models_{\mathsf{min}}^{\mathsf{max}} \psi\}$*.*

**Definition 9 (Deterministic FAC).** *An FAC* $(\Sigma, Q, \mathcal{C}, \mathcal{A}, \Phi, q^I, \mathcal{F})$ *is deterministic if and only if* $|\delta((q, \gamma), l)| \leq 1$ *for all* $q \in Q, l \in \Sigma$ *and* $\gamma : \mathcal{C} \to \mathbb{N}_0$*.*

Deciding whether an FAC is deterministic can be done in polynomial time as follows: For each state $p$, for each two different $(p_1, \psi_1)$, $(p_2, \psi_2)$ both in $\Phi(p)$, verify that either $\mathcal{A}(p_1) \neq \mathcal{A}(p_2)$ or that $\psi_1$ and $\psi_2$ are not overlapping.

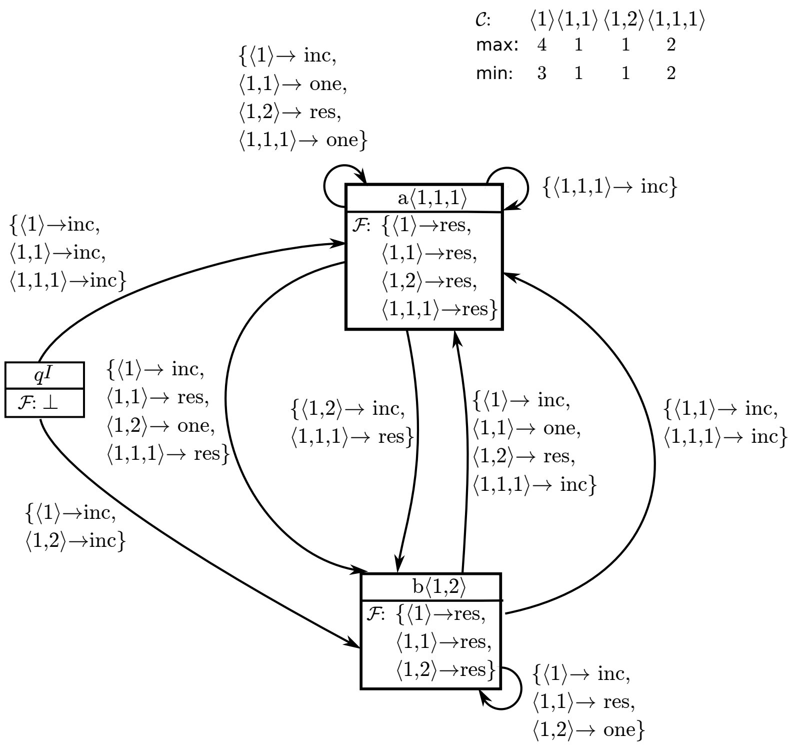
## 5.2 Word Recognition

An FAC either *accepts* or *rejects* a given input. A deterministic FAC recognizes a word by treating letters in the word one by one. It starts in the *initial configuration* $(q^I, \gamma_0)$. An FAC in configuration $(q, \gamma)$, with letter $l \in \Sigma$ next in the word, will reject the word if $\delta((q, \gamma), l)$ is empty. Otherwise it enters the unique configuration $(q', \gamma') \in \delta((q, \gamma), l)$. If the whole word has been read, a deterministic FAC accepts the word if and only if it is in a final configuration. The subset of $\Sigma^*$ consisting of words being accepted by an FAC A is denoted $\|A\|$. A deterministic FAC accepts or rejects a word in time linear in the length of the word.

*Example 10.* Let $\Sigma = \{a, b\}$, $Q = \{q^I, a\langle 1,1,1\rangle, b\langle 1,2\rangle\}$, and $\mathcal{C} = \{\langle 1\rangle, \langle 1,1\rangle, \langle 1,1,1\rangle, \langle 1,2\rangle\}$. Figure 2 illustrates a deterministic FAC $(\Sigma, Q, \mathcal{C}, \mathcal{A}, \Phi, \mathsf{min}, \mathsf{max}, q^I, \mathcal{F})$ which recognizes $\|(\&(a^2, b))^{3..4}\|$. Note that the names of the non-initial states are decorated with the values of $\mathcal{A}$. Every state is depicted as a rectangle with the name of the state, and with $\mathcal{F}$ described by the reset instructions. Every member of $\Phi$ is shown as an arrow, annotated with the corresponding update instruction. $\mathcal{C}$, $\mathsf{min}$, and $\mathsf{max}$ are shown in the top of the figure. The sequence of configurations of this FAC while recognizing *aabbaabaa* is :

$$\begin{array}{ll}
(q^I, & \gamma_0) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 1, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 1, \langle 1,2\rangle \mapsto 0\}) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 1, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 2, \langle 1,2\rangle \mapsto 0\}) \\
(b\langle 1,2\rangle, & \{\langle 1\rangle \mapsto 1, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 0, \langle 1,2\rangle \mapsto 1\}) \\
(b\langle 1,2\rangle, & \{\langle 1\rangle \mapsto 2, \langle 1,1\rangle \mapsto 0, \langle 1,1,1\rangle \mapsto 0, \langle 1,2\rangle \mapsto 1\}) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 2, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 1, \langle 1,2\rangle \mapsto 1\}) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 2, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 2, \langle 1,2\rangle \mapsto 1\}) \\
(b\langle 1,2\rangle, & \{\langle 1\rangle \mapsto 3, \langle 1,1\rangle \mapsto 0, \langle 1,1,1\rangle \mapsto 0, \langle 1,2\rangle \mapsto 1\}) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 3, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 1, \langle 1,2\rangle \mapsto 1\}) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 3, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 2, \langle 1,2\rangle \mapsto 1\})
\end{array}$$

The last configuration is final, since $\mathsf{min}(\langle 1\rangle) \leq 3$, $\mathsf{min}(\langle 1,1\rangle) \leq 1$, and $\mathsf{min}(\langle 1,1,1\rangle) \leq 2$.



C: ⟨1⟩ ⟨1,1⟩ ⟨1,2⟩ ⟨1,1,1⟩
max: 4 1 1 2
min: 3 1 1 2
{⟨1⟩→ inc, ⟨1,1⟩→ one, ⟨1,2⟩→ res, ⟨1,1,1⟩→ one}
{⟨1,1,1⟩→ inc}
a⟨1,1,1⟩
F: {⟨1⟩→res, ⟨1,1⟩→res, ⟨1,2⟩→res, ⟨1,1,1⟩→res}
{⟨1⟩→inc, ⟨1,1⟩→inc, ⟨1,1,1⟩→inc}
qI
F: ⊥
{⟨1⟩→ inc, ⟨1,1⟩→ res, ⟨1,2⟩→ one, ⟨1,1,1⟩→ res}
{⟨1,2⟩→ inc, ⟨1,1,1⟩→ res}
{⟨1⟩→ inc, ⟨1,1⟩→ one, ⟨1,2⟩→ res, ⟨1,1,1⟩→ inc}
{⟨1,1⟩→ inc, ⟨1,1,1⟩→ inc}
{⟨1⟩→inc, ⟨1,2⟩→inc}
b⟨1,2⟩
F: {⟨1⟩→res, ⟨1,1⟩→res, ⟨1,2⟩→res}
{⟨1⟩→ inc, ⟨1,1⟩→ res, ⟨1,2⟩→ one}

**Fig. 2.** Illustration of FAC recognizing $\|(\&(a^2, b))^{3..4}\|$

For each letter matched by the FAC, it must test satisfiability of the update instructions corresponding to transitions to the states with a matching letter. Since the sum of these update instructions is smaller than the whole FAC, and testing satisfiability of update instructions is linear, we get the following:

**Lemma 11 (Linear-time recognition).** *For any deterministic FAC $A = (\Sigma, Q, \mathcal{C}, \mathcal{A}, \Phi, \mathsf{min}, \mathsf{max}, q^I, \mathcal{F})$, if $\sigma(A)$ is the size of $A$, then for any word $w \in \Sigma^*$, the FAC $A$ accepts or rejects $w$ in time $O(|w|\sigma(A))$.*

## 6 Unambiguity

In this section we will define the right unambiguity we need for constructing deterministic automata. *Strongly 1-unambiguous* regular expressions were first defined by Koch & Scherzinger [20], but the definitions used here also bear on Gelade et al. [8]. A deterministic FAC can be constructed in polynomial time from such expressions. We recall the definition of 1-unambiguity such that the difference with strong 1-unambiguity becomes clear.

**Definition 12 (1-unambiguity [3,5]).** *A regular expression $r$ is 1-unambiguous if for all $upv, uqw \in \|\mu(r)\|$, where $u, v, w \in (\mathsf{pos}(r))^*$ and $p, q \in \mathsf{pos}(r)$, $r[p] = r[q]$ implies $p = q$.*

Strong 1-unambiguity is needed to prevent unambiguities related to the numerical constraints. For example, $(a^{3..4})^2$ is 1-unambiguous, but there is an ambiguity related to which of the two numerical constraints should be increased when seeing the fourth $a$ in a word. To formalize this ambiguity we will use *subscripted* expressions, and the languages of these. Subscripting is inspired by the *bracketing* used by Koch & Scherzinger [20] and Gelade et al. [8]. The intuition is that for a regular expression $r$, the subscripted regular expression $\mathsf{ss}(r)$, is such that all subexpressions of the form $r_1^{l..u}$ or $\&(r_1, \ldots, r_n)$ are subscripted with their position in the term tree. For example, $\mathsf{ss}((\&(a^2, b))^{3..4}) = (\&(a^2_{\langle 1,1\rangle}, b)_{\langle 1\rangle})^{3..4}_{\langle\rangle}$.

To define the language of a subscripted expression we will use some more notation: For a position $p = \langle j_1, \ldots, j_n\rangle$, and a positive integer $i$, $pi$ denotes the position $\langle j_1, \ldots, j_n, i\rangle$. For a regular expression $r$, let $\Gamma_r = \bigcup_{p\in\mathsf{pos}(r)}\{\uparrow_p, \downarrow_p\}$. For a set $L$, $\epsilon^L$ denotes $\{\epsilon\}\cap L$ and $L^>$ denotes $L - \{\epsilon\}$. The language of a subscripted expression $r$ is a set of strings over $\mathsf{sym}(r) \cup \Gamma_r$. For the not subscripted parts we use the same rules as in Definition 2, while $\|r_p^{l..u}\| = (\bigcup_{i=l}^{u}(\{\uparrow_{p1}\}\cdot\|r\|)^i)\cdot\{\downarrow_{p1}\}$, and $\|\&(r_1, \ldots, r_n)_p\| =$

$$\epsilon^{\|\&(r_1,\ldots,r_n)\|} \cup \left( \bigcup_{\sigma\in\mathsf{Perm}(\{1,\ldots,n\})} \begin{pmatrix} \epsilon^{\|r_{\sigma_1}\|} \cup \{\uparrow_{p\sigma_1}\}\cdot\|r_{\sigma_1}\|^> \\ \cdots \\ \epsilon^{\|r_{\sigma_n}\|} \cup \{\uparrow_{p\sigma_n}\}\cdot\|r_{\sigma_n}\|^> \end{pmatrix}^> \right) \cdot \{\downarrow_{p1}\cdots\downarrow_{pn}\}$$

The ambiguity observed in $(a^{3..4})^2$ corresponds to the fact that there are $u, v, w$ such that both $u\cdot a\cdot\uparrow_{\langle 1,1\rangle}\cdot a\cdot v$ and $u\cdot a\cdot\downarrow_{\langle 1,1\rangle}\cdot\uparrow_{\langle 1\rangle}\cdot\uparrow_{\langle 1,1\rangle}\cdot a\cdot w$ are words in $\|\mathsf{ss}((a^{3..4})^2)\|$.

**Definition 13 (Strong 1-unambiguity [8,20]).** *A regular expression $r$ is strongly 1-unambiguous if it is 1-unambiguous, and for all $u\alpha av, u\beta bw \in \|\mathsf{ss}(r)\|$, where $a, b \in \mathsf{sym}(r)$, $\alpha, \beta \in \Gamma_r^*$ and $u, v, w \in (\Sigma\cup\Gamma_r)^*$, $a = b$ implies $\alpha = \beta$.*

Examples of expressions that are not strongly 1-unambiguous are $(a^{1..2})^{1..2}$, $(a^*a)^{2..3}$ and $(\&(a^{1..2}, b))^{1..2}$, while $(a+b)^{1..4}$ is strongly 1-unambiguous.

We can now formulate the main result of this paper. The construction of an FAC from a regular expression is based on first, last, and follow sets. There is not space for it in this paper, but it can be seen in [15, p.86-107].

**Theorem 14.** *For any regular expression $r$, we can in polynomial time construct an FAC recognizing exactly $\|r\|$. For any word $w$, and any strongly 1-unambiguous regular expression $r$, we can in polynomial time decide whether $w \in \|r\|$.*

## 7 Related Work

The present paper is based on Chapter 3 of Hovland [15]. Proofs, definitions, and examples left out of the present paper for reasons of space can be found in [15]

Sperberg-McQueen [24] has studied regular expressions with numerical constraints and a translation to finite automata with counters, though no proofs are given. Gelade et al. [9,10] and Gelade et al. [8] also wrote about this, including full proofs. The latter was published simultaneously with the paper [14]. The present paper is based on [14], but also incorporates ideas from [8], most notably the bracketing, which was inspiration for the subscripted expressions. Section 6 from [8], including the proofs for Sect. 6 in the Appendix of [8] has inspired some of the content concerning subscripting, strong 1-unambiguity, and proving correctness of the construction of FACs.

Kilpeläinen & Tuhkanen [17,18,19], Gelade [7], Gelade et al. [9,10], and Gelade et al. [8] also investigated properties of the regular expressions with numerical constraints, and give algorithms for membership. Stockmeyer & Meyer [22] study the regular expressions with squaring, a subclass of the regular expressions with numerical constraints. Colazzo, Ghelli & Sartiani, describe in [12] an algorithm for linear-time membership in a subclass of regular expressions called collision-free. The collision-free regular expressions have at most one occurrence of each symbol from $\Sigma$, and the counters (and the Kleene star) can only be applied directly to letters or disjunctions of letters. The latter class is strictly included in the class of strongly 1-unambiguous regular expressions.

Extensions of finite automata similar to Finite Automata with Counters have been studied by many authors. The earliest is the treatment of multicounter automata by Greibach [13]. In the multicounter-automata counters can only increase or decrease by 1, and the transition function cannot read the values of the counters. An instruction corresponding to res or one does therefore not exist. Sperberg-McQueen [24] describe the *Counter-extended Finite-state Automata* (CFA) and Gelade et al. [8] describe the *CNFA*. Both of the latter automata classes use separate expressions for the update instructions (called *actions* by Sperberg-McQueen) and for specifying the conditions/guards. In the FACs in the present paper these guards (or conditions) are implicit, and calculated directly from the update instructions. The language for guards is also quite expressive, and this leads to higher expressive power in the CNFAs and CFAs compared to FACs. Gelade et al. [9,10] describe *NFA(#)*s which have a counter for each state. The FACs described in the present paper are a variant of those described by the author in [14], modified to fit the combination of numerical constraints and unordered concatenation. In the case of unordered concatenation, only the values 0 and 1 are used. The update instruction one is new.

Brüggemann-Klein [4,2] gives an algorithm for deciding 1-unambiguity of regular expressions with unordered concatenation. Unordered concatenation is also mentioned in [5,3]. Strong 1-unambiguity has also been mentioned by Brüggemann-Klein & Wood [5,3] and Sperberg-McQueen [24], and Gelade et al. [8]. The first in-depth study of strong 1-unambiguity was by Koch & Scherzinger [20].

## 8 Conclusion

We have studied the membership problem for regular expressions extended with numerical constraints and with unordered concatenation, an operator similar to "&" in SGML. The membership problem was shown to be NP-complete already without the numerical constraints. We defined *Finite Automata with Counters* (FAC). There is a polynomial-time translation from the regular expressions with numerical constraints and unordered concatenation to FACs. Further we defined *strongly 1-unambiguous regular expressions*, a subset of the regular expressions with numerical constraints and unordered concatenation in constraint normal form, and for which the FAC resulting from the translation is deterministic. The deterministic FAC can recognize the language of the given regular expression in time linear in the size of word to be tested. Testing whether an FAC is deterministic can be done in polynomial time.

## References

1. Bezem, M., Klop, J.W., de Vrijer, R. (eds.): Term Rewriting Systems. Cambridge University Press (2003), http://www.cs.vu.nl/~terese
2. Brggemann-Klein, A.: Compiler-construction tools and techniques for SGML parsers: Difficulties and solutions (May 1994), http://xml.coverpages.org/brugg-standardEP-ps.gz
3. Brüggemann-Klein, A.: Regular expressions into finite automata. Theoretical Computer Science 120(2), 197–213 (1993)
4. Brüggemann-Klein, A.: Unambiguity of Extended Regular Expressions in SGML Document Grammars. In: Lengauer, T. (ed.) ESA 1993. LNCS, vol. 726, pp. 73–84. Springer, Heidelberg (1993)
5. Brüggemann-Klein, A., Wood, D.: One-unambiguous regular languages. Information and Computation 140(2), 229–253 (1998)
6. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC, pp. 151–158. ACM (1971)
7. Gelade, W.: Succinctness of Regular Expressions with Interleaving, Intersection and Counting. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 363–374. Springer, Heidelberg (2008)

8. Gelade, W., Gyssens, M., Martens, W.: Regular Expressions with Counting: Weak versus Strong Determinism. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 369–381. Springer, Heidelberg (2009), http://lrb.cs.uni-dortmund.de/~martens/data/mfcs09-appendix.pdf
9. Gelade, W., Martens, W., Neven, F.: Optimizing Schema Languages for XML: Numerical Constraints and Interleaving. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 269–283. Springer, Heidelberg (2006)
10. Gelade, W., Martens, W., Neven, F.: Optimizing schema languages for XML: Numerical constraints and interleaving. SIAM J. Comput. 38(5), 2021–2043 (2009)
11. Ghelli, G., Colazzo, D., Sartiani, C.: Linear time membership for a class of XML types with interleaving and counting. In: PLAN-X (2008)
12. Ghelli, G., Colazzo, D., Sartiani, C.: Linear time membership in a class of regular expressions with interleaving and counting. In: Shanahan, J.G., Amer-Yahia, S., Manolescu, I., Zhang, Y., Evans, D.A., Kolcz, A., Choi, K.S., Chowdhury, A. (eds.) CIKM, pp. 389–398. ACM (2008)
13. Greibach, S.A.: Remarks on blind and partially blind one-way multicounter machines. Theor. Comput. Sci. 7, 311–324 (1978)
14. Hovland, D.: Regular Expressions with Numerical Constraints and Automata with Counters. In: Leucker, M., Morgan, C. (eds.) ICTAC 2009. LNCS, vol. 5684, pp. 231–245. Springer, Heidelberg (2009)
15. Hovland, D.: Feasible Algorithms for Semantics — Employing Automata and Inference Systems. Ph.D. thesis, Universitetet i Bergen (2010), http://hdl.handle.net/1956/4325
16. ISO 8879. Information processing — text and office systems — standard generalized markup language (SGML) (October 1986)
17. Kilpeläinen, P., Tuhkanen, R.: Regular expressions with numerical occurrence indicators - preliminary results. In: Kilpeläinen, P., Päivinen, N. (eds.) SPLST, pp. 163–173. University of Kuopio, Department of Computer Science (2003)
18. Kilpeläinen, P., Tuhkanen, R.: Towards efficient implementation of XML schema content models. In: Munson, E.V., Vion-Dury, J.Y. (eds.) ACM Symposium on Document Engineering, pp. 239–241. ACM (2004)
19. Kilpeläinen, P., Tuhkanen, R.: One-unambiguity of regular expressions with numeric occurrence indicators. Information and Computation 205(6), 890–916 (2007)
20. Koch, C., Scherzinger, S.: Attribute grammars for scalable query processing on XML streams. VLDB J. 16(3), 317–342 (2007)
21. Mayer, A.J., Stockmeyer, L.J.: Word problems-this time with interleaving. Inf. Comput. 115(2), 293–311 (1994)
22. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: Proceedings of FOCS, pp. 125–129. IEEE (1972)
23. Ogden, W.F., Riddle, W.E., Rounds, W.C.: Complexity of expressions allowing concurrency. In: POPL, pp. 185–194 (1978)
24. Sperberg-McQueen, C.M.: Notes on finite state automata with counters (2004), http://www.w3.org/TR/xml/
25. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Studies in Constructive Mathematics and Mathematical Logic, Part 2, pp. 115–225 (1968)

# Characterizing the Rational Functions by Restarting Transducers

Norbert Hundeshagen and Friedrich Otto

Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
{hundeshagen,otto}@theory.informatik.uni-kassel.de

**Abstract.** A *restarting transducer* is a restarting automaton that is equipped with an *output function*. Accordingly, restarting transducers compute binary relations, and deterministic restarting transducers compute functions. Here we characterize the *rational functions* and some of their proper subclasses by certain types of deterministic restarting transducers with window size one.

## 1 Introduction

The restarting automaton [8] was invented to model the so-called "analysis by reduction," which is a linguistic technique used to analyze sentences of natural languages with free word order. A restarting automaton consists of a finite-state control, a flexible tape with end markers, and a read/write window of a fixed size working on that tape. It works in cycles, where in each cycle it performs a single rewrite operation that shortens the tape contents. After a finite number of cycles, it halts and accepts (or rejects). Thus, a restarting automaton is a *language accepting device*. The real goal of performing analysis by reduction, however, is not simply to accept or reject a given input sentence, but to extract information from that sentence and to translate it into another form, e.g., into a formal representation. Accordingly, we are interested in computing *binary relations*. Observe that rational and subsequential relations have found many applications in e.g. compiler construction (see, e.g., [1]) and natural language and speech processing (see, e.g., [11]).

In [6] a binary relation is associated to a (deterministic) restarting automaton $M$ by splitting its tape alphabet $\Gamma$ into three (disjoint) parts: an *input alphabet* $\Sigma$, an *output alphabet* $\Delta$, and a set of *auxiliary symbols* $\Gamma \smallsetminus (\Sigma \cup \Delta)$. Then a pair $(u, v) \in \Sigma^* \times \Delta^*$ belongs to the relation $\mathrm{Rel}_{\mathrm{io}}(M)$ associated with $M$ if and only if $M$ accepts a word from the shuffle of $u$ and $v$, that is, if and only if $M$ accepts a word $w \in (\Sigma \cup \Delta)^*$ such that $\mathsf{Pr}^{\Sigma}(w) = u$ and $\mathsf{Pr}^{\Delta}(w) = v$. Here $\mathsf{Pr}^{\Sigma}$ ($\mathsf{Pr}^{\Delta}$) denotes the projection from $(\Sigma \cup \Delta)^*$ onto $\Sigma^*$ ($\Delta^*$). This approach, however, is not very satisfying as the "output" $v$ must be given together with the "input" $u$. Here we follow a different approach in that we extend the restarting automaton to a *restarting transducer* by introducing an additional *output function*: whenever a restart operation or an accept operation is performed, then additionally an output string is generated (see Section 2 for the definition).

In the current paper we concentrate on a very restricted class of restarting transducers: restarting transducers with window size one. As each rewrite operation of such a transducer just erases a single symbol from the tape, no auxiliary symbols can be introduced in the course of a computation. Accordingly, we don't need to consider auxiliary symbols at all. It is known that deterministic restarting automata with window size one accept exactly the regular languages [9,12], and this result even extends to deterministic *nonforgetting* restarting automata that are monotone [5]. Here we show that the class of functions computed by deterministic monotone nonforgetting RR-transducers (see below) of window size one coincides with the class of rational functions, and that the deterministic monotone nonforgetting R-transducers (see below) of window size one characterize the subsequential functions. Also we characterize the class of functions that are computed by *deterministic generalized sequential machines* by a restricted type of deterministic R-transducers with window size one. As the corresponding types of restarting automata represent some regular languages in a more succinct way than even nondeterministic finite-state acceptors (see [5,9]), these characterizations show that our restarting transducers can be used to describe certain rational or subsequential functions more succinctly than rational transducers.

The paper is structured as follows. After defining the types of restarting transducers in Section 2 that we want to study, we present some detailed examples. These examples are then used in Section 3 to establish some hierarchy results for the classes of functions that are computed by the various restricted types of deterministic restarting transducers with window size one. The announced characterizations are presented in Section 4.

## 2 Restarting Transducer with Window Size One

We assume that the reader is familiar with the basic concepts and main results from Formal Language Theory, for which we use [4] as our main reference. By REG and CFL we denote the classes of regular and context-free languages. Further, basic knowledge on rational relations and functions is expected, where [2,3] serve as our main references. Finally, concerning restarting automata we refer to [13]. For a finite alphabet $\Sigma$, we use $\Sigma^*$ to denote the set of all words over $\Sigma$ including the empty word $\varepsilon$. For a word $w \in \Sigma^*$, $|w|$ denotes the *length* of $w$.

### 2.1 Restarting Automata with Window Size One

In the literature many different types of restarting automata have been studied. Here we only present those types that we will need in what follows.

A *nonforgetting restarting automaton with window size one*, that is, an nf-RR(1)-automaton, is defined by a 7-tuple $M = (Q, \Sigma, \text{¢}, \$, q_0, 1, \delta)$. Here $Q$ is a finite set of internal states, $\Sigma$ is a finite alphabet, the symbols $\text{¢}, \$ \notin \Sigma$ serve as markers for the left and right border of the workspace, respectively, $q_0 \in Q$ is the initial state, and $\delta$ is the *transition relation* that associates a finite set of transition steps to pairs of the form $(q, a)$, where $q \in Q$ is a state and $a \in$

$\Sigma \cup \{¢, \$\}$ is a possible content of the window. There are four types of transition steps: *move-right steps* of the form $(q', \mathsf{MVR})$, which shift the window one step to the right and change the internal state to $q'$, *delete steps* of the form $(q', \varepsilon)$ that erase the content $a$ of the window and change the internal state to $q'$, *restart steps* of the form $(q', \mathsf{Restart})$ that place the window over the left end of the tape and change the internal state to $q'$, and *accept steps* ($\mathsf{Accept}$), which cause the automaton to halt and accept. Some additional restrictions apply in that the sentinels ¢ and \$ must not be deleted, and that the window must not move right across the \$-symbol. Further, $M$ is *deterministic* if $\delta$ is a partial function. We use the prefix det- to denote deterministic types of nf-RR-automata.

A *configuration* of $M$ is described by a word $\alpha q \beta$, where $q \in Q$ and either $\alpha = \varepsilon$ and $\beta \in \{¢\} \cdot \Sigma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Sigma^*$ and $\beta \in \Sigma^* \cdot \{\$\}$; here $q$ is the current internal state, $\alpha\beta$ is the content of the tape, and the window contains the first symbol of $\beta$. A *restarting configuration* is of the form $q¢w\$$, and an *initial configuration* is of the form $q_0¢w\$$. By $\vdash_M$ we denote the single-step computation relation of $M$, and $\vdash_M^*$ denotes the reflexive transitive closure of $\vdash_M$.

The automaton $M$ proceeds as follows. Starting from a restarting configuration $q¢w\$$, where $q \in Q$ and $w \in \Sigma^*$, the window is shifted to the right by a sequence of move-right steps until a configuration of the form $¢xqay\$$ is reached such that $(q', \varepsilon) \in \delta(q, a)$, where $w = xay$. Now the latter configuration can be transformed into the configuration $¢xq'y\$$, and the computation proceeds with further move-right steps until eventually a restart operation is executed, which yields a restarting configuration of the form $p¢xy\$$ for some $p \in Q$. This sequence of computational steps, which is called a *cycle*, is expressed as $q¢w\$ \vdash_M^c p¢xy\$$, and $\vdash_M^{c^*}$ is used to denote the reflexive transitive closure of this relation. Observe that a cycle must contain exactly one application of a delete step. A computation of $M$ consists of a finite sequence of cycles that is followed by a tail computation, which consists of a sequence of move-right operations (and possibly a single application of a delete operation) that is possibly followed by an accept step. A word $w \in \Sigma^*$ is *accepted* by $M$, if there exists an accepting computation of $M$ which starts with the initial configuration $q_0¢w\$$. By $L(M)$ we denote the language consisting of all words that are accepted by $M$.

Each cycle $C$ of a restarting automaton $M$ contains a unique configuration $\alpha q \beta$ in which a delete step is applied. Then $|\beta|$ is called the *right distance* of $C$, denoted as $D_r(C)$. A sequence of cycles $(C_1, C_2, \ldots, C_n)$ of $M$ is called *monotone* if $D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n)$. A computation of $M$ is called *monotone* if the corresponding sequence of cycles is monotone. Finally, $M$ itself is called *monotone* if *all* its computations that start from an initial configuration are monotone. We use the prefix mon- to denote monotone types of restarting automata. By adopting the proof from [7] it can be shown that monotonicity is decidable for nf-RR(1)-automata.

$M$ is called a *nonforgetting* R(1)-*automaton* (nf-R(1)-automaton, for short) if it is a nonforgetting RR(1)-automaton for which each delete operation is immediately followed by a restart operation. To simplify the description we combine each delete operation of an R(1)-automaton with the subsequent restart operation. Of course, the notions of determinism and monotonicity also apply to $\mathsf{R}(1)$-automata. Further, for each type of (restarting) automaton $\mathsf{X}$, we denote the class of all languages that are accepted by automata of type $\mathsf{X}$ by $\mathcal{L}(\mathsf{X})$. On $\mathsf{nf}$-$\mathsf{RR}(1)$-automata, the following result has been obtained in [5].

**Theorem 1.** $\mathcal{L}(\mathsf{mon\text{-}nf\text{-}R}(1)) = \mathcal{L}(\mathsf{det\text{-}mon\text{-}nf\text{-}RR}(1)) = \mathsf{REG}$.

Finally, the *forgetting* restarting automaton is obtained from the nonforgetting restarting automaton by requiring that it is reset to its initial state by all its restart operations. This is actually the model defined originally (see [8,13]). Therefore, the forgetting variants of the $\mathsf{nf}$-$\mathsf{RR}(1)$- and the $\mathsf{nf}$-$\mathsf{R}(1)$-automaton are simply denoted as $\mathsf{RR}(1)$- and $\mathsf{R}(1)$-automata.

Following [6] we associate a binary relation with a restarting automaton. Let $M$ be an $\mathsf{nf}$-$\mathsf{RR}(1)$-automaton on $\Gamma$, let $\Sigma$ be a proper subalphabet of $\Gamma$, and let $\Delta = \Gamma \smallsetminus \Sigma$. We interpret $\Sigma$ as an *input* and $\Delta$ as an *output alphabet*. Then

$$\mathrm{Rel}_{\mathrm{io}}(M) = \{\, (u,v) \in \Sigma^* \times \Delta^* \mid \exists w \in L(M) : u = \mathsf{Pr}^{\Sigma}(w) \text{ and } v = \mathsf{Pr}^{\Delta}(w) \,\}$$

is called the (*input-output*) *relation* of $M$. By $\mathcal{R}el_{\mathrm{io}}(\mathsf{nf\text{-}RR}(1))$ we denote the class of all these relations of $\mathsf{nf}$-$\mathsf{RR}(1)$-automata. Concerning the relations of this form we have the following characterization. Here $\mathsf{RatRel}$ denotes the class of *rational relations* (see, e.g., [2]).

**Corollary 1.** *Let $R \subseteq \Sigma^* \times \Delta^*$ be a binary relation, where $\Sigma$ and $\Delta$ are disjoint. Then $R$ is a rational relation if and only if it is the relation of a* $\mathsf{mon}$-$\mathsf{nf}$-$\mathsf{R}(1)$-*automaton if and only if it is the relation of a* $\mathsf{det}$-$\mathsf{mon}$-$\mathsf{nf}$-$\mathsf{RR}(1)$-*automaton.*

*Proof.* By a theorem of Nivat (see, e.g., [2]) a binary relation $R \subseteq \Sigma^* \times \Delta^*$ is rational if and only if there exists a regular language $L \subseteq (\Sigma \cup \Delta)^*$ such that $R = \{\, (\mathsf{Pr}^{\Sigma}(w), \mathsf{Pr}^{\Delta}(w)) \mid w \in L \,\}$. By Theorem 1 the class of regular languages coincides with the class of languages that are accepted by $\mathsf{mon}$-$\mathsf{nf}$-$\mathsf{R}(1)$- and $\mathsf{det}$-$\mathsf{mon}$-$\mathsf{nf}$-$\mathsf{RR}(1)$-automata. Thus, if $M$ is an automaton of one of these two types such that $L(M) = L$, then $R$ coincides with the relation $\mathrm{Rel}_{\mathrm{io}}(M)$. □

## 2.2 Restarting Transducer

Here we introduce the *restarting transducer*, that is, a restarting automaton with output. Although this generalization can be applied to all types of restarting automata, we state it here only for $\mathsf{nf}$-$\mathsf{RR}(1)$- and $\mathsf{nf}$-$\mathsf{R}(1)$-automata.

An $\mathsf{nf}$-$\mathsf{RR}(1)$-*transducer*, abbreviated as $\mathsf{nf}$-$\mathsf{RR}(1)$-$\mathsf{Td}$, is given through an 8-tuple $T = (Q, \Sigma, \Delta, ¢, \$, q_0, 1, \delta)$, where $Q$, $\Sigma$, $¢$, $\$$, $q_0$ and the window size 1 are defined as for an $\mathsf{nf}$-$\mathsf{RR}(1)$-automaton, $\Delta$ is a finite output alphabet, and $\delta$ is the *transition relation* that associates a finite set of transition steps to pairs of the form $(q, a)$, where $q \in Q$ and $a \in \Sigma \cup \{¢, \$\}$. There are four types of transition steps: *move-right steps* and *delete steps* are defined as for $\mathsf{nf}$-$\mathsf{RR}(1)$-automata, a *restart step* of the form $(q', \mathsf{Restart}, v)$ places the window over the left end of the tape, it changes the internal state to $q'$, and it *outputs* the word $v \in \Delta^*$, and an *accept step* of the form $(\mathsf{Accept}, v)$ causes the transducer to halt and accept while producing the output $v \in \Delta^*$. The restarting transducer is *deterministic* if $\delta$ is a (partial) function. Further, $T$ is an nf-R(1)-Td if the underlying restarting automaton is an nf-R(1)-automaton. Finally, the R(1)- and the RR(1)*-transducers* are obtained by requiring that each restart operation resets the internal state to the initial state $q_0$.

A *configuration* of $T$ is described by a pair $(\alpha q \beta, z)$, where $\alpha q \beta$ is a configuration of the underlying restarting automaton and $z \in \Delta^*$ is an *output word*. If $\alpha\beta = \text{¢}xay\$$ such that $\delta(q, a)$ contains the triple $(q', \mathsf{Restart}, v)$, then $(\alpha q \beta, z) \vdash_T (q'\alpha\beta, zv)$, that is, $T$ may restart changing its internal state to $q'$ and producing the output $v$. Further, if $\delta(q, a)$ contains the pair $(\mathsf{Accept}, v)$, then $(\alpha q \beta, z) \vdash_T (\mathsf{Accept}, zv)$, that is, $T$ may halt and accept producing the output $v$. An *accepting computation* of $T$ consists of a finite sequence of *cycles* that is followed by an accepting *tail computation*, that is, it can be described as

$$\begin{aligned}(q_0\text{¢}w\$, \varepsilon) \vdash_T^c (q_{i_1}\text{¢}w_1\$, v_1) \vdash_T^c \cdots \vdash_T^c (q_{i_m}\text{¢}w_m\$, v_1 \cdots v_m)\\ \vdash_T^* (\mathsf{Accept}, v_1 \cdots v_m v_{m+1}).\end{aligned}$$

With $T$ we associate the following (*input-output*) *relation*

$$\text{Rel}(T) = \{\, (w, z) \in \Sigma^* \times \Delta^* \mid (q_0\text{¢}w\$, \varepsilon) \vdash_T^* (\mathsf{Accept}, z) \,\}.$$

For $w \in \Sigma^*$, $T(w) = \{\, z \in \Delta^* \mid (w, z) \in \text{Rel}(T) \,\}$. If $T$ is deterministic, then $\text{Rel}(T)$ is obviously the graph of a (partial) function. The corresponding function will be called the *transduction computed by* $T$.

Obviously, the notion of *monotonicity* generalizes from restarting automata to restarting transducers. Further, a restarting transducer is called *proper* if all its accept operations are of the form $(\mathsf{Accept}, \varepsilon)$, that is, in the last step of an accepting computation it can only output the empty word. We use the prefix prop- to denote proper types of restarting transducers.

## 2.3 Examples of Functions Computed by RR(1)-Transducers

Here we present two examples of nonforgetting RR(1)-transducers. In order to describe an nf-RR(1)-transducer in a compact way, we use *meta-instructions* of the form $(q, E_1, a \to \varepsilon, E_2, q'; v)$, where $q, q'$ are internal states of $T$, $E_1, E_2$ are regular expressions, $a \in \Sigma$ is a letter to be deleted, and $v \in \Delta^*$ is an output word (cf., e.g., [10]). This meta-instruction is applicable to configurations of the form $(q\text{¢}w\$, z)$, where $w$ can be factored as $w = xay$ such that $\text{¢}x \in E_1$ and $y\$ \in E_2$, and it transforms this configuration into the configuration $(q'\text{¢}xy\$, zv)$. To describe accepting tail computations we use meta-instructions of the form $(q, E, \mathsf{Accept}; v)$ that enable $T$ to perform an accepting tail computation starting from a configuration of the form $(q\text{¢}w\$, z)$ such that $\text{¢}w\$ \in E$. In this situation $T$ reaches the final configuration $(\mathsf{Accept}, zv)$. Observe that meta-instructions are just a convenient way for describing restarting automata. In particular, such a description is essentially nondeterministic. When it comes to properties like determinism, then one first has to construct the real transition relation of the restarting automaton considered, but this is always possible.

*Example 1.* The function $\tau_1 : a^* \to \{b,c\}^*$ that is defined by $\tau_1(a^{2n}) = b^{2n}$ and $\tau_1(a^{2n+1}) = c^{2n+1}$ ($n \in \mathbb{N}$) is computed by the proper det-mon-nf-RR(1)-transducer $T_1$ that is described by the following meta-instructions:

(1) $(q_0, ¢, a \to \varepsilon, a \cdot (aa)^* \cdot \$, q_1; b)$,
(2) $(q_0, ¢, a \to \varepsilon, (aa)^* \cdot \$, q_2; c)$,
(3) $(q_0, ¢\$, \mathsf{Accept}; \varepsilon)$,
(4) $(q_1, ¢, a \to \varepsilon, a^* \cdot \$, q_1; b)$,
(5) $(q_1, ¢\$, \mathsf{Accept}; \varepsilon)$,
(6) $(q_2, ¢, a \to \varepsilon, a^* \cdot \$, q_2; c)$,
(7) $(q_2, ¢\$, \mathsf{Accept}; \varepsilon)$.

Recall that a restarting transducer produces its outputs only while executing restart and accept transitions, that is, when it executes the last step of a cycle or a tail computation.

*Example 2.* The function $\tau_2 : \{0,1\}^* \to \{0,1\}^*$ that is defined by

$$\tau_2(w) = \begin{cases} 0^{|x|}, & \text{if } w = x0 \text{ and } x \in \{0,1\}^*, \\ 1^{|x|}, & \text{if } w = x1 \text{ and } x \in \{0,1\}^*, \end{cases}$$

is computed by the proper det-RR(1)-transducer $T_2$ that is described by the following meta-instructions, where $a, b \in \{0,1\}$:

(1) $(q_0, ¢, a \to \varepsilon, \{0,1\}^* \cdot b\$, q_0; b)$, (2) $(q_0, ¢b\$, \mathsf{Accept}; \varepsilon)$.

## 3 Relations Computed by nf-RR(1)-Transducers

For each type X of restarting transducers, we use $\mathcal{R}el(\mathsf{X})$ to denote the class of relations that are computed by the transducers of type X. From the above definitions we immediately obtain the following inclusions. Just observe that each det-R(1)- and each det-RR(1)-transducer is necessarily monotone.

**Proposition 1.**

(a) $\mathcal{R}el(\text{det-R(1)-Td}) \subseteq \mathcal{R}el(\text{det-mon-nf-R(1)-Td}) \subseteq \mathcal{R}el(\text{det-mon-nf-RR(1)-Td})$.
(b) $\mathcal{R}el(\text{det-R(1)-Td}) \subseteq \mathcal{R}el(\text{det-RR(1)-Td}) \subseteq \mathcal{R}el(\text{det-mon-nf-RR(1)-Td})$.
(c) $\mathcal{R}el(\text{det-R(1)-Td}) \subseteq \mathcal{R}el(\text{det-mon-nf-R(1)-Td}) \subseteq \mathcal{R}el(\text{mon-nf-R(1)-Td})$.

Obviously, the above inclusions also hold for the corresponding types of *proper* transducers. Further, $\mathcal{R}el(\text{prop-X}) \subseteq \mathcal{R}el(\mathsf{X})$ holds obviously for each type X of restarting transducers. Using Examples 1 and 2 it can be shown that several of the inclusions above are actually strict.

**Proposition 2.** *The function $\tau_1$ can neither be computed by a* det-RR(1)-*, nor by an* R(1)-*, nor by a* det-mon-nf-R(1)-*transducer.*

**Proposition 3.** *The function $\tau_2$ can neither be computed by an* R(1)- *nor by a* det-mon-nf-R(1)-*transducer.*

**Proposition 4.** *Let $\tau_3$ denote the partial function $\tau_3 : \{a,b\}^* \to \{a,b\}^*$ that is defined by $ab^n \mapsto ba^n$ for all $n \geq 0$. Then $\tau_3$ is computed by a* det-mon-nf-R(1)-*transducer, but it cannot be computed by any* RR(1)-*transducer.*

From these propositions we obtain the hierarchy shown in Figure 1. Here the properness of the inclusions of the deterministic classes in the corresponding nondeterministic classes follows from the simple fact that deterministic transducers can only compute functions, while nondeterministic transducers can also compute relations that are not functions.

$$\begin{tikzcd}
 & & \mathcal{R}el(\mathsf{mon\text{-}nf\text{-}RR(1)\text{-}Td}) & & \\
\mathcal{R}el(\mathsf{det\text{-}mon\text{-}nf\text{-}RR(1)\text{-}Td}) \arrow[urr] & & & & \mathcal{R}el(\mathsf{mon\text{-}nf\text{-}R(1)\text{-}Td}) \arrow[ull] \\
\mathcal{R}el(\mathsf{det\text{-}RR(1)\text{-}Td}) \arrow[u, "\tau_1"] & & \mathcal{R}el(\mathsf{det\text{-}mon\text{-}nf\text{-}R(1)\text{-}Td}) \arrow[ull, "\tau_1"'] \arrow[urr] & & \mathcal{R}el(\mathsf{mon\text{-}R(1)\text{-}Td}) \arrow[u, "\tau_3"'] \\
 & & \mathcal{R}el(\mathsf{det\text{-}R(1)\text{-}Td}) \arrow[ull, "\tau_2"'] \arrow[u, "\tau_3"] \arrow[urr] & &
\end{tikzcd}$$

**Fig. 1.** Hierarchy of classes of transductions computed by various types of restarting transducers. Here arrows denote proper inclusions.

How expressive are the mon-nf-RR(1)-transducers? Obviously, the rational relation $R_1 = \{ (\varepsilon, a^n) \mid n \geq 0 \}$ cannot be computed by any nf-RR(1)-transducer. On the other hand, the relation $R_2 = \{ (a^n b^n, \varepsilon), (a^n b^{n+1}, \varepsilon) \mid n \geq 0 \}$ is not rational, but it is computed by a mon-RR(1)-transducer that is obtained from a mon-RR(1)-automaton for the language $L_2 = \{ a^n b^n, a^n b^{n+1} \mid n \geq 0 \}$. Hence, we obtain the following incomparability result.

**Proposition 5.** *The class of relations* $\mathcal{R}el(\mathsf{mon\text{-}nf\text{-}RR(1)\text{-}Td})$ *is incomparable to the class* RatRel *of rational relations with respect to inclusion.*

In contrast to this incomparability result we will see below that det-mon-nf-RR(1)- and the mon-nf-R(1)-transducers only compute rational relations.

**Lemma 1.** $\mathcal{R}el(\mathsf{det\text{-}mon\text{-}nf\text{-}RR(1)\text{-}Td}) \subsetneq \mathcal{R}el_{\mathrm{io}}(\mathsf{det\text{-}mon\text{-}nf\text{-}RR(1)})$.

*Proof.* Because of the relation $R_1$ above and Corollary 1 it remains to verify that the inclusion above holds. Accordingly, let $T = (Q, \Sigma, \Delta, ¢, \$, q_0, 1, \delta)$ be a det-mon-nf-RR(1)-transducer that computes a relation $\mathrm{Rel}(T) \subseteq \Sigma^* \times \Delta^*$. Thus, a pair $(u, v) \in \Sigma^* \times \Delta^*$ belongs to $\mathrm{Rel}(T)$ if and only if there exists a computation of the form $(q_0 ¢ u \$, \varepsilon) \vdash_T^{c^*} (q_i ¢ u' \$, v') \vdash_T^* (\mathsf{Accept}, v'v'')$ such that $v = v'v''$.

Without loss of generality we can assume that $T$ performs restart and accept instructions only on the \$-symbol. Also we may assume that $\Sigma$ and $\Delta$ are disjoint. We now define a det-mon-nf-RR(1)-automaton $M$ by meta-instructions from a description of $T$ by meta-instructions, where $\mathsf{sh}(L, L')$ denotes the *shuffle* of two languages $L$ and $L'$. Each rewriting meta-instruction of $T$ is translated into finitely many rewriting meta-instructions of $M$. Here in order to increase readability we just consider the case that a single output letter $b$ is produced, but this construction is easily extended to the case of output words of any positive length by using additional rewriting meta-instructions and additional restart states. Let $(q_i, E_1, a \to \varepsilon, E_2 \cdot \$, q_j; b)$ be a rewriting meta-instruction ($a \in \Sigma$, $b \in \Delta$) of $T$. It is translated into the rewriting meta-instructions

$$(q_i, E_1, a \to \varepsilon, \mathsf{sh}(E_2, \Delta^*) \cdot \$, q_i') \text{ and } (q_i', E_1, b \to \varepsilon, \mathsf{sh}(E_2, \Delta^*) \cdot \$, q_j)$$

of $M$, where $q_i'$ is a new state. If a rewriting meta-instruction of $T$ is of the form $(q_i, E_1, a \to \varepsilon, E_2 \cdot \$, q_j; \varepsilon)$, then we simply take the rewriting meta-instruction $(q_i, E_1, a \to \varepsilon, \mathrm{sh}(E_2, \Delta^*) \cdot \$, q_j)$ for $M$. Finally, each accepting meta-instruction $(q_i, E \cdot \$, \mathsf{Accept}; b)$ of $T$ yields an accepting meta-instruction $(q_i, E \cdot b\$, \mathsf{Accept})$ of $M$. Based on this description the transition function of $M$ can be derived from the transition function of $T$.

It remains to show that $\mathrm{Rel}(T) = \mathrm{Rel}_{\mathrm{io}}(M)$ holds. Let $(u, v) \in \mathrm{Rel}(T)$, that is, there exists an accepting computation of $T$ that consumes input $u \in \Sigma^*$ and produces output $v \in \Delta^*$. This computation consists of a sequence of cycles $C_1, C_2, \ldots, C_{m-1}$, where $C_i$ ($1 \le i \le m-1$) is of the form

$$\begin{aligned}(q_i¢x_ia_iy_i\$, v_i) &\vdash^*_{\mathrm{MVR}} (¢x_ip_ia_iy_i\$, v_i) \vdash_{\mathrm{Delete}} (¢x_ip_i'y_i\$, v_i)\\ &\vdash^*_{\mathrm{MVR}} (¢x_iy_i\hat{p}_i\$, v_i) \vdash_{\mathrm{Restart}} (q_{i+1}¢x_iy_i\$, v_ib_i),\end{aligned}$$

and a tail computation of the form

$$(q_m¢w_m\$, v_m) \vdash^*_{\mathrm{MVR}} (¢w_mq_m'\$, v_m) \vdash_{\mathrm{Accept}} (\mathsf{Accept}, v_mb').$$

In the above cycle a rewriting meta-instruction $(q_i, E_1, a_i \to \varepsilon, E_2 \cdot \$, q_{i+1}; b_i)$ is applied, where $¢x_i \in E_1$ and $y_i \in E_2$, and in the above tail computation an accepting meta-instruction $(q_m, E\cdot\$, \mathsf{Accept}; b')$ is applied, where $¢w_m \in E$ holds. Obviously, for all $i = 1, \ldots, m-2$, $x_iy_i = x_{i+1}a_{i+1}y_{i+1}$, and as $T$ is monotone, we see that $|y_i| \ge |y_{i+1}|$ holds.

**Case 1.** If $|y_i| > |y_{i+1}|$, then $a_{i+1}y_{i+1}$ is a suffix of $y_i$. Thus, if we insert the letter $b_i$ immediately to the right of the letter $a_i$, then $M$ will execute the following sequence of two cycles using the meta-instructions that have been obtained from the above meta-instruction of $T$:

$$\begin{aligned}(q_i¢x_ia_ib_iy_i\$) &\vdash^*_{\mathrm{MVR}} (¢x_ip_ia_ib_iy_i\$) \vdash_{\mathrm{Delete}} (¢x_ip_i'b_iy_i\$)\\ &\vdash^*_{\mathrm{MVR}} (¢x_ib_iy_i\hat{p}_i\$) \vdash_{\mathrm{Restart}} (q_i'¢x_ib_iy_i\$)\\ &\vdash^*_{\mathrm{MVR}} (¢x_ip_i''b_iy_i\$) \vdash_{\mathrm{Delete}} (¢x_i\tilde{p}_iy_i\$)\\ &\vdash^*_{\mathrm{MVR}} (¢x_iy_i\hat{p}_i'\$) \vdash_{\mathrm{Restart}} (q_{i+1}¢x_iy_i\$).\end{aligned}$$

**Case 2.** If $|y_i| = |y_{i+1}|$, then $x_i = x_{i+1}a_{i+1}$. In this situation we insert the word $b_ib_{i+1}$ immediately to the right of the factor $a_{i+1}a_i$. Then $M$ will execute the following sequence of two cycles using the meta-instructions that have been obtained from the above meta-instruction of $T$:

$$\begin{aligned}(q_i¢x_ia_ib_ib_{i+1}y_i\$) &\vdash^*_{\mathrm{MVR}} (¢x_ip_ia_ib_ib_{i+1}y_i\$) \vdash_{\mathrm{Delete}} (¢x_ip_i'b_ib_{i+1}y_i\$)\\ &\vdash^*_{\mathrm{MVR}} (¢x_ib_ib_{i+1}y_i\hat{p}_i\$) \vdash_{\mathrm{Restart}} (q_i'¢x_ib_ib_{i+1}y_i\$)\end{aligned}$$

$$\begin{array}{llll} \vdash^*_{\text{MVR}} & (¢x_i p''_i b_i b_{i+1} y_i \$) & \vdash_{\text{Delete}} & (¢x_i \tilde{p}_i b_{i+1} y_i \$) \\ \vdash^*_{\text{MVR}} & (¢x_i b_{i+1} y_i \hat{p}'_i \$) & \vdash_{\text{Restart}} & (q_{i+1} ¢x_i b_{i+1} y_i \$) \\ = & (q_{i+1} ¢x_{i+1} a_{i+1} b_{i+1} y_{i+1} \$). & & \end{array}$$

By combining these two cases we obtain a word $w \in \mathsf{sh}(u, v)$ such that the computation of $M$ on input $w$ mirrors the computation of $T$ on input $u$, and it follows that $(u, v) \in \text{Rel}_{\text{io}}(M)$. Conversely, it can be checked easily that $(x, y) \in \text{Rel}(T)$ holds for each pair $(x, y) \in \text{Rel}_{\text{io}}(M)$. Thus, $\text{Rel}(T) = \text{Rel}_{\text{io}}(M)$ follows. In addition, as $T$ is deterministic and monotone, so is $M$. $\square$

By adjusting the above proof idea also the following inclusion can be derived.

**Lemma 2.** $\mathcal{R}el(\textsf{mon-nf-R(1)-Td}) \subsetneq \mathcal{R}el_{\text{io}}(\textsf{mon-nf-R(1)})$.

Together with Corollary 1 and Proposition 5 these lemmata yield the following results, where RatF denotes the class of *rational functions.*

**Corollary 2.** (a) $\mathcal{R}el(\textsf{det-mon-nf-RR(1)-Td}) \subseteq \textsf{RatF}$.
(b) $\mathcal{R}el(\textsf{mon-nf-R(1)-Td}) \subsetneq \textsf{RatRel}$.
(c) $\mathcal{R}el(\textsf{mon-nf-R(1)-Td}) \subsetneq \mathcal{R}el(\textsf{mon-nf-RR(1)-Td})$.

## 4 Sequential and Rational Functions

Now we want to relate some types of deterministic nf-RR(1)-transducers to certain subclasses of the rational functions. Therefore we describe in short these subclasses. Details can be found in, e.g., [2] and [3].

A *rational transducer* is defined as $T = (Q, \Sigma, \Delta, q_0, F, E)$, where $Q$ is a finite set of internal states, $\Sigma$ is a finite input alphabet, $\Delta$ is a finite output alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $E \subset Q \times \Sigma^* \times \Delta^* \times Q$ is a finite set of transitions. The relation $\text{Rel}(T)$ computed by $T$ consists of all pairs $(u, v) \in \Sigma^* \times \Delta^*$ such that there exists a computation of $T$ that, starting from the initial state $q_0$, reaches a final state $q \in F$ reading the word $u$ and producing the output $v$.

A *sequential transducer* is a rational transducer $T = (Q, \Sigma, \Delta, q_0, Q, E)$ for which $E \subset Q \times \Sigma \times \Delta^* \times Q$ is a partial function from $Q \times \Sigma$ into $\Delta^* \times Q$. Observe that all internal states of a sequential transducer are final, and that in each step it reads a single symbol. Then the relation $\text{Rel}(T)$ is obviously a partial function. It is called a *sequential function*, and by SeqF we denote the class of all sequential functions.

A *subsequential transducer* consists of a pair $T_\varphi = (T, \varphi)$, where $T = (Q, \Sigma, \Delta, q_0, Q, E)$ is a sequential transducer and $\varphi : Q \to \Delta^*$ is a partial function. For $u \in \Sigma^*$, let $q_0 \cdot u \in Q$ denote the state that $T$ reaches from its initial state $q_0$ on reading $u$. Then the relation $\text{Rel}(T_\varphi)$ is defined as

$$\text{Rel}(T_\varphi) = \{ (u, z) \in \Sigma^* \times \Delta^* \mid \exists v \in \Delta^* : (u, v) \in \text{Rel}(T) \text{ and } z = v\varphi(q_0 \cdot u) \}.$$

Obviously, $\mathrm{Rel}(T_\varphi)$ is a partial function. A partial function $R$ is called *subsequential* if there exists a subsequential transducer $T_\varphi$ such that $R$ coincides with the relation $\mathrm{Rel}(T_\varphi)$. By $\mathsf{SubSeqF}$ we denote the class of all subsequential functions.

Finally, a *deterministic generalized sequential machine* ($\mathsf{dgsm}$) is a rational transducer $T = (Q, \Sigma, \Delta, q_0, F, E)$ for which $E$ is a partial function from $Q \times \Sigma$ into $\Delta^* \times Q$ [4]. Thus, it differs from a sequential transducer only in that it has a designated set of final state. The relation $\mathrm{Rel}(T)$ is a partial function, which is called a *dgsm-mapping.* By $\mathsf{DgsmF}$ we denote the class of all dgsm-mappings.

Concerning the various types of rational functions introduced above, we have the chain $\mathsf{SeqF} \subsetneq \mathsf{DgsmF} \subsetneq \mathsf{SubSeqF} \subsetneq \mathsf{RatF}$ of proper inclusions [2,3]. Our first characterization result shows that the dgsm-mappings correspond to a particular class of restarting transducers.

**Theorem 2.** *A function $f : \Sigma^* \to \Delta^*$ is a dgsm-mapping if and only if it can be computed by a proper* $\mathsf{det\text{-}mon\text{-}nf\text{-}R(1)}$*-transducer.*

*Proof.* Let $M = (Q, \Sigma, \Delta, q_0, F, E)$ be a $\mathsf{dgsm}$. We define a $\mathsf{det\text{-}mon\text{-}nf\text{-}R(1)}$-transducer $T = (Q, \Sigma, \Delta, ¢, \$, q_0, 1, \delta)$ such that $\mathrm{Rel}(T) = \mathrm{Rel}(M)$ holds. The transducer $T$ is obtained from $M$ by converting every transition step $(q, x) \to (p, y)$ ($q, p \in Q$, $x \in \Sigma$, and $y \in \Delta^*$) of $M$ into the transition steps $\delta(q, ¢) = (q, \mathsf{MVR})$ and $\delta(q, x) = (p, \mathsf{Restart}, y)$. As $T$ is an $\mathsf{R(1)}$-transducer, its restart operations are combined with delete operations. Thus, $T$ simulates $M$ by erasing its tape inscription letter by letter from left to right, for each letter producing the corresponding output. Finally, $T$ accepts restarting from the restarting configuration $(q¢\$, w)$ producing the empty output if and only if $q$ is a final state of $M$. It follows that $\mathrm{Rel}(T) = \mathrm{Rel}(M)$, and that $T$ is proper, monotone and deterministic.

Conversely let $T$ be a proper $\mathsf{det\text{-}mon\text{-}nf\text{-}R(1)}$-transducer that computes a transduction $t : \Sigma^* \to \Delta^*$. In [5] it is shown that each $\mathsf{det\text{-}mon\text{-}nf\text{-}R(1)}$-automaton can be simulated by a deterministic finite-state acceptor (dfa). During the simulation the dfa has to store a bounded number of possible delete/restart operations of the restarting automaton in its finite-state control in order to verify that it has detected a correct sequence of cycles within the computation being simulated. Now by storing the possible output word together with each delete/restart operation, a dgsm can be designed that simulates the transducer $T$. □

If the given $\mathsf{det\text{-}mon\text{-}nf\text{-}R(1)}$-transducer $T$ is not proper, that is, if it produces non-empty outputs during some of its accept transitions, then the construction mentioned above yields a subsequential transducer. On the other hand, it is easily seen that a subsequential transducer can be simulated by a $\mathsf{det\text{-}mon\text{-}nf\text{-}R(1)}$-transducer that is allowed to produce non-empty outputs during its accept instructions. Thus, we have the following consequence.

**Corollary 3.** *A function $f : \Sigma^* \to \Delta^*$ is a subsequential function if and only if it can be computed by a* $\mathsf{det\text{-}mon\text{-}nf\text{-}R(1)}$*-transducer.*

Finally, we want to characterize the class of rational functions in terms of restarting transducers. To this end we need the following result of Santean.

Here $\mu_\$ : \Sigma^* \to (\Sigma \cup \{\$\})^*$ denotes the function defined by $\mu_\$(\varepsilon) = \varepsilon$ and $\mu_\$(a_1 \dots a_k) = a_1 \dots a_k\$a_2 \dots a_k\$ \dots \$a_{k-1}a_k\$a_k$ for $k \geq 1$ and $a_1, \dots, a_k \in \Sigma$.

**Theorem 3 ([14]).** *If $f : \Sigma^* \to \Delta^*$ is a rational function such that $f(\varepsilon) = \varepsilon$, then there exists a sequential function $f_L : (\Sigma \cup \{\$\})^* \to \Delta^*$ such that $f = f_L \circ \mu_\$$.*

Of course, $\mu_\$$ is not rational, and in fact, it is not even a pushdown function. However, the restarting transducers are somehow naturally equipped to simulate this preprocessing stage.

**Theorem 4.** $\mathsf{RatF} \subseteq \mathcal{R}el(\mathsf{det\text{-}mon\text{-}nf\text{-}RR(1)\text{-}Td})$.

*Proof.* Let $f : \Sigma^* \to \Delta^*$ be a rational function. Let us first assume that $f(\varepsilon) = \varepsilon$ holds. By Theorem 3 there exists a sequential function $f_L : (\Sigma \cup \{\$\})^* \to \Delta^*$ such that $f = f_L \circ \mu_\$$. As the function $f_L$ is sequential, it can be computed by a proper det-mon-nf-RR(1)-transducer $T$ (see Theorem 2).

Now this transducer can be extended to a det-mon-nf-RR(1)-transducer $T_f$ for computing $f$. The sequential transducer for $f_L$ that is given in the proof of Theorem 3 produces a non-empty output only on seeing the \$-symbol. Now $T_f$ proceeds as follows. During the first cycle on input $u = a_1 \dots a_k$, it erases the letter $a_1$ and simulates the internal transitions of the sequential transducer for $f_L$ until it reaches the \$-symbol. At this time it restarts and produces the corresponding output. Now the next cycle starts with the tape content $a_2 \dots a_k$. Continuing in this way $f(u) = f_L \circ \mu_\$(u)$ is computed. Thus, $T_f$ is a proper det-mon-nf-RR(1)-transducer that computes the function $f$.

Finally, if $f(\varepsilon) \neq \varepsilon$, then we apply the construction above to the partial function $f'$ that is defined by $f'(u) = f(u)$ for all $u \in \Sigma^+$ and $f'(\varepsilon) = \varepsilon$. This yields a proper det-mon-nf-RR(1)-transducer $T'_f$ for computing $f'$. We then extend $T'_f$ such that, starting from its initial state, it accepts on empty input producing the output $f(\varepsilon)$. □

$$\begin{tikzcd}
\mathsf{DgsmF} \arrow[r] \arrow[d, equal] & \mathsf{SubSeqF} \arrow[r] \arrow[d, equal] & \mathsf{RatF} \arrow[d, equal] \\
\mathcal{R}el(\mathsf{prop\text{-}det\text{-}mon\text{-}nf\text{-}R(1)\text{-}Td}) & \mathcal{R}el(\mathsf{det\text{-}mon\text{-}nf\text{-}R(1)\text{-}Td}) & \mathcal{R}el(\mathsf{det\text{-}mon\text{-}nf\text{-}RR(1)\text{-}Td})
\end{tikzcd}$$

**Fig. 2.** Classes of rational functions computed by some types of restarting transducers

Together with Corollary 2 (a) this yields the following result.

**Corollary 4.** $\mathsf{RatF} = \mathcal{R}el(\mathsf{det\text{-}mon\text{-}nf\text{-}RR(1)\text{-}Td})$.

Actually the above proof shows that $\mathcal{R}el(\mathsf{prop\text{-}det\text{-}mon\text{-}nf\text{-}RR(1)\text{-}Td})$ coincides with the class of rational functions $f$ satisfying $f(\varepsilon) = \varepsilon$. In summary we have derived the characterizations shown in Figure 2.

## 5 Concluding Remarks

We have introduced various types of restarting transducers, and we have characterized three classes of rational functions through certain types of these transducers. It remains open whether there exists a class of det-nf-RR(1)-transducers that compute exactly the sequential functions. Further, are there characterizations of the classes of functions that are computed by deterministic R(1)- or RR(1)-transducers? For example, the function $\tau_3$ of Proposition 4 is a sequential function, but it cannot be computed by any RR(1)-transducer. On the other hand, the function $\tau_2$ of Example 2 is computed by a proper det-RR(1)-transducer, but it is not even subsequential. Finally, it remains to characterize the classes of binary relations that are computed by the various types of nondeterministic nf-RR(1)-transducers. Recall from Section 3 that the rational relation $R_1 = \{ (\varepsilon, a^n) \mid n \geq 0 \}$ is not computed by any nf-RR(1)-transducer. In fact, it is easily seen that, for any nf-RR(1)-transducer $T$, there exist constants $c_1, c_2 \geq 0$ such that $|v| \leq c_1 \cdot |u| + c_2$ holds for all pairs $(u, v) \in R(T)$.

## References

1. Aho, A., Sethi, R., Ullman, J.: Compilers: Principles and Techniques and Tools. Addison-Weslay, Reading (1986)
2. Berstel, J.: Transductions and Context-Free Languages. Teubner, Stuttgart (1979)
3. Choffrut, C., Culik, K.: Properties of finite and pushdown transducers. SIAM J. Comput. 12, 300–315 (1983)
4. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
5. Hundeshagen, N., Otto, F.: Characterizing the Regular Languages by Nonforgetting Restarting Automata. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 288–299. Springer, Heidelberg (2011)
6. Hundeshagen, N., Otto, F., Vollweiler, M.: Transductions Computed by PC-Systems of Monotone Deterministic Restarting Automata. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 163–172. Springer, Heidelberg (2011)
7. Jančar, P., Mráz, F., Plátek, M., Vogel, J.: Monotonicity of restarting automata. J. Autom. Lang. Comb. 12, 355–371 (2007)
8. Jančar, P., Mráz, F., Plátek, M., Vogel, J.: Restarting Automata. In: Reichel, H. (ed.) FCT 1995. LNCS, vol. 965, pp. 283–292. Springer, Heidelberg (1995)
9. Kutrib, M., Reimann, J.: Succinct description of regular languages by weak restarting automata. Inform. Comput. 206, 1152–1160 (2008)
10. Messerschmidt, H., Otto, F.: A hierarchy of monotone deterministic non-forgetting restarting automata. Theory Comput. Syst. 48, 343–373 (2011)
11. Mohri, M.: Finite-state transducers in language and speech processing. Computational Linguistics 23, 269–311 (1997)
12. Mráz, F.: Lookahead hierarchies of restarting automata. J. Autom. Lang. Comb. 6, 493–506 (2001)
13. Otto, F.: Restarting Automata. In: Ésik, Z., Martin-Vide, C., Mitrana, V. (eds.) Recent Advances in Formal Languages and Applications. SCI, vol. 25, pp. 269–303. Springer, Berlin (2006)
14. Santean, N.: Bimachines and structurally-reversed automata. J. Autom. Lang. Comb. 9, 121–146 (2004)

# Weak Synchronization and Synchronizability of Multitape Pushdown Automata and Turing Machines

Oscar H. Ibarra[1,⋆] and Nicholas Q. Tran[2]

[1] Department of Computer Science
University of California, Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

[2] Department of Mathematics & Computer Science
Santa Clara University, Santa Clara, CA 95053, USA
ntran@math.scu.edu

**Abstract.** Given an $n$-tape automaton $M$ with a one-way read-only head per tape which is delimited by an end marker \$ and a nonnegative integer $k$, we say that $M$ is weakly $k$-synchronized if for every $n$-tuple $x = (x_1, \ldots, x_n)$ that is accepted, there is an accepting computation on $x$ such that no pair of input heads, neither of which is on \$, are more than $k$ tape cells apart at any time during the computation. When a head reaches the marker, it can no longer move. As usual, an $n$-tuple $x = (x_1, \ldots, x_n)$ is accepted if $M$ eventually reaches the configuration where all $n$ heads are on \$ in an accepting state. We look at the following problems: (1) Given an $n$-tape automaton $M$, is it weakly $k$-synchronized for a given $k$ (for some $k$)? and (2) Given an $n$-tape automaton $M$, does there exist a weakly $k$-synchronized automaton for a given $k$ (for some $k$) $M'$ such that $L(M') = L(M)$? In an earlier paper [1], we studied the case of multitape finite automata (NFAs). Here, we investigate the case of multitape pushdown automata (NPDAs), multitape Turing machines, and other multitape models. The results that we obtain contrast those of the earlier results and involve some rather intricate constructions.

**Keywords:** Multitape NPDAs, Weakly Synchronized, Reversal-bounded Counters, Multitape Turing Machines, (Un)decidability, Halting Problem, Post Correspondence Problem.

## 1 Introduction

A serious web security vulnerability can occur when a user input string is embedded into an interpreted script which is then executed with system privileges. Such a carefully crafted embedded input string can alter the intended meaning of the script and bypass security checks, such as in the case of SQL injection attacks. A current research effort to combat this problem involves reachability analysis of string variables in a script that allows general assertions to be made

⋆ Supported in part by NSF Grants CCF-1143892 and CCF-1117708.

about their values (e.g., a string variable will never contain a specific set of characters). Earlier reachability analyses model each string variable using a separate finite-state automaton. More recently, multitrack and then multitape finite-state automata have been used to model sets of string variables, allowing assertions about relationships between the string members; see [6,7,8] for concrete examples of reachability analyses using finite-state automata.

Although multitape automata support a richer class of assertions than multitrack automata, decision problems involving them are often undecidable. (Here we are assuming the usual convention that strings in an input tuple to multitrack automata are left-justified, and the shorter strings are padded to the right with the symbol $\lambda$ to have the same length as the longest string; on the other hand, if $\lambda$'s are allowed to appear anywhere in the strings, then multitrack and multitape automata are equivalent.) Clearly, multitape automata can be easily converted to equivalent multitrack automata if the distance between any two tape heads can be bounded by a constant $k$; we say such machines are $k$-synchronized or simply synchronized if the constant is not important or not known. This observation gives rise to three decisions problems:

- given a multitape automaton $M$ and a constant $k$, is $M$ $k$-synchronized?
- given a multitape automaton $M$, is $M$ synchronized?
- given a multitape automaton $M$, is there an automaton $M'$ of the same type such that $L(M) = L(M')$ and $M'$ is synchronized?

We call the property involved in the first two questions **synchronization** and the property in the last question **synchronizability**. We have studied these questions in two earlier papers:

1. In [4], we studied the notion of **strong head synchronization**: an $n$-tape automaton $M$ is *strongly $k$-synchronized* if at any time during the computation on *any* input $n$-tuple $(x_1, \ldots, x_n)$ (accepted or not), no pair of input heads, neither of which is on \$, are more than $k$ tape cells apart. In that paper, we showed the following:
   (**) It is decidable to determine, given an $n$-tape NPDA $M$, whether it is strongly $k$-synchronized for some $k$, and if this is the case, the smallest such $k$ can be found.
2. In [1], we studied the notion of **weak head synchronization for multitape finite automata**: an $n$-tape automaton $M$ is *weakly $k$-synchronized* if for any *accepted* input $n$-tuple $(x_1, \ldots, x_n)$ there exists some accepting computation of $M$ such that at any time no pair of input heads, neither of which is on \$, are more than $k$ tape cells apart.

   Weak synchronization is a more general notion than strong synchronization. Obviously, a strongly synchronized machine is also weakly synchronized, but the converse is not true. Consider, e.g., the set $L = \{(a^m\$, b^n\$) \mid m, n > 0\}$. We can construct a 2-tape NFA $M$, which when given input $(a^m\$, b^n\$)$, nondeterministically executes (a) or (b) below:
   (a) $M$ reads $a^m\$$ on tape 1 until head 1 reaches \$, and then reads $b^n\$$ on tape 2 until head 2 reaches \$ and then accepts.

(b) $M$ reads the symbols on the two tapes simultaneously until one head reaches \$. Then the other head scans the remaining symbols on its tape and accepts.

Then $M$ is not strongly synchronized, because of (a). However, $M$ is weakly synchronized (in fact, weakly 0-synchronized) because every tuple ($a^m\$$, $b^n\$$) can be accepted in a computation as described in (b). Thus strongly synchronized implies weakly synchronized, but not conversely.

It turns out that weak synchronization and synchronizability decision problems are more difficult and subtle than their strong counterparts. For example, it was shown in [1] that, unlike (**) above, it is undecidable to determine, given a 2-ambiguous 2-tape NFA, whether it is weakly $k$-synchronized. However, the problem is decidable if $M$ is 1-ambiguous (i.e., unambiguous).

In the present paper we study the notion of **weak head synchronization and synchronizability for nondeterministic multitape automata equipped with a pushdown stack (NPDA) or a counter (NCM) and possibly augmented with additional reversal-bounded counters, as well as multitape space-bounded Turing machines.** We obtain contrasting decidability results for restricted versions of these machines in terms of ambiguity and boundedness of one or more strings in the input tuples.

**Note:** Some proofs are omitted due to lack of space. All proofs will be given in a full version of the paper.

## 2 Preliminaries

A (one-way) $n$-tape deterministic finite automaton (DFA) $M$ is a finite automaton with $n$ tapes where the content of each tape is a string over input alphabet $\Sigma$. Each tape is read-only and has an associated one-way input head. We assume that each tape has a right end marker \$ (not in $\Sigma$). On a given $n$-tuple input $x = (x_1, \ldots, x_n)$, $M$ starts in the initial state $q_0$ with all the heads on the first symbols of their respective tapes. The transition function of $M$ consists of rules of the form $\delta(q, a_1, \ldots, a_n) = (p, d_1, \ldots, d_n)$ (resp. $= \varnothing$). Such a rule means that if $M$ is in state $q$, with head $H_i$ on symbol $a_i$, then the machine moves $H_i$ in direction 1 or 0 (for right move or stationary move), and enters state $p$ (resp., halts). When a head reaches the end marker \$, that head has to remain on the end marker. The input $x$ is accepted if $M$ eventually reaches the configuration where all $n$ heads are on \$ in an accepting state. Let $M$ be an $n$-tape DFA and $k \geq 0$. $M$ is *weakly $k$-synchronized* if for every $n$-tuple $x = (x_1, \ldots, x_n)$ that is accepted, the unique computation on $x$ is such that at any time during the computation, no pair of input heads, neither of which is on \$, are more than $k$ cells apart. Notice that, since the condition in the definition concerns pairs of heads that are both on symbols in $\Sigma$, if one of these two heads is on \$, then we can stipulate that the condition is automatically satisfied, irrespective of the distance between the heads. In particular, if $k = 0$, then all heads move to the right synchronously at the same time (except for heads that reach the right end marker early). $M$ is *weakly synchronized* if it is weakly $k$-synchronized for some $k$.

The above definitions generalize to $n$-tape nondeterministic finite automata (NFAs). Now, *weakly $k$-synchronized* requires that every $n$-tuple $x = (x_1, \ldots, x_n)$ that is accepted has a computation on $x$ such that at any time during the computation, no pair of input heads, neither of which is on \$, are more than $k$ cells apart. The definitions can also be generalized to $n$-tape deterministic pushdown automata (DPDAs) and $n$-tape nondeterministic pushdown automata (NPDAs), which may even be augmented with a finite number of reversal-bounded counters. At each step, each counter (which is initially set to zero) can be incremented by 1, decremented by 1, or left unchanged and can be tested for zero. The counters are reversal-bounded in the sense that there is a specified $r$ such that during any computation, no counter can change mode from increasing to decreasing and vice-versa more than $r$ of times. A counter is 1-reversal if once it decrements, it can no longer increment. Clearly, an $r$-reversal counter can be simulated by $\lceil (r+1)/2 \rceil$ 1-reversal counters.

A nondeterministic counter machine (NCM) is an NFA augmented with one counter. Note that an NCM is a special case of an NPDA where the stack alphabet consists of only one symbol, in addition to a distinguished bottom-of-the-stack symbol $B$ which is never modified. Hence the stack can be thought of as a counter since it can only push or pop the same symbol, which would correspond to incrementing or decrementing the stack height by 1. The count is zero when the stack contains only the bottom symbol $B$. DCM is the deterministic version of NCM.

Given an $n$-tuple $(x_1, \ldots, x_n)$, denote by $AL(x_1, \ldots, x_n)$ an $n$-track string where the symbols of $x_i$'s are left-justified (i.e., the symbols are aligned) and the shorter strings are right-filled with blanks ($\lambda$) to make all tracks the same length. For example, $AL(01, 1111, 101)$ has $01\lambda\lambda$ on the upper track, 1111 on the middle track, and $101\lambda$ on the lower track. Given a set $L$ of $n$-tuples, define $AL(L) = \{AL(x) \mid x \in L\}$.

A machine is *$k$-ambiguous* if there are at most $k$ accepting computations for any input. Note that unambiguous is the same as 1-ambiguous, and deterministic is a special case of unambiguous.

A language is bounded if it is a subset of $a_1^* \cdots a_n^*$ for some distinct letters (symbols) $a_1, \ldots, a_n$. A multitape machine is *unary* if each tape contains a string over a single symbol (letter); *bounded* if each tape contains a string from a bounded language; and *all-but-one-bounded (ABO-bounded)* if all but the first tape contains a string from a bounded language. We also refer to the inputs of such machines as unary, bounded, ABO-bounded, respectively.

The following lemma can be easily verified.

**Lemma 1.** *Let $L$ be a set of $n$-tuples.*

1. *$L$ is accepted by a weakly 0-synchronized $n$-tape NFA if and only if $AL(L)$ is regular.*
2. *$L$ is accepted by a weakly 0-synchronized $n$-tape NPDA if and only if $AL(L)$ is context-free.*

Let $\mathbb{N}$ be the set of nonnegative integers and $k$ be a positive integer. A subset $Q$ of $\mathbb{N}^k$ is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_n$ in $\mathbb{N}^k$ such that $Q = \{v_0 + t_1 v_1 + \cdots + t_n v_n \mid t_1, \ldots, t_n \in \mathbb{N}\}$. The vectors $v_0$ (referred to as the *constant vector*) and $v_1, \ldots, v_n$ (referred to as the *periods*) are called the *generators* of the linear set $Q$. The set $Q \subseteq \mathbb{N}^k$ is *semilinear* if it is a finite union of linear sets. The empty set is a trivial (semi)linear set, where the set of generators is empty. Every finite subset of $\mathbb{N}^k$ is semilinear – it is a finite union of linear sets whose generators are constant vectors. Semilinear sets are closed under (finite) union, complementation and intersection. It is known that the disjointness, containment, and equivalence problems for semilinear sets are decidable [2].

Let $\Sigma = \{a_1, \ldots, a_k\}$. For $w \in \Sigma^*$, let $|w|$ is the number of letters in $w$, and $|w|_{a_i}$ denote the number of occurrences of $a_i$ in $w$. The *Parikh image* $P(w)$ of $w$ is the vector $(|w|_{a_1}, \ldots, |w|_{a_k})$; similarly, the Parikh image of a language $L$ is defined as $P(L) = \{P(w) \mid w \in L\}$.

It is known that the Parikh image of a language $L$ accepted by an NPDA (i.e., $L$ is context-free) is an effectively computable semilinear set [5]. This was generalized in [3]:

**Theorem 2.** *The Parikh image of a language $L$ accepted by an NPDA with 1-reversal counters is an effectively computable semilinear set.*

We will need the following result from [3]:

**Theorem 3.** *The emptiness (Is $L(M) = \varnothing$?) and infiniteness (Is $L(M)$ infinite?) problems for 1-tape NPDAs with reversal-bounded counters are decidable.*

**Convention:** (1) We shall also refer to a set of $n$-tuples accepted by an $n$-tape machine as a language. (2) All input $n$-tuples $(x_1, \ldots, x_n)$ are delimited by a right end marker \$ on each tape, although sometimes the end markers are not explicitly shown.

## 3 Weak Synchronization of Multitape Automata

In this section, we study the weak synchronization decision problem for multitape NCM/NPDA with/without additional reversal-bounded counters. We obtain sharp contrasting results by considering further restrictions on ambiguity (unambiguous, finitely ambiguous, infinitely ambiguous) and boundedness of input strings (bounded, ABO-bounded, unbounded). The results will be organized according to their proof techniques, which fall under three categories: simulation by 1NPDAs augmented by reversal-bounded counters, reduction from Post Correspondence Problem, and reduction from halting problems of Turing and 2-counter machines. (Note that 1NPDA is an abbreviation for 1-tape NPDA.)

## 3.1 Technique 1: Simulation by 1NPDAs with Reversal-Bounded Counters

We will see that it is decidable, given an unambiguous multitape NPDA $M$ with reversal-bounded counters, whether $M$ is weakly $k$-synchronized for a given $k$ (resp., for some $k$). We will also show that if $M$ is ambiguous but its tapes are over bounded languages, it is decidable whether it is weakly $k$-synchronized for a given $k$.

### Unambiguous Multitape NPDAs

Let $M$ be an $n$-tape NPDA $M$ augmented with reversal-bounded counters. $M$ may be ambiguous. Define $L = \{1^d \mid d \geq 0$, for some computation of $M$ on some input $n$-tuple $x = (x_1, \ldots, x_n)$ (accepting or not) there are two heads that get separated more than $d$ cells apart during the computation$\}$.

For $k \geq 0$, define $L_k = \{1^k \mid$ for some computation of $M$ on some input $n$-tuple $x = (x_1, \ldots, x_n)$ (accepting or not) there are two heads that get separated more than $k$ cells apart during the computation$\}$.

In [4], it was shown that we can construct a 1-tape NPDA $M'$ (resp., $M_k$) augmented with reversal bounded counters which accepts $L$ (resp., $L_k$). Then $M$ is not strongly $k$-synchronized for some $k$ (resp., for a given $k$) if and only if $L$ is infinite (resp., $L_k \neq \varnothing$), which is decidable by Theorem 3. When $M$ is unambiguous, similar constructions work for the "weakly" version by modifying $M'$ (resp., $M_k$) so that it accepts $1^d$ (resp., $1^k$) if during the unique accepting computation of $M$ on some input $n$-tuple $x$ that is accepted, there are two heads that get separated more than $d$ (resp., $k$) cells apart. Hence, we have:

**Theorem 4.** *It is decidable to determine, given an unambiguous $n$-tape NPDA $M$ augmented with reversal-bounded counters, whether it is weakly $k$-synchronized for some $k$ (resp., for a given $k$).*

### Bounded-Input Multitape NPDAs

When the multitape NPDA is ambiguous but it is over bounded languages, we have:

**Theorem 5.** *It is decidable to determine, given an integer $k \geq 0$ and an $n$-tape NPDA $M$ over $a_{11}^* \cdots a_{1m_1}^* \times \cdots \times a_{n1}^* \cdots a_{nm_n}^*$ where the $a_{ij}$'s are distinct symbols, whether $M$ is weakly $k$-synchronized.*

*Proof.* We prove the theorem for the case of 2 tapes. The same technique works for any number of tapes. Let $M$ be a 2-tape NPDA, where the string on tape 1 is from the bounded language $a_1^* \cdots a_r^*$ and the string on tape 2 is from $b_1^* \cdots b_s^*$, for some distinct symbols $a_1, \ldots, a_r, b_1, \ldots, b_s$.

Construct a (1-tape) NPDA $M_1$ with $r + s$ 1-reversal counters $C_1, \ldots, C_r$, $D_1, \ldots, D_s$ that accepts a language that is a subset of $a_1^* \cdots a_r^* b_1^* \cdots b_s^*$ such that

$L(M_1) = \{a_1^{i_1} \cdots a_r^{i_r} b_1^{j_1} \cdots b_s^{j_s} \mid (a_1^{i_1} \cdots a_r^{i_r}, b_1^{j_1} \cdots b_s^{j_s}) \in L(M)\}$. $M_1$ operates as follows when given the input $a_1^{i_1} \cdots a_r^{i_r} b_1^{j_1} \cdots b_s^{j_s}$: It reads the input and stores $i_1, \ldots, i_r$, $j_1, \ldots, j_s$ in counters $C_1, \ldots, C_r, D_1, \ldots, D_s$. Then $M_1$ simulates the computation of $M$ on $(a_1^{i_1} \cdots a_r^{i_r}, b_1^{j_1} \cdots b_s^{j_s})$, using the integers stored in the counters. $M$ accepts if and only if $M_1$ accepts. It is known that the Parikh image of the language accepted by a 1-tape NPDA with 1-reversal counters ($M_1$ in this case) is an effectively computable semilinear set, i.e.,

$$P(L(M_1)) = \{(i_1, \ldots, i_r, j_1, \ldots, j_s) \mid a_1^{i_1} \cdots a_r^{i_r} b_1^{j_1} \cdots b_s^{j_s} \in L(M_1)\}$$

is a semilinear set [3].

Next, we construct from $M$ a 2-tape NPDA $M'$ which simulates $M$ faithfully except that it halts and rejects whenever the heads of $M$ are no longer within $k$ cells apart (when neither head is on \$). Clearly $L(M') \subseteq L(M)$, and $M$ is weakly 0-synchronized iff $L(M) = L(M')$. To decide this condition, we construct from $M'$ (as above), an NPDA with 1-reversal counters $M_1'$ such that $L(M_1') = \{a_1^{i_1} \cdots a_r^{i_r} b_1^{j_1} \cdots b_s^{j_s} \mid (a_1^{i_1} \cdots a_r^{i_r} b_1^{j_1} \cdots b_s^{j_s}) \in L(M_1')\}$. Then $P(L(M_1'))$ is also semilinear. Clearly, $L(M) = L(M')$ iff $L(M_1) = L(M_1')$. The result follows, since the equivalence of semilinear sets is decidable [2]. □

The above result generalizes to more generally bounded languages and machines augmented with reversal-bounded counters:

**Theorem 6.** *It is decidable to determine, given an $n$-tape NPDA $M$ augmented with reversal-bounded counters over $x_{11}^* \cdots x_{1m_1}^* \times \cdots \times x_{n1}^* \cdots x_{nm_n}^*$ for some (not necessarily distinct) nonnull strings $x_{ij}$'s and an integer $k \geq 0$, whether $M$ is weakly $k$-synchronized.*

## 3.2 Technique 2: Reduction from Post Correspondence Problem

In [1], it was shown that it is undecidable to determine, given a 2-ambiguous 2-tape NFA $M$, whether $M$ is weakly $k$-synchronized for a given $k$ (resp., for some $k$), and whether there is a weakly 0-synchronized 2-tape NFA $M'$ such that $L(M') = L(M)$. We extend this result to 2-ambiguous 2-tape 1-reversal NCMs and 2-ambiguous 2-tape 3-reversal NPDAs. Recall that an NCM is a special case of an NPDA (it has a counter instead of a stack). These results are obtained by reduction from the undecidable problem of Post Correspondence Problem (PCP).

An instance $I = (u_1, \ldots, u_n); (v_1, \ldots, v_n)$ of the PCP is a pair of $n$-tuples of nonnull strings over an alphabet with at least two symbols. A solution to $I$ is a sequence of indices $i_1, i_2, \ldots, i_m$ such that $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$. It is well known that it is undecidable to determine, given a PCP instance $I$, whether it has a solution. We can define $W(I) = \{x \mid x = u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}, m \geq 1, 1 \leq i_1, \ldots, i_m \leq n\}$. Then $I$ has a solution if and only if $W(I) \neq \varnothing$. We shall also refer to a string $x$ in $W(I)$ as a solution to $I$.

## 2-Ambiguous 2-Tape NPDAs

**Theorem 7.** *The following problems are undecidable, given a 2-ambiguous 2-tape 1-reversal NCM $M$:*

1. *Is $M$ weakly $k$-synchronized for a given $k$?*
2. *Is $M$ weakly $k$-synchronized for some $k$?*
3. *Is there a 2-tape NCM (or NPDA) $M'$ that is weakly 0-synchronized (or weakly $k$-synchronized for a given $k$, or weakly $k$-synchronized for some $k$) such that $L(M') = L(M)$?*

*Proof.* Synchronization (the first two items) is implied by Theorem 2 from [1]. It remains to show synchronizability (the last item). We first prove the result for a general (i.e., unboundedly ambiguous) 2-tape 1-reversal NCM $M$.

Let $I = (u_1, \ldots, u_n); (v_1, \ldots, v_n)$ be an instance of the PCP, where the $u_i$'s and the $v_i$'s are nonnull strings in $\{0, 1\}^*$. Let $a, b, c$ be new symbols. Define the language (of tuples):

$$L = \{(xa^r b^s, yc^i) \mid r, s, i > 0, x \neq y\} \cup$$
$$\{(xa^{3i} b^{2i}, xc^i) \mid i > 0, x \text{ is a solution to } I\}.$$

$L$ clearly can be accepted by a 2-tape 1-reversal NCM $M$ which, when given input $(xa^r b^s, yc^i)$, nondeterministically accepts if one of the following holds:

(a) $x \neq y$: this step can be performed deterministically with the two heads in 0-sync.
(b) $x = u_{i_1} \ldots u_{i_k}$ and $y = v_{i_1} \ldots v_{i_k}$ for some (guessed) index sequence $i_1, \ldots, i_k$, $r = 3i$ and $s = 2i$: after verifying that $x = y$, $M$ verifies that $r = 3i$ while simultaneously storing $i$ in the counter and then verifies that $s = 2i$, reversing the counter only once in the process.

If $I$ has no solutions, then all accepting computations are of type (a) and hence $M$ is weakly 0-synchronized. We show that if $I$ has a solution then no 2-tape NCM (or NPDA) $M$ accepting $L$ can be weakly 0-synchronized (weakly $k$-synchronized for a given $k$, or weakly $k$-synchronized for some $k$).

Suppose such an $M$ exists, so $AL(L)$ is context-free by Lemma 1. Let

$$t = (xa^{3m} b^{2m}, xc^m)$$

be the tuple in $L$ where $x$ is a solution to the PCP instance $I$ and $m$ is the Ogden's pumping lemma constant for $L$.

We mark the $m$ symbols $AL(a, a)$ in $AL(t)$. According to Ogden's lemma, $AL(t)$ can be written as $UVXYZ$ where $X$ has at least one marked position; either $U$ and $V$ both have marked positions, or $Y$ and $Z$ both have marked positions; $VXY$ has at most $m$ marked positions; and $UV^kXY^kZ$ is in $L$ for every $k \geq 0$.

We have five cases:

1. $Y$ and $Z$ both have marked positions: this means that $V$ and $Y$ do not contain $b$'s in their upper tracks, and $Y$ has at least one $AL(a, a)$. Hence $UVVXYYZ$ does not have the correct number of $b$'s on its upper track, and cannot be in $L$, a contradiction.
2. $U$ and $V$ both have marked positions: this means that $|V| \geq 1$, $V$ consists solely of $AL(a, a)$, and $Y$ either is empty or consists solely of $AL(a, a)$, $AL(a, \lambda)$, or $AL(b, \lambda)$.
   (a) $Y$ is empty: same as case 1.
   (b) $Y$ consists solely of $AL(a, a)$: same as case 1.
   (c) $Y$ consists solely of $AL(a, \lambda)$: same as case 1.
   (d) $Y$ consists solely of $AL(b, \lambda)$: the strings $UV^{k+1}XY^{k+1}Z$ are of the form

$$AL(xa^{3m+k|V|}b^{2m+k|Y|}, xc^{m+k|V|}),$$

   where $3m + k|V| = 3(m + k|V|)$, which implies that $|V| = 0$, a contradiction.

It follows the PCP instance $I$ does not have a solution if and only if $L(M) = L(M')$ for some weakly 0-synchronized 2-tape NCM (or NPDA) $M'$. This shows the undecidability of part 3.

We now modify the construction of the 2-tape NCM $M$ above to make it 2-ambiguous. The sources of ambiguity are in computations of type (b). Clearly, since computations of type (a) is deterministic, if we can make computations of type (b) deterministic, then the 2-NCM will be 2-ambiguous.

We accomplish this as follows. Instead of $x$, we use $x'$ where $x'$ has two tracks: track 1 contains $x$ and track 2 contains the "encoding" of the indices that are used to match $x$ and $y$; $y$ remains single-track. Specifically, let $I = (u_1, \ldots, u_n); (v_1, \ldots, v_n)$ be an instance of the PCP. Let $\#, e_1, \ldots, e_n$ be new symbols. For $1 \leq i \leq n$, let the string $E(i) = e_i\#^{|u_i|-1}$. Thus, the length of $E(i)$ is equal to the length of $u_i$. Let $\Delta = \{\#, e_1, \ldots, e_n\}$ and define the language:

$$\begin{aligned}L = \{&(x'a^rb^s, yc^i) \mid r, s, i > 0, x' \text{ is a 2-track tape where the first track contains}\\ &x \text{ and the second track is a string in } \Delta^*, x \neq y\} \cup\\ &\{x'a^{3i}b^{2i}, yc^i) \mid i > 0, x' \text{ is a 2-track tape where the first track contains } x\\ &\text{and the second track is a string } E(i_1) \cdots E(i_r) \text{ for some}\\ &i_1, \ldots, i_r, x = y, x = u_{i_1} \cdots u_{i_r}, y = v_{i_1} \cdots v_{i_r}, j = 2i\}.\end{aligned}$$

One can easily check that the computations of types (a) and (b) can be made deterministic. However, it is possible that the same input of the form $(x'a^rb^s, yc^i)$, where $x \neq y$ can be accepted in both computations of type (a) or (b). ($x$ is the first track of $x'$.) Hence, $M$ is 2-ambiguous. □

### 2-Ambiguous ABO-Bounded-Input 2-Tape NPDAs

In Theorem 7, the tapes of $M$ have unrestricted inputs. We now state a somewhat stronger variation in that one tape is restricted to unary inputs, but $M$ is now a 2-ambiguous 2-tape 3-reversal NPDA; its proof also involves a reduction from PCP. (The stack of an NPDA is $r$-reversal if the number of times it changes mode from pushing to popping and vice-versa during any computation is at most $r$ times.)

**Theorem 8.** *The following problems are undecidable, given a 2-ambiguous 2-tape 3-reversal NPDA $M$ over $\Sigma^* \times c^*$, where $\Sigma$ is an alphabet with at least 2 symbols:*

1. *Is $M$ weakly $k$-synchronized for a given $k$?*
2. *Is $M$ weakly $k$-synchronized for some $k$?*
3. *Is there a 2-tape NPDA $M'$ that is weakly 0-synchronized (weakly $k$-synchronized for a given $k$, or weakly $k$-synchronized for some $k$) such that $L(M') = L(M)$?*

## 3.3 Technique 3: Reduction from Halting Problems

In this subsection we present two additional improvements of Theorem 7. First, we strengthen Theorem 7 by showing that the decision problems for 2-tape 1-reversal NCM remains undecidable even when one tape is over a unary alphabet, although the machine is now allowed to have unbounded ambiguity (Theorem 9). Second, we prove another improvement of Theorem 7 that allows all but one input tape to be bounded at the cost of unbounded reversals of the counter (Theorem 11). Both sets of results are proved by reduction from the undecidable halting problems for Turing and counter machines.

### Ambiguous ABO-Bounded-Input 2-Tape NCMs

**Theorem 9.** *The following problems are undecidable, given a 2-tape 1-reversal NCM $M$ over input tuples from $\Sigma^* \times c^*$, where $\Sigma$ has at least 2 symbols and $c$ is a symbol:*

1. *Is $M$ weakly $k$-synchronized for a given $k$ ?*
2. *Is $M$ weakly $k$-synchronized for some $k$ ?*
3. *Is there a 2-tape NCM (or NPDA) $M'$ that is weakly 0-synchronized (or weakly $k$-synchronized for a given $k$, or weakly $k$-synchronized for some $k$) such that $L(M') = L(M)$? Is there a 2-tape 1-reversal weakly 0-synchronized NPDA $M'$ such that $L(M) = L(M')$ ?*

*Proof.* By reduction from the halting problem for Turing machines. Let $T$ be an arbitrary Turing machine. The (unique) halting computation of $T$ on blank input, if it exists, can be described by a sequence of instantaneous descriptions $H(T) = I_1\#I_2\#\ldots\#I_m$, where $I_1$ is the initial instantaneous description of $T$,

$I_m$ is a halting instantaneous description of $T$, and $I_{j+1}$ follows from $I_j$ in one step for $j = 1, 2, \ldots, m-1$. Let $\Sigma$ be set of symbols that can occur in $H(T)$ and $a, b, c$ be new symbols. Define

$$L_T = \{(xa^r b^s, c^{|x|+i}) \mid x \in \Sigma, r, s, i > 0, x \neq H(T) \text{ or } (r = 3i, s = 2i)\}.$$

$L(T)$ can be accepted by a 2-tape 1-reversal NCM $M$ as follows: on input $(xa^r b^s, c^t)$, $M$ nondeterministically verifies one of the following two possibilities:

1. $x \neq H(T)$: either $x$ is not in the well-formed regular set, or $I_1$ is not the initial ID, or $I_m$ is not a halting ID, or $I_{j+1}$ does not follow from $I_j$ in one step for some $j$. The first three conditions can be checked with a DFA. The last condition can be checked by guessing $j$ and guessing the location where $I_j$ and $I_{j+1}$ do not agree using one reversal of the counter. Since the number of candidates for $j$ depends on $m$, $M$ is not finitely ambiguous. Regardless of the guessed values, $M$'s heads move in 0-sync in this process.
2. $r = 3i$ and $s = 2i$, where $i = t - |x|$: $M$ first verifies that $t \geq |x|$ by moving its two heads in 0-sync until the first $a$ is read, and then verifies that $r = 3i$ by moving the first head three times as fast. At the same time, $M$ loads the counter with $i$. When $M$ has finished reading the second tape, it then verifies that $s = 2i$ using one reversal of the counter. Note that since $i$ is arbitrary, the two heads of $M$ are separated by an arbitrary distance in this process.

If $T$ does not halt on blank input, then any input that is accepted via a computation of type (2) above is also accepted by a computation of type (1), and hence $M$ is weakly 0-synchronized.

On the other hand, if $T$ halts on blank input, then there is a (unique) accepting computation $x$ of $T$. For any value of $i$, $(xa^{3i}b^{2i}, c^{|x|+i})$ is accepted via a computation of type (2) of $M$ only. Now if $L_T$ is accepted by some weakly 0-synchronized 2-tape 1-reversal NCM $M'$, then $AL(L_T)$ must be context-free by Lemma 1, which is a contradiction since for large values of $i$, $(xa^{3i}b^{2i}, c^{|x|+i})$ can be pumped using Ogden's Lemma to get a string not in $L_T$ as shown in the proof of Theorem 7.

We have shown that $M$ is weakly 0-synchronized, in fact, weakly synchronized iff $L(M) = L(M')$ for some weakly 0-synchronized $M'$ iff $T$ halts on blank input. □

In contrast to the above theorem, the following result was shown in [1]:

**Theorem 10.** *It is decidable to determine, given an $n$-tape NFA $M$ over $\Sigma^* \times x_{21}^* \cdots x_{2m_2}^* \times \cdots \times x_{n1}^* \cdots x_{nm_n}^*$ for some (not necessarily distinct) nonnull strings $x_{ij}$'s and a nonnegative integer $k$, whether $M$ is weakly $k$-synchronized.*

## 3-Ambiguous ABO-Bounded-Input Unbounded-Reversal 2-Tape NCMs

The following theorem can be proved using reduction from the halting problem of 2-counter machines:

**Theorem 11.** *The following problems are undecidable, given a 3-ambiguous 2-tape NCM $M$ over input tuples from $\Sigma^* \times c^*$, where $\Sigma$ has at least 2 symbols and $c$ is a symbol:*

1. *Is $M$ weakly $k$-synchronized for a given $k$ ?*
2. *Is $M$ weakly $k$-synchronized for some $k$ ?*
3. *Is there a 2-tape NCM (or NPDA) $M'$ that is weakly 0-synchronized (or weakly $k$-synchronized for a given $k$, or weakly $k$-synchronized for some $k$) such that $L(M') = L(M)$?*

**Open:** Note that in Theorem 11, the counter of the NCM is unrestricted. Can the theorem be further strengthen so that the NCM is $k$-ambiguous for some $k$ and the counter is $r$-reversal for some $r$?

## 4 Space-Bounded Multitape Turing Machines

In this subsection, we investigate the weak synchronization problems for Turing machines.

**Theorem 12.** *The following problems are undecidable, given a 2-tape DTM $M$ with logarithmic work space over input tuples from $a^* \times b^*$, where $a$ and $b$ are symbols:*

1. *Is $M$ weakly $k$-synchronized for a given $k$ ?*
2. *Is $M$ weakly $k$-synchronized for some $k$ ?*

*Proof.* By reduction from the halting Problem of Turing machines. Let $T$ be any one-tape Turing machine. The halting computation $H(T) = I_0\#I_1 \ldots \#I_r$ of $T$ on blank input, if it exists, can be encoded and decoded using only two symbols and logarithmic space. Define

$$L_T = \{(a^i, b) : i > 0, \mid i = 2^n m, m \text{ odd, binary}(n) = H(T)\}.$$

Clearly there is a 2-tape DTM with logarithmic work space to accept $L$: on input $(a^i, b)$, $M$ first computes the binary representation of $i$ on its work tape, and then computes the binary representation of $n$, where $i = 2^n m$, $m$ is odd. $M$ then verifies that the binary representation of $n$ is the encoding of the halting computation of $T$ on blank input: $H(T) = I_0\#I_1\# \ldots \#I_r$. It then moves the second head until it reads the right marker \$ and then accepts. Otherwise $M$ rejects. This can be done deterministically in logarithmic space using only the first head.

If $T$ does not halt on blank input, then $L_T = \varnothing$, and hence $M$ is vacuously weakly 0-synchronized. Else if $T$ halts on blank input, then $L_T$ is infinite; since there is only one $b$ and arbitrarily many $a$'s in accepted inputs, $M$ cannot be weakly 0-synchronized for any $k$. Hence we have shown that $T$ halts on blank input iff $M$ is weakly 0-synchronized iff $M$ is weakly $k$-synchronized for any $k$.

Unlike previous results of this type, there is a weakly 0-synchronized 2-tape DFA $M'$ with logarithmic work space to accept $L_T$: instead of moving the second head after verifying that the first tape encodes the halting computation of $T$, $M'$ verifies that the second tape contains $b$ by moving the second head in 0-sync with the first head. We will see that machines of this type are always synchronizable (Theorem 13, part 3). □

**Theorem 13.**

1. *Let $S(n) \in o(n)$. It is undecidable to determine, given a 2-ambiguous 2-tape NFA $M$, whether there exists a 2-tape $S(n)$ space-bounded NTM $M'$ that is weakly 0-synchronized (or weakly $k$-synchronized for a given $k$, or weakly $k$-synchronized for some $k$) such that $L(M') = L(M)$.*
2. *Let $S(n) \in \Omega(n)$. Then any multitape $S(n)$ space-bounded NTM can be converted to an equivalent weakly 0-synchronized multitape $S(n)$ space-bounded $M'$.*
3. *Let $S(n) \in \Omega(\log\ n)$. Then any multitape $S(n)$ space-bounded NTM over ABO-bounded languages can be converted to an equivalent weakly 0-synchronized multitape $S(n)$ space-bounded $M'$.*

## 5 Summary

We investigated the boundaries between decidability and undecidability of whether a pushdown automaton (resp., counter machine, Turing machine) with certain restrictions is $k$-synchronized for a given $k$ or for some $k$ (synchronization).

The main result is synchronization is undecidable for 2-tape 2-ambiguous, 1-reversal NCMs (Thm. 7). It becomes decidable when the NCMs are either unambiguous (Thm. 4) or bounded (Thms. 5,6); these decidability results hold even the machines are allowed additional tapes, additional reversal-bounded counters and one pushdown stack.

When one tape is unbounded, synchronization remains undecidable for ambiguous NCMs with one reversal (Thm. 9), for 3-ambiguous NCMs with unlimited counter reversals (Thm. 11), and 2-ambiguous NPDAs with three reversals (Thm. 8). Finally, synchronization is undecidable for 2-tape logarithmic space-bounded DTMs.

We also obtained undecidability of whether there exist equivalent synchronized versions of a given machine (synchronizability) for some of the above-mentioned classes; in contrast, we showed how to construct equivalent synchronized versions in two cases of space-bounded NTMs.

## References

1. Egecioglu, O., Ibarra, O.H., Tran, N.: Multitape NFA: weak synchronization of the input heads. In: Proc. of the 38th International Conference on Current Trends in Theory and Practice of Computer Science (to appear, 2012)

2. Ginsburg, G., Spanier, E.: Bounded Algol-like languages. Trans. of the Amer. Math. Society 113, 333–368 (1964)
3. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. J. Assoc. Comput. Mach. 25, 116–133 (1978)
4. Ibarra, O.H., Tran, N.: On synchronized multitape and multihead automata. In: Proc. of the 13th Int. Workshop on Descriptional Complexity of Formal Systems, pp. 184–197 (2011)
5. Parikh, R.J.: On context-free languages. J. Assoc. Comput. Mach. 13, 570–581 (1966)
6. Yu, F., Bultan, T., Cova, M., Ibarra, O.H.: Symbolic String Verification: An Automata-Based Approach. In: Havelund, K., Majumdar, R. (eds.) SPIN 2008. LNCS, vol. 5156, pp. 306–324. Springer, Heidelberg (2008)
7. Yu, F., Bultan, T., Ibarra, O.H.: Symbolic String Verification: Combining String Analysis and Size Analysis. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 322–336. Springer, Heidelberg (2009)
8. Yu, F., Bultan, T., Ibarra, O.H.: Relational String Verification Using Multi-track Automata. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 290–299. Springer, Heidelberg (2011)

# Feasible Automata for Two-Variable Logic with Successor on Data Words

Ahmet Kara[1], Thomas Schwentick[1,⋆], and Tony Tan[2,⋆⋆]

[1] Technical University of Dortmund, Germany
[2] University of Edinburgh, UK

**Abstract.** We introduce an automata model for data words, that is words that carry at each position a symbol from a finite alphabet and a value from an unbounded data domain. The model is (semantically) a restriction of data automata, introduced by Bojanczyk, et. al. in 2006, therefore it is called *weak data automata*. It is strictly less expressive than data automata and the expressive power is incomparable with register automata. The expressive power of weak data automata corresponds exactly to existential monadic second order logic with successor $+1$ and data value equality $\sim$, $\mathsf{EMSO}^2(+1, \sim)$. It follows from previous work, David, et. al. in 2010, that the nonemptiness problem for weak data automata can be decided in 2-NEXPTIME. Furthermore, we study weak Büchi automata on data $\omega$-strings. They can be characterized by the extension of $\mathsf{EMSO}^2(+1, \sim)$ with existential quantifiers for infinite sets. Finally, the same complexity bound for its nonemptiness problem is established by a nondeterministic polynomial time reduction to the nonemptiness problem of weak data automata.

## 1 Introduction

Motivated by challenges in XML reasoning and infinite-state Model Checking, an extension of strings and finitely labelled trees by data values has been investigated in recent years. In classical automata theory, a string is a sequence of positions that carry a symbol from some finite alphabet. In a nutshell, *data strings* generalize strings, in that every position additionally carries a data value from some infinite domain. In the same way, *data trees* generalize (finitely) labelled trees. In XML Theory, data trees model XML documents. Here, the data values can be used to represent attribute values or text content. Both, cannot be adequately modelled by a finite alphabet. In a Model Checking[1] scenario, the data values can be used, e.g., to represent process id's or other data.

⋆ The first and the second authors acknowledge the financial support by the German DFG under grant SCHW 678/4-1.

⋆⋆ The third author acknowledes the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.

[1] In the Model Checking setting, a position might carry a finite set of propositional variables, instead of a symbol.

Early investigations in this area usually considered strings over an "infinite alphabet", that is, each position only has a value, but no finite-alphabet symbol [2,21,7,15,16,19]. Many of the automata models and logics that have been studied for data strings and trees lack the usual nice decidability properties of automata over finite alphabets, unless strong restrictions are imposed [11,4,3,1].

A result that is particularly interesting for our investigations is the decidability of the satisfiability problem for two-variable logic over data strings [4]. Here, as usual, the logical quantifiers range over the positions of the data string and it can be checked whether a position $x$ carries a symbol $a$ (written: $a(x)$), whether it is to the left of a position $y$ ($x + 1 = y$), whether $x$ is somewhere to the left of $y$ ($x < y$) and whether $x$ and $y$ carry the same data value ($x \sim y$). The logic is denoted by $\mathsf{FO}^2(+1, <, \sim)$. The result was shown with the help of a newly introduced automata model for data words, *data automata* (DA). It turned out, that the expressive power of these automata can be actually characterized by the extension of $\mathsf{FO}^2(+1, <, \sim)$ with existential quantification over sets (of positions) and an additional predicate that holds for $x$ and $y$ if $y$ is the nearest position to the right of $x$ with the same data value.

However, the complexity of the decision procedure for $\mathsf{FO}^2(+1, <, \sim)$ is very high. The problem is equivalent to the Reachability problem for Petri nets [13], a notoriously hard problem whose complexity has not been resolved exactly. Thus, it has been investigated how the complexity can be reduced, by dropping one of the predicates $x < y$ or $x + 1 = y$. In the latter case (that is, for $\mathsf{FO}^2(<, \sim)$) the complexity decreases to NEXPTIME [4]. In the former case ($\mathsf{FO}^2(+1, \sim)$) the complexity also becomes elementary. In [3] a 3-NEXPTIME bound was shown for the case of data trees and this bound clearly carries over to data strings. A more direct proof with a 4-NEXPTIME bound was given in [9] and a 2-NEXPTIME bound was obtained in [20].

The high complexity of the satisfiability of $\mathsf{FO}^2(+1, <, \sim)$ in [4] results from the high complexity of the nonemptiness problem for data automata. One of the starting questions for this paper was:

(1) Is there a natural restriction of data automata with (i) a better complexity and (ii) a correspondence to $\mathsf{EMSO}^2(+1, \sim)$, the closure of $\mathsf{FO}^2(+1, \sim)$ under existential set quantification?

We show that such a restriction indeed exists. Data automata consist of two automata $\mathcal{A}$ and $\mathcal{B}$. Automaton $\mathcal{A}$ is a non-deterministic letter-to-letter transducer that constructs, given the finite alphabet part of the input data string[2] $u$, a new data string $w$ (where, for each position, the data value in $w$ is the same as in $u$). The second automaton $\mathcal{B}$ can then check properties of the subsequences of $w$ that carry the same data value. We define *weak data automata (WDA)* which also use a non-deterministic letter-to-letter transducer but can only test some simple constraints of the subsequences in the second part. These constraints are (unary) key, inclusion and denial constraints and they are evaluated for each

[2] The transducer also sees whether a position has the same data value as the next one.

class separately (there are no inter-class constraints). Essentially, the key difference between weak data automata and data automata is that these constraints are invariant under reordering of positions. Weak data automata extend the automata model of [9] by so-called *denial constraints*, in a sense that will be made precise in Lemma 1.

It turns out that WDA are expressively weaker than data automata, incomparable with register automata [15,1] and that their expressiveness can be precisely characterized by the extension of $\mathsf{FO}^2(+1,\sim)$ by existential set quantification, that is, $\mathsf{EMSO}^2(+1,\sim)$. As the property that we use to separate the expressive power of WDA and DA can be defined in $\mathsf{EMSO}^2(+1,<,\sim)$ we get that $\mathsf{EMSO}^2(+1,\sim) \not\equiv \mathsf{EMSO}^2(+1,<,\sim)$ as opposed to the classical setting (without data values) where $\mathsf{EMSO}^2(+1) \equiv \mathsf{EMSO}^2(+1,<)$. Indeed, one of the benefits of the logical characterization is that it gives an easy means to show non-expressibility for $\mathsf{EMSO}^2(+1,\sim)$ (and $\mathsf{FO}^2(+1,\sim)$). From results in [9] it immediately follows that the nonemptiness problem for WDA can be solved in 2-NEXPTIME.

As mentioned above, one motivation to study data strings comes from Model Checking. In that context, systems are usually considered to run forever and to produce infinite traces. Thus, data $\omega$-words need to be considered as well, and this was actually one of the main motivations of this research. In particular we address the following questions.

(2) Do the complexity results of [9] carry over to data $\omega$-strings?
(3) Can the expressibility results and logical characterizations of the first part of the paper also be established for data $\omega$-strings?

It is straightforward to adapt weak data automata for data $\omega$-strings. The transducer can simply be equipped with a Büchi acceptance mechanism. We refer to the resulting model as *weak Büchi data automata (WBDA)*. It turns out that the answer to both questions, (2) and (3), is affirmative. For (3), this is not hard to prove. The separation of WDA from DA also separates WBDA from Büchi data automata. It is also not too hard to get a logical characterization of WBDA by extending $\mathsf{EMSO}^2(+1,\sim)$ with existential set quantifiers that are semantically restricted to bind to infinite sets. The answer to question (2) required considerably more effort. However, we establish a 2-NEXPTIME upper bound for the nonemptiness problem for WBDAs by a nondeterministic polynomial time reduction to the nonemptiness for WDA.

*Related work.* Some related work was already mentioned above. The pioneering works in Linear Temporal Logic for $\omega$-words with data are the papers [11,10]. In [10] an extension of Linear Temporal Logic (LTL) to handle data values is proposed and its satisfiability problem is shown to be decidable. The decision procedure is a reduction to the reachability problem in Petri nets, thus resulting in a similarly unknown complexity as for data automata. The logic and automata considered in [11] are decidable for finite data words, but not primitive recursive, and undecidable for $\omega$-words. In [18] it is shown that with a *safety* restriction both the logic and the automata become decidable, even in EXPSPACE. In [10] a logic with $\mathsf{PSPACE}$ complexity is considered. It must be noted that in [4] it is shown that the satisfiability of $\mathsf{EMSO}^2(+1, <, \sim)$ is decidable also on $\omega$-words. In [5], $\mathsf{MSO}$ logic on data words (with possibly multiple data values per position) is compared to automata models for various types of successor relations. An extension of data automata with decidable emptiness problem is studied in [24]. In [8] it is shown that rigidly guarded MSO corresponds to recognizability by *orbit finite data monoids.*

*Organization.* We give basic definitions in Section 2. In Section 3, weak data automata are defined, their complexity is given, and their expressive power is compared with other models. Section 4 gives the logical characterization of WDA by $\mathsf{EMSO}^2(+1, \sim)$. Section 5 studies data $\omega$-strings and shows how the nonemptiness problem of WBDA can be nondeterministically reduced in polynomial time to the nonemptiness of WDA. Section 6 states some open problems. Proofs omitted due to space constraints can be found in the full version of this paper ([17]).

## 2 Notation

*Data words.* Let $\Sigma$ be a finite alphabet and $\mathfrak{D}$ an infinite set of data values. A *finite* word is an element of $\Sigma^*$, while an $\omega$-word is an element of $\Sigma^\omega$. A finite *data word* is an element of $(\Sigma \times \mathfrak{D})^*$, while a *data $\omega$-word* is an element of $(\Sigma \times \mathfrak{D})^\omega$. We often refer to data words also as *data strings.*

We write a data (finite or $\omega$-) word $w$ as $\binom{a_1}{d_1}\binom{a_2}{d_2}\cdots$, where $a_1, a_2, \ldots \in \Sigma$ and $d_1, d_2, \ldots \in \mathfrak{D}$. The symbol $a_i$ is the *label* of position $i$, while the value $d_i$ is the *data value* of position $i$. The *projection* of $w$ to the alphabet $\Sigma$ is denoted by $\mathsf{Str}(w) = a_1 a_2 \ldots$. A position in $w$ is called an $a$-position, if the label of that position is $a$.

A maximal set of positions with the same data value $d$ is called a *class* $c^d$ of the word and the $\Sigma$-string induced by the symbols at its positions is called the *class string* $w^d$. The *profile word* of a data $\omega$-word $w = \binom{a_1}{d_1}\binom{a_2}{d_2}\cdots$ is $\mathsf{Profile}(w) = (a_1, s_1), (a_2, s_2), \ldots \in (\Sigma \times \{\top, \bot\})^\omega$, where for each position $i \geq 1$ the component $s_i$ is $\top$ if and only if $d_i = d_{i+1}$. The profile word of a finite data word $\binom{a_1}{d_1}\binom{a_2}{d_2}\cdots\binom{a_n}{d_n}$ is defined similarly, with the addition that the component $s_n$ is $\bot$.

*Automata and Büchi automata.* An *automaton* $\mathcal{A}$ over the alphabet $\Sigma$ is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, F \rangle$, where $Q$ is a finite set of states, $q_0 \in Q$ is the initial state, $\Delta \subseteq Q \times \Sigma \times Q$ is a set of transitions and $F \subseteq Q$ is a set of accepting states. A run of $\mathcal{A}$ on a word $w = a_1 a_2 \ldots a_n$ is a sequence $\rho = q_1 \ldots q_n$ of states from $Q$ such that $(q_i, a_{i+1}, q_{i+1}) \in \Delta$ for each $0 \leq i < n$. The run $\rho$ is accepting if $q_n \in F$.

A *Büchi automaton* $\mathcal{A}$ is syntactically just an automaton. A run of $\mathcal{A}$ on an $\omega$-word $w = a_1 a_2 \ldots$ is an infinite sequence $\rho = q_1 q_2 \ldots$ of states from $Q$ such that $(q_i, a_{i+1}, q_{i+1}) \in \Delta$ for each $i \geq 0$. Let $\mathsf{Inf}(\rho)$ denote the set of states that appear infinitely many times in $\rho$. The run $\rho$ is accepting if $\mathsf{Inf}(\rho) \cap F \neq \emptyset$.

A word (resp. an $\omega$-word) $w$ is accepted by an automaton (resp. Büchi automaton) $\mathcal{A}$, if there exists an accepting run of $\mathcal{A}$ on $w$. As usual, $\mathcal{L}(\mathcal{A})$ (resp. $\mathcal{L}^\omega(\mathcal{A})$) denotes the set of words (resp. $\omega$-words) accepted by the automaton $\mathcal{A}$.

*Letter-to-letter transducers.* A *letter-to-letter transducer* over the input alphabet $\Sigma$ and the output alphabet $\Gamma$ is a tuple $\mathcal{T} = \langle \Sigma, \Gamma, Q, q_0, \Delta, F\rangle$, where $Q$, $q_0$, $F$ are the set of states, the initial state, and the set of accepting states, respectively, and $\Delta \subseteq Q \times \Sigma \times Q \times \Gamma$ is the set of transitions. The intuitive meaning of a transition $(q, a, q', \gamma)$ is that when the automaton is in state $q$, reading the symbol $a$, then it can move to the state $q'$ and output $\gamma$. A *run* of $\mathcal{T}$ on a word $w = a_1a_2 \dots a_n$ is a sequence $(q_1, \gamma_1), \dots, (q_n, \gamma_n)$ of pairs from $Q \times \Gamma$ such that $(q_i, a_{i+1}, q_{i+1}, \gamma_{i+1}) \in \Delta$ for each $0 \le i < n$. Likewise, a *run* of $\mathcal{T}$ on an $\omega$-word $w = a_1a_2 \dots$ is a sequence $(q_1, \gamma_1), (q_2, \gamma_2), \dots$ over $Q \times \Gamma$ such that $(q_i, a_{i+1}, q_{i+1}, \gamma_{i+1}) \in \Delta$ for each $i \ge 0$. A run is *accepting* if it is accepting in the sense of (Büchi) automata. We say that $v = \gamma_1\gamma_2 \dots$ is an output of $\mathcal{T}$ on $w$, if there exists an accepting run $(q_1, \gamma_1), (q_2, \gamma_2), \dots$ of $\mathcal{T}$ on $w$.

*Data automata.* A *data automaton (DA)* is a pair $(\mathcal{A}, \mathcal{B})$, where $\mathcal{A}$ is a letter-to-letter transducer with input alphabet $\Sigma \times \{\top, \bot\}$ and output alphabet $\Gamma$ and $\mathcal{B}$ is a finite state automaton over the alphabet $\Gamma$. A data word $w$ is accepted by $(\mathcal{A}, \mathcal{B})$ if the following holds.

– $\mathsf{Profile}(w)$ is accepted by $\mathcal{A}$, yielding an output $u$.
– For each data value $d$ of $w$, the class string $u^d$ is accepted by $\mathcal{B}$.

Data automata were introduced in the stated form in [4]. In [1] it was shown that their expressive power is not affected, if $\mathcal{A}$ gets $\mathsf{Str}(w)$ as input as opposed to $\mathsf{Profile}(w)$. In more recent papers, data automata are therefore defined in the (syntactically) weaker form with input $\mathsf{Str}(w)$.

## 3 Weak Data Automata

In this section we define a new automata model for finite data words and study its expressive power and its complexity. The model follows a similar approach as the model of data automata. The profile of the input data word is transformed by a letter-to-letter transducer and then further conditions on the resulting class strings are imposed. However, the conditions that can be stated in the new automata model are much more limited than those of a data automaton (hence the name *weak* data automata).

Let $\Gamma$ be an alphabet. Weak data automata allow three kinds of data constraints over $\Gamma$:

1. *key constraints*, written in the form: $\mathsf{key}(\gamma)$, where $\gamma \in \Gamma$.
2. *inclusion constraints*, written in the form: $V(\gamma) \subseteq \bigcup_{\gamma' \in R} V(\gamma')$, where $\gamma \in \Gamma$, $R \subseteq \Gamma$.
3. *denial constraints*, written in the form: $V(\gamma) \cap V(\gamma') = \emptyset$, where $\gamma, \gamma' \in \Gamma$.

Whether a data word $w = \binom{a_1}{d_1} \cdots \binom{a_n}{d_n}$ satisfies a data constraint $C$, written as $w \models C$, is defined as follows. In the following, we denote by $V_w(a)$, the set of data values found in $a$-positions in $w$, i.e., $V_w(a) = \{d_i \mid a_i = a\}$, for each $a \in \Gamma$.

1. $w \models \mathsf{key}(\gamma)$, if every two $\gamma$-positions in $w$ have different data values.
2. $w \models V(\gamma) \subseteq \bigcup_{\gamma' \in R} V(\gamma')$, if $V_w(\gamma) \subseteq \bigcup_{\gamma' \in R} V_w(\gamma')$.
3. $w \models V(\gamma) \cap V(\gamma') = \emptyset$, if $V_w(\gamma) \cap V_w(\gamma') = \emptyset$.

It should be noted that the satisfaction of a data constraint by a data word does not depend on the order of the positions of the data word. If $\mathcal{C}$ is a collection of data constraints, then we write $w \models \mathcal{C}$, if $w \models C$ for all $C \in \mathcal{C}$.

A *weak data automaton (WDA)* over the alphabet $\Sigma$ is a pair $(\mathcal{A}, \mathcal{C})$, where $\mathcal{A}$ is a letter-to-letter transducer with input alphabet $\Sigma \times \{\top, \bot\}$ and output alphabet $\Gamma$ and $\mathcal{C}$ is a collection of data constraints over the alphabet $\Gamma$. A data word $w = \binom{a_1}{d_1} \cdots \binom{a_n}{d_n}$ is accepted by a WDA $(\mathcal{A}, \mathcal{C})$, if

– there is an accepting run of $\mathcal{A}$ on Profile($w$), with an output $\gamma_1 \ldots \gamma_n$, and
– the induced data word $w = \binom{\gamma_1}{d_1} \cdots \binom{\gamma_n}{d_n}$ satisfies all the constraints in $\mathcal{C}$.

We write $\mathcal{L}(\mathcal{A}, \mathcal{C})$ to denote the language that consists of all data words accepted by $(\mathcal{A}, \mathcal{C})$.

We first discuss some extensions of WDA by the constraints that were studied in [9].

– *Disjunctive key constraints* are written in the form: $\mathsf{key}(K)$, where $K \subseteq \Gamma$. Such a constraint is satisfied by a data word if each of its classes has at most one position with a symbol from $K$.
– *Disjunctive inclusion constraints* are written in the form: $\bigcup_{\gamma \in S} V(\gamma) \subseteq \bigcup_{\gamma' \in R} V(\gamma')$, where $S, R \subseteq \Gamma$. Such a constraint is satisfied by a data word if each class with a position with a symbol from $S$ also has a position with a symbol from $R$.

An *extended weak data automaton* is defined like a WDA but it further allows disjunctive key and inclusion constraints. The proof of the following Lemma can be found in the full version of this paper ([17]).

**Lemma 1.** *From each extended WDA $(\mathcal{A}, \mathcal{C})$ an equivalent WDA of polynomial size can be constructed in polynomial time.*

Next, we compare the expressive power of weak data automata with other automata models for data words. More precisely we compare it with register automata [15,1] and data automata. Register automata are an extension of finite state automata with a fixed number of registers in which they can store data values and compare them with the data value of subsequent positions. For a precise definition we refer[3] the reader to [1].

[3] The precursor model *finite-memory automata* was defined on “strings over infinite alphabets”, that is, essentially data strings without a $\Sigma$-component [15].

We consider the following two data languages.

- $L_{a<b}$ consists of all data words over the alphabet $\{a, b\}$ with the property that for every $a$-position $i$ there is a $b$-position $j > i$ with the same data value;
- $L_{a*b}$ is the subset of $L_{a<b}$ where the next $b$-position $j$ with the same data value as $i$ always satisfies $j = i + 2$.

**Lemma 2.** *Neither $L_{a*b}$ nor $L_{a<b}$ can be recognized by a WDA.*

*Proof.* We first show that no WDA recognizes $L_{a*b}$. Towards a contradiction, we thus assume that $L_{a*b}$ is recognized by some weak data automaton $(\mathcal{A}, \mathcal{C})$.

To this end, let $n = |\Gamma|^4+1$ and let $d_1, d'_1, d_2, d'_2, \ldots d_n, d'_n$ be pairwise different data values. We consider the data word

$$w = \binom{a}{d_1}\binom{a}{d'_1}\binom{b}{d_1}\binom{b}{d'_1}\binom{a}{d_2}\binom{a}{d'_2}\binom{b}{d_2}\binom{b}{d'_2}\cdots\binom{a}{d_n}\binom{a}{d'_n}\binom{b}{d_n}\binom{b}{d'_n}$$

of length $4n$. Clearly, $w$ is in $L_{a*b}$ and its profile is $((a, \bot)(a, \bot)(b, \bot)(b, \bot))^n$.

Let $\gamma = \gamma_1\gamma_2 \cdots \gamma_{4n}$ be an output of $\mathcal{A}$ on the profile of $w$ such that $\binom{\gamma_1}{d_1} \cdots \binom{\gamma_{4n}}{d'_n}$ satisfies all constraints in $\mathcal{C}$. By the choice of $n$, there exist numbers $i, j$ with $0 \leq i < j < n$ such that $\gamma_{4i+1}\gamma_{4i+2}\gamma_{4i+3}\gamma_{4i+4} = \gamma_{4j+1}\gamma_{4j+2}\gamma_{4j+3}\gamma_{4j+4}$.

Let $u$ be the data word obtained from $w$ by swapping the positions of the data values $d_{i+1}d'_{i+1}$ and $d_{j+1}d'_{j+1}$. That is, $u$ equals

$$\binom{a}{d_1}\cdots\binom{a}{d_{i+1}}\binom{a}{d'_{i+1}}\binom{b}{d_{j+1}}\binom{b}{d'_{j+1}}\cdots\binom{a}{d_{j+1}}\binom{a}{d'_{j+1}}\binom{b}{d_{i+1}}\binom{b}{d'_{i+1}}\cdots\binom{b}{d'_n}.$$

Clearly, $u \notin L_{a*b}$. However, because $\mathsf{Profile}(u) = \mathsf{Profile}(w)$, $\gamma_1\gamma_2 \ldots \gamma_{4n}$ is also an output of $\mathcal{A}$ on $\mathsf{Profile}(u)$. Moreover, $V_u(\gamma) = V_w(\gamma)$ for each $\gamma \in \Gamma$, and therefore the validity of inclusion and denial constraints does not change. Furthermore, as in $u$ and $w$ every data value occurs at exactly one $a$-position and at exactly one $b$-position, they cannot be distinguished by key constraints, either. Thus, $u \in \mathcal{L}(\mathcal{A}, \mathcal{C})$, the desired contradiction.

The proof for $L_{a<b}$ is exactly the same, as $w \in L_{a<b}$ and $u \notin L_{a<b}$ (because of $\binom{a}{d_{j+1}}$). □

**Theorem 3.** *(a) The class of data languages that are recognized by WDA is strictly included in the class of data languages recognized by DA.*
*(b) The classes of languages recognized by WDA and by register automata are incomparable.*

*Proof.* Towards (a) we first show that every WDA can be translated into a DA and thus WDA recognize a subclass of DA. That the subclass is strict can then be concluded from (b) as register automata are captured by DA [1] and thus there is a data language that can be recognized by a DA but not a WDA.

Let thus $(\mathcal{A}, \mathcal{C})$ be a WDA. Then $(\mathcal{A}, \mathcal{B})$ is a data automaton for $L(\mathcal{A}, \mathcal{C})$, where the automaton $\mathcal{B}$ tests the constraints in $\mathcal{C}$ as follows.

– For every key constraint $\mathsf{key}(\gamma)$ of $\mathcal{C}$, $\mathcal{B}$ tests that every class string has at most one $\gamma$-position.
– For every inclusion constraint $V(\gamma) \subseteq \bigcup_{\gamma' \in R} V(\gamma')$, $\mathcal{B}$ tests that every class string with a $\gamma$-position also has a $\gamma'$-position, for some $\gamma' \in R$.
– For every denial constraint $V(\gamma) \cap V(\gamma') = \emptyset$, $\mathcal{B}$ checks that classes with a $\gamma$-position do not have any $\gamma'$-positions.

To show statement (b) we first consider the separation language $L = L_{a*b}$ which cannot be recognized by a WDA by Lemma 2. However, $L_{a*b}$ can be easily recognized by a register automaton that always stores the last two data values in two registers and the information about their symbols in its state.

On the other hand, it is trivial to show that the set of all data strings over $\Sigma = \{a\}$ in which every data value occurs only once can be recognized by a WDA $(\mathcal{A}, \mathcal{C})$, where $\mathcal{A}$ is simply an identity transducer and $\mathcal{C} = \{\mathsf{key}(a)\}$. It is already shown in [15] that such a language cannot be recognized by register automata. $\square$

The complexity of the nonemptiness problem for WDA follows directly from results in [9].

**Theorem 4.** *The nonemptiness problem for WDA is decidable in 2-NEXPTIME.*

*Proof.* In [9], it was shown that given an automaton $\mathcal{A}$ that reads profile strings and a set $\mathcal{C}$ of disjunctive key and inclusion constraints, to decide whether there is a data word $w$ such that $\mathcal{A}$ accepts $\mathsf{Profile}(w)$ and $w \models \mathcal{C}$ can be done in nondeterministic double exponential time.

Clearly, this is basically the same as the nonemptiness problem for WDA with disjunctive key and inclusion constraints only. It thus only remains to show that denial constraints can be translated into disjunctive constraints in a nonemptiness respecting fashion. To this end, a denial constraint $V(\gamma_1) \cap V(\gamma_2) = \emptyset$ can be replaced as follows. We add two new symbols $\gamma_1', \gamma_2'$ and require that in each class with $\gamma_i$ one $\gamma_i'$ occurs but $\gamma_1'$ and $\gamma_2'$ do not co-occur by two inclusion constraints $V(\gamma_1) \subseteq V(\gamma_1')$ and $V(\gamma_2) \subseteq V(\gamma_2')$ and a disjunctive key constraint for $\{\gamma_1', \gamma_2'\}$. $\square$

## 4 A Logical Characterization of Weak Data Automata

In this section, we give a logical characterization of the data languages recognized by weak data automata in terms of existential second order logic. The characterization is an analogue of the Theorem of Büchi, Elgot and Trakhtenbrot [6,12,23] for string languages. This theorem can be stated for various logics, the most interesting one for our context is that $\mathsf{EMSO}^2(+1)$ characterizes exactly the regular languages.

We use logical structures $w = \langle \{1, \ldots, n\}, +1, <, \{a(\cdot)\}_{a \in \Sigma}, \sim \rangle$ to represent data words, where $\{1, \ldots, n\}$ is the set of positions, $+1$ is the successor relation (i.e., $+1(i,j)$ if $i + 1 = j$), $<$ is the order relation (i.e., $<(i,j)$ if $i < j$), the

$a(\cdot)$'s are the label relations, and $i \sim j$ holds if positions $i$ and $j$ have the same data value. As the empty data word can not be properly represented, the logical characterization of WDA ignores the empty data word. That is, we associate with each WDA a formula $\varphi$ such that, if the automaton accepts the empty data string then its language is the language defined by $\varphi$ augmented by $\epsilon$.

For a set $\mathcal{S} \subseteq \{+1, <, \sim\}$ of relation symbols, we write $\mathsf{FO}(\mathcal{S})$ for first-order logic with the vocabulary $\mathcal{S}$, $\mathsf{MSO}(\mathcal{S})$ for monadic second-order logic (which extends $\mathsf{FO}(\mathcal{S})$ with quantification over sets of positions), and $\mathsf{EMSO}(\mathcal{S})$ for existential monadic second order logic, that is, all sentences of the form $\exists R_1 \ldots \exists R_m\ \psi$, where $\psi$ is an $\mathsf{FO}(\mathcal{S})$ formula extended with the unary predicates $R_1, \ldots, R_m$. By $\mathsf{FO}^2(\mathcal{S})$ we denote the restriction of $\mathsf{FO}(\mathcal{S})$ to sentences with two variables $x$ and $y$, and by $\mathsf{EMSO}^2(\mathcal{S})$ the restriction of $\mathsf{EMSO}(\mathcal{S})$ where the first-order part uses only two variables.

### 4.1 From Weak Data Automata to $\mathsf{EMSO}^2(+1, \sim)$

**Theorem 5.** *For every weak data automaton $(\mathcal{A}, \mathcal{C})$, an equivalent $\mathsf{EMSO}^2(+1, \sim)$ formula $\varphi$ is constructible in polynomial time.*

The construction can be found in the full version [17]. It is similar as the classical translation from NFAs to $\mathsf{MSO}$ formulas. See, for example, [22].

### 4.2 From $\mathsf{EMSO}^2(+1, \sim)$ to Weak Data Automata

**Theorem 6.** *There is an algorithm that translates every $\mathsf{EMSO}^2(+1, \sim)$ formula $\varphi$ into an equivalent weak data automaton $(\mathcal{A}, \mathcal{C})$ in doubly exponential time. In particular, the output alphabet $\Gamma$ of $\mathcal{A}$ and the number of constraints are $\mathcal{C}$ is at most exponential.*

The first two steps of the following translation are very similar to the one given in [4].

*Proof.* The algorithm first transforms $\varphi$ into an equivalent $\mathsf{EMSO}^2(+1, \sim)$ formula in Scott normal form (SNF) of the form $\psi = \exists R_1 \ldots \exists R_n [\forall x \forall y\ \chi' \wedge \bigwedge_{i=1}^{m} \forall x \exists y\ \chi_i']$, where $\chi'$ and each $\chi_i'$ are quantifier-free [14]. The size of $\psi$ is linear in the size of $\varphi$, in particular, $n = \mathcal{O}(|\varphi|)$ and $m = \mathcal{O}(|\varphi|)$.

Then it rewrites formula $\chi'$ into an, at most exponential, conjunction $\chi = \bigwedge_j \neg(\alpha_j(x) \wedge \beta_j(y) \wedge \delta_j(x, y) \wedge \epsilon_j(x, y))$, where, for every $j$, $\alpha_j, \beta_j$ are conjunctions of literals with unary relation symbols, $\delta_j$ is $x \sim y$ or $x \not\sim y$ and $\epsilon_j(x, y)$ is one[4] of $x = y$, $y = x + 1$ and $|x - y| > 1$, where the latter is an abbreviation for the formula $\neg(y = x + 1) \wedge \neg(x = y + 1) \wedge x \neq y$, expressing that the distance of $x$ and $y$ is at least two.

Likewise, it rewrites every $\chi_i'$ into an, at most exponential, disjunction $\chi_i = \bigvee_j (\alpha_j^i(x) \wedge \beta_j^i(y) \wedge \delta_j^i(x, y) \wedge \epsilon_j^i(x, y))$, where the atomic formulas are of the respective forms as above.

[4] The case $x = y + 1$ does not need to be considered as it can be obtained by swapping $x$ and $y$.

The idea of the construction is that $\mathcal{A}$ guesses some relations that allow to state some of the properties expressed in $\psi$ by constraints of $\mathcal{C}$. The details are given in [17]. $\square$

We note that in the upper bound of the algorithm for nonemptiness of WDA transferred from [9], the doubly exponential term only depends on the alphabet size. By combing this with the bounds of Theorem 6 we obtain a 3-NEXPTIME upper bound for satisfiability of $\mathsf{FO}^2(+1, \sim)$ (which is worse than the bound in [20]). We also note that the construction underlying the proof of Theorem 6 can be turned into a nondeterministic exponential time reduction from satisfiability for $\mathsf{FO}^2(+1, \sim)$ to nonemptiness for WDA resulting in an automaton with a singly exponential number of states. The reduction guesses the order in which types appear in the accepted string (as opposed to the construction in the proof of Theorem 6).

The previous two theorems yield the following logical characterization.

**Theorem 7.** *Weak data automata and* $\mathsf{EMSO}^2(+1, \sim)$ *are equivalent in expressive power.*

We note that on strings $\mathsf{EMSO}^2(+1)$ and $\mathsf{EMSO}^2(+1, <)$ are expressively equivalent. It is an interesting consequence of the above characterization that this equivalence does not hold for data strings.

**Corollary 8.** $\mathsf{EMSO}^2(+1, \sim)$ *is strictly less expressive than* $\mathsf{EMSO}^2(+1, <, \sim)$.

*Proof.* The inclusion holds by definition. It is strict because the language $L_{a<b}$ cannot be recognized by an WDA (Lemma 2) and thus cannot be defined in $\mathsf{EMSO}^2(+1, \sim)$, but it can be expressed by the simple formula $\forall x \exists y (a(x) \to (b(y) \wedge x < y \wedge x \sim y))$. $\square$

## 5 Weak Büchi Data Automata

In this section we consider automata and logics for *data $\omega$-words*, that is, data words of infinite length. Weak data automata $(\mathcal{A}, \mathcal{C})$ can easily be adapted for data $\omega$-words. The automaton $\mathcal{A}$ is simply interpreted as a letter-to-letter Büchi transducer. A run is accepting if it visits infinitely often a state from $F$. We refer to the resulting model as weak Büchi data automata (WBDA). We write $\mathcal{L}^\omega(\mathcal{A}, \mathcal{C})$ for the set of data $\omega$-words accepted by $(\mathcal{A}, \mathcal{C})$. We remark that for data automata and register automata there is no "official" variant for data $\omega$-words. However, Theorem 3 carries over to any such models, provided that they deal with data $\omega$-words obtained by padding a finite data word by an infinite suffix $\binom{a}{d}^\omega$ in a natural way.

We use logical structures $w = \langle \mathbb{N}, +1, <, \{a(\cdot)\}_{a \in \Sigma}, \sim \rangle$, to represent data $\omega$-words, where $\mathbb{N}$ is the set $\{1, 2, \ldots\}$ of natural numbers which represent the positions and the other relations are as in the case of data words. For a set $\mathcal{S} \subseteq \{+1, <, \sim\}$ of relation symbols $\mathsf{E}_\infty\mathsf{MSO}(\mathcal{S})$ consists of all formulas of the form $\exists_\infty R_1 \ldots \exists_\infty R_m \exists S_1 \ldots \exists S_\ell \; \varphi$ where $\varphi \in \mathsf{FO}^2(\mathcal{S})$. Here all relation symbols $R_i, S_i$ are unary. The $\exists_\infty$ are semantically restricted to bind to infinite sets only.

*Remark 9.* It is folklore that languages (without data) accepted by Büchi automata are precisely languages expressible in formulas of the form:

$$\exists_\infty R_1 \cdots \exists_\infty R_m \exists S_1 \cdots \exists S_\ell \quad \varphi$$

for some $\varphi \in \mathsf{FO}^2(+1)$. However, we have not found an explicit reference for this result in the literature. We remark that it is relatively straightforward to show that quantification of finite sets is not necessary. However, such finite set quantifications are included for the simplicity of our presentation and proofs.

The following theorem is a straightforward generalization of Theorem 7. The proof is given in [17].

**Theorem 10.** *Weak Büchi data automata and $\mathsf{E}_\infty\mathsf{MSO}^2(+1,\sim)$ are equivalent in expressive power.*

**Theorem 11.** *The nonemptiness problem for weak Büchi data automata is decidable in 2-NEXPTIME.*

The proof is by a reduction to the nonemptiness problem for WDA. The result then follows from Theorem 4. The approach is a classical one. We show that if the language of a WBDA $(\mathcal{A}, \mathcal{C})$ is non-empty then a finite data string of the form $uv$ can be constructed such that there is a run of $\mathcal{A}$ which loops over $v$. The "unravelling" $uv^\omega$ is then also accepted by the automaton. But some care is needed to assign data values in a suitable manner. Details are given in [17].

## 6 Conclusion

We conclude this paper with two open problems for future directions. An obvious open problem is to determine the exact complexity of the nonemptiness problem for weak data automata. The current 2-NEXPTIME yields a 3-NEXPTIME upper bound for the satisfiability problem for $\mathsf{EMSO}^2(+1,\sim)$. However, as the latter problem can be solved in 2-NEXPTIME [20], there might be further room for improvement.

Another interesting question is how our results can be applied to temporal logics. In [11], it is shown that the *simple* fragment of freeze LTL with one register has the same expressive power as a certain two variable logic. We conjecture that there is a correspondence between our logics and the restriction of simple LTL to the operators $\mathtt{X}$, $\mathtt{X}^{-1}$ and an operator that allows navigation to any other position.

**Acknowledgement.** We thank Christof Löding for helpful remarks on automata and logics for $\omega$-words and Thomas Zeume for thorough proof reading.

## References

1. Björklund, H., Schwentick, T.: On notions of regularity for data languages. Theor. Comput. Sci. 411(4-5), 702–715 (2010)

2. Boasson, L.: Some applications of CFL's over infinte alphabets. Theoretical Computer Science, 146–151 (1981)
3. Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. J. ACM 56(3) (2009)
4. Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: LICS, pp. 7–16 (2006)
5. Bollig, B.: An automaton over data words that captures EMSO logic. CoRR abs/1101.4475 (2011)
6. Büchi, J.R.: Weak second-order arithmetic and finite automata. Z. Math. Logik Grundl. Math. 6, 66–92 (1960)
7. Cheng, E.Y.C., Kaminski, M.: Context-free languages over infinite alphabets. Acta Inf. 35(3), 245–267 (1998)
8. Colcombet, T., Ley, C., Puppis, G.: On the Use of Guards for Logics with Data. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 243–255. Springer, Heidelberg (2011)
9. David, C., Libkin, L., Tan, T.: On the Satisfiability of Two-Variable Logic over Data Words. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR-17. LNCS, vol. 6397, pp. 248–262. Springer, Heidelberg (2010)
10. Demri, S., D'Souza, D., Gascon, R.: A Decidable Temporal Logic of Repeating Values. In: Artemov, S., Nerode, A. (eds.) LFCS 2007. LNCS, vol. 4514, pp. 180–194. Springer, Heidelberg (2007)
11. Demri, S., Lazic, R.: LTL with the freeze quantifier and register automata. ACM Trans. Comput. Log. 10(3) (2009)
12. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Transactions of The American Mathematical Society 98, 21 (1961)
13. Gischer, J.L.: Shuffle languages, Petri nets, and context-sensitive grammars. Commun. ACM 24(9), 597–605 (1981)
14. Grädel, E., Otto, M.: On logics with two variables. Theor. Comput. Sci. 224(1-2), 73–113 (1999)
15. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. 134(2), 329–363 (1994)
16. Kaminski, M., Tan, T.: Regular expressions for languages over infinite alphabets. Fundam. Inform. 69(3), 301–318 (2006)
17. Kara, A., Schwentick, T., Tan, T.: Feasible automata for two-variable logic with successor on data words, arXiv:1110.1221v1
18. Lazic, R.: Safety alternating automata on data words. ACM Trans. Comput. Log. 12(2), 10 (2011)
19. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Log. 5(3), 403–435 (2004)
20. Niewerth, M., Schwentick, T.: Two-variable logic and key constraints on data words. In: ICDT, pp. 138–149 (2011)
21. Otto, F.: Classes of regular and context-free languages over countably infinite alphabets. Discrete Applied Mathematics 12(1), 41–56 (1985)
22. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. III, pp. 389–455. Springer, Heidelberg (1997)
23. Trakhtenbrot, B.: Finite automata and logic of monadic predicates. Doklady Akademii Nauk SSSR 140, 326–329 (1961)
24. Wu, Z.: A decidable extension of data automata. In: GandALF, pp. 116–130 (2011)

# Nash Equilibria in Concurrent Priced Games

Miroslav Klimoš[1], Kim G. Larsen[2], Filip Štefaňák[1], and Jeppe Thaarup[2]

[1] Masaryk University, Faculty of Informatics, Czech Republic
[2] Aalborg University, Department of Computer Science, Denmark

**Abstract.** Concurrent game structures model multi-player games played on finite graphs where the players simultaneously choose their moves and collectively determine the next state of the game. We extend this model with prices on transitions for each player. We study pure Nash equilibria in this framework where each player's payoff is the accumulated price of all transitions until reaching their goal state. We provide a construction of a Büchi automaton accepting all Nash equilibria outcomes and show how this construction can be used to solve a variety of related problems, such as finding pareto-optimal equilibria. Furthermore, we prove the problem of deciding the existence of equilibria to be NP-complete.

## 1 Introduction

Games played on graphs have enjoyed much attention from computer scientists in the past decades. Traditionally, they have been used to model scenarios where an actor tries to find a course of action against an unpredictable environment. Games have proven to be a helpful formalism with many applications. Bisimulation, accepting conditions of alternating automata, satisfiability of predicate logic can all be expressed as a two-player game with antagonistic objectives. Only one player can win in this case and the focus is usually limited to finding out which player has a winning strategy.

*Non-zero-sum Games.* In *non-zero-sum* games the players have independent objectives. Each player only cares about their own objective and does not care about objectives of others. Furthermore, it is natural to consider more than two players in this context. Such generalization of games allows for more realistic expression of real world problems and has been prominently used in economics, evolutionary biology or political science. In computer science, they have been used to model network routing problems [5]. The non-zero-sum games have been the focus of the *game theory* branch of mathematics for many years. However, strategies are rarely studied as objects that have an internal structure.

The objectives of players in non-zero-sum games can be qualitative or quantitative. In the qualitative setting, each player can either win or lose, so an outcome of the game has a set of winners. In the quantitative setting, the result of each player is a number – the cost – which they try to minimize (or maximize – in this case, the number is called payoff). In our case of graph games, the moves of the game are equipped with individual prices for each player.

*Concurrent Games.* Traditionally, the players take their decicions with full information and the game is turn-based. If the result depends on simultaneous and secret choices of multiple players, such as in the game rock-paper-scissors, we call the game *concurrent* [1]. Concurrent games are sufficient to describe all turn-based games, but they can model additional interesting problems.

*Nash Equilibria.* Rational players adjust their play to the play of their opponents to improve their own benefit. If the game is repeated, the course of the game changes until they reach a situation where no player can further improve by unilaterally changing their play. Such state is called a *Nash equilibrium* [6] and it is the game theorists' tool of choice for the analysis of non-zero-sum games. Barring pacts between the players, the situation always stabilizes in a state that is a Nash equilibrium. A *pure* Nash equilibrium does not always exist and is not always optimal, but a game can also have multiple equilibria.

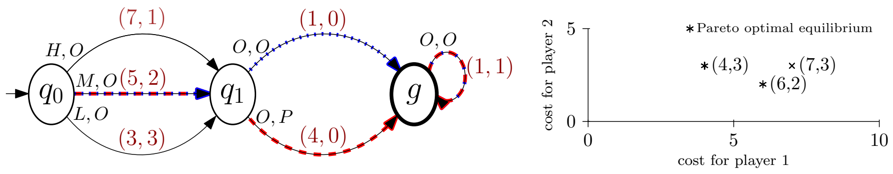


**Fig. 1.** A Priced Concurrent Games Structure and the Cost for its Equilibria

*Our Contribution.* In Section 2, we introduce Priced Concurrent Games Structures (PCGS), deterministic concurrent game graphs with non-negative integer prices on transitions for each player with individual reachability objectives. An example of a two-player PCGS can be seen in Fig. 1. From each state, a transition is determined by a choice of both players (pair of letters). Each transition is assigned a pair of numbers representing costs for the respective players. The goal state for both players is a bold circle.

A player provides a strategy that can consider the whole history of the game to choose a next move. The combination of strategies determines a run in the graph, which yields the cost for each player, defined as the accumulated price of all transitions until reaching their goal state.

The studied problem is to characterize all pure-strategy Nash equilibria of a given game. Variants of the problem include deciding existence of an equilibrium and limiting the search to equilibria with the costs of players constrained by bounds. Costs of all equilibria for the example in Fig. 1 are plotted in the chart.

In sections 3 and 4 we identify all outcomes of pure-strategy Nash equilibria by constructing a Büchi automaton accepting precisely the language of outcomes of all equilibrium strategy profiles satisfying a bound vector. Such characterization allows for simple reduction of other similar problems, e.g. deciding the existence of any equilibrium by checking emptiness of the language of a Büchi automaton.

In Section 5, we characterize the complexity of the decision problem by proving that it is NP-complete in its several variants, except for the special case of turn-based games without bounds. We prove that an equilibrium always exists in turn-based games, which makes the general decision problem trivial.

*Related Work.* Bouyer et. al. explore in [2] the existence of Nash equilibria in multiplayer concurrent games with reachability objectives. They include timed games, but they only consider qualitative reachability objectives. Brihaye et. al. study in [3] turn-based quantitative multiplayer games with reachability objectives. They prove existence of finite-memory Nash equilibria in such games. We confirm this in our framework as a corollary of our main result. However, it is necessary to point out that the formalisms are not completely equivalent. Most recently, Ummels et. al. study Nash equilibria in [7], using concurrent games but only with respect to limit-average objectives. The important distinction is that the initial part of the game is irrelevant to them. Thus, although related, the studied problems are quite different.

## 2 Preliminaries

We start with definitions of Concurrent Game Structures, computations, (full-memory) strategies and strategy profiles and outcomes of strategies. Then we introduce Priced Concurrent Game Structures by adding prices to transitions. Afterwards we formally define Nash equilibria on priced games.

We use the symbol $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ for the set of natural numbers with zero and infinity. The $a$-th projection of a vector $X$ is denoted by $X_a$. We use the notation $X_{-a} = (X_1 \dots X_{a-1}, X_{a+1} \dots)$ for the vector $X$ without its $a$-th element. We define a vector extension operator $\rhd_a$, which adds the first argument to the position $a$ in the vector given as the second argument, i.e. $x_a \rhd_a (x_1 \dots x_{a-1}, x_{a+1} \dots x_n) = (x_1 \dots x_n)$ and $X_a \rhd_a X_{-a} = X$.

A word over alphabet $\Sigma$ is a (finite or infinite) sequence of elements from $\Sigma$. Given a word $w$ and $i \leq |w|$, $w[i]$ denotes the $i$-th element of $w$, $w_i$ is a prefix of $w$ of length $i$, and $w^i$ is the $i$-th suffix s. t. $w = w_i.w^i$. An empty word is denoted by $\epsilon$.

For the rest of the article we use standard comparison operators over natural numbers including zero and special symbols $\infty$ and $\perp$. For the sake of comparison, $\infty$ is the largest element and $\perp$ is the smallest element. Addition or subtraction involving $\infty$ results in $\infty$ (except when $\perp$ is involved). Addition or subtraction involving $\perp$ always results in $\perp$.

**Definition 1 (Concurrent Game Structures).** *A Concurrent Game Structure (CGS) is a tuple $(K, Q, q_0, \Phi, \phi, \mathbb{M}, \Delta, \delta)$ with the following components:*

- *A natural number $K \geq 1$ of* players. *We identify the players with numbers $1, \dots, K$ and we use notation $\Omega = \{1, \dots, K\}$ for the set of players.*
- *A finite set $Q$ of* states.
- *An* initial state *$q_0 \in Q$.*
- *A finite set $\Phi$ of* propositions.
- *A* labeling function *$\phi : Q \to 2^{\Phi}$, such that for each state $q \in Q$, $\phi(q) \subseteq \Phi$ is a set of propositions true at $q$.*

- *A non-empty, finite set* $\mathbb{M}$ *of* moves*.*
- *A* move function $\Delta : \Omega \times Q \to 2^{\mathbb{M}} \setminus \{\emptyset\}$*, that defines a set of possible moves for each player and each state. For each state* $q \in Q$*, a move vector at* $q$ *is a vector* $J \in \mathbb{M}^k$ *such that* $J_a \in \Delta_a(q)$ *for each player* $a$*. Given a state* $q \in Q$ *we write* $\Delta(q) = \prod_{a\in\Omega} \Delta_a(q)$ *for the set of all move vectors. We denote by* $\Delta_{-b}(q) = \prod_{a\in\Omega, a\neq b} \Delta_a(q)$ *the set of vectors of the possible moves of all players except* $b$*.*
- *A* transition function $\delta$*, such that for each state* $q \in Q$ *and each move vector* $J \in \Delta(q)$*, it determines a state* $\delta(q, J) \in Q$ *that results from state* $q$ *if every player* $a \in \Omega$ *chooses move* $J_a$*.*

Let us remark that commonly studied turn-based games are a special case of CGS, where in every state each player but one has exactly one possible move. We define the extended transition function $\hat{\delta}$ over finite words of move vectors inductively: $\hat{\delta}(q, \epsilon) = q$ and $\hat{\delta}(q, J.\Lambda) = \hat{\delta}(\delta(q, J), \Lambda)$, where $J \in \mathbb{M}^K$, $\Lambda \in (\mathbb{M}^K)^*$.

**Definition 2 (Computation).** *Let* $\Lambda$ *be a (finite or infinite) word over alphabet* $\mathbb{M}^K$ *and* $q$ *a state of a CGS. We say that* $\Lambda$ *is a* computation from $q$ *if for each position* $i \in \mathbb{N}$*,* $\Lambda[i+1] \in \Delta(\hat{\delta}(q, \Lambda_i))$*.*

**Lemma 3.** *If* $\Lambda$ *is a computation from state* $q$*, then* $\Lambda^i$ *is a computation from state* $\hat{\delta}(q, \Lambda_i)$*.*

**Definition 4 (Strategy).** *Given a CGS* $T$*,* $q \in Q$ *and* $a \in \Omega$*, a function* $f : (\mathbb{M}^K)^* \to \mathbb{M}$ *is a* (pure) strategy from state $q$ for player $a$*, if* $r = \hat{\delta}(q, \Lambda)$ *implies* $f(\Lambda) \in \Delta_a(r)$*. Thus, a strategy of a player is a function that for a finite history determines their next move. A vector* $(f_1, \ldots, f_K)$ *is a* strategy profile from state $q$*, if for each* $a \in \Omega$*,* $f_a$ *is a strategy from state* $q$ *for player* $a$*.*

The operator $\rhd$ allows us to change the strategy of a player in a strategy profile: if $F$ is a strategy profile from state $q$ and $f$ is a strategy from state $q$ for player $a$, $f \rhd_a F_{-a}$ is a strategy profile, where all players except $a$ play according to $F$ and player $a$ uses the strategy $f$. Similarly, if $j \in \Delta_a(q)$ is a move of player $a$ in state $q$ and $J \in \Delta(q)$ is a move vector, $j \rhd_a J_{-a}$ is a move vector with changed move for player $a$.

**Definition 5 (Outcome).** *The* outcome *is the function* $\lambda$ *from strategy profiles to computations, s. t. whenever* $F$ *is a strategy profile from a state* $q$*,* $\lambda(F)$ *is the infinite computation from* $q$ *s. t.* $\lambda(F)[i+1] = (F_1(\lambda(F)_i), \ \ldots, F_K(\lambda(F)_i))$*.*

*That is,* $\lambda(F)$ *is the computation where each step consists of individual moves of strategies from* $F$ *based on current history.*

**Definition 6 (Priced Concurrent Game Structures).** *A Priced Concurrent Game Structure (PCGS) is a tuple* $(K, Q, q_0, \Phi, \phi, \mathbb{M}, \Delta, \delta, \gamma)$ *with the following differences to CGS:*

- *The set of propositions* $\Phi$ *always includes* $g_1, \ldots, g_K$*, which are used to represent* goal states *for the respective players.*

– *The* price *function $\gamma$ assigns to each state $q$ and each move vector $J \in \Delta(q)$ a $K$-tuple of non-negative natural numbers which correspond to the price for each player. $\gamma_i$ denotes the $i$-th projection of $\gamma$, i.e. the price for player $i$.*

**Definition 7 (Cost).** *Given a PCGS $T$, the* cost function *is the function $\varGamma : Q \times (\mathbb{M}^K)^* \to \mathbb{N}^K_\infty$ such that for each player $a \in \varOmega$, state $q \in Q$ and computation $\varLambda$ from $q$,*

$$\varGamma_a(q, \varLambda) = \sum_{i=1}^{k} \gamma_a(\hat{\delta}(q, \varLambda_{i-1}), \varLambda[i]),$$

*where $k$ is the smallest position such that $g_a \in \phi(\hat{\delta}(q, \varLambda_k))$. If no such $k$ exists, $\varGamma_a(q, \varLambda) = \infty$.*

*We omit the state $q$ since it is usually clear from the context of the computation $\varLambda$ and write just $\varGamma(\varLambda)$.*

**Lemma 8.** *Let $\varLambda$ be a computation from $q$. If $g_a \in \phi(q)$, then $\varGamma_a(\varLambda) = 0$. Otherwise, $\varGamma_a(\varLambda) = \varGamma_a(\varLambda^1) + \gamma_a(q, \varLambda[1])$.*

**Definition 9 (Nash Equilibrium).** *Given a PCGS with initial state $q_0$, we say that strategy profile $F$ from state $q_0$ is a* Nash equilibrium*, if for each player $a \in \varOmega$ and for all strategies $f_a$ of player $a$,*

$$\varGamma_a(\lambda(f_a \rhd_a F_{-a})) \geq \varGamma_a(\lambda(F)).$$

*That is, no player can reduce their cost $\varGamma_a$ by changing their strategy.*

*We say that a Nash equilibrium $F$* satisfies bounds *$B \in \mathbb{N}^K_\infty$, if $B_a \geq \varGamma_a(\lambda(F))$.*

*Example 10.* Let us go back to the example in Fig. 1. Consider the following strategies $f_H, f_M, f_L$ for player 1 from $q_0$ (all possible strategies) and $f_1, f_2, f_3$ for player 2 from $q_0$ (3 out of 8 possible strategies). For any history $\varLambda \in (\mathbb{M}^2)^*$:

$$f_H(\varLambda) = \begin{cases} H & \text{if } \varLambda = \epsilon \\ O & \text{otherwise} \end{cases} \qquad f_1(\varLambda) = \begin{cases} O & \text{always} \end{cases}$$

$$f_M(\varLambda) = \begin{cases} M & \text{if } \varLambda = \epsilon \\ O & \text{otherwise} \end{cases} \qquad f_2(\varLambda) = \begin{cases} P & \text{if } \varLambda = (L, O) \\ O & \text{otherwise} \end{cases}$$

$$f_L(\varLambda) = \begin{cases} L & \text{if } \varLambda = \epsilon \\ O & \text{otherwise} \end{cases} \qquad f_3(\varLambda) = \begin{cases} P & \text{if } \varLambda = (L, O) \text{ or } (M, O) \\ O & \text{otherwise} \end{cases}$$

The outcome for the strategy profile $(f_M, f_2)$ is $\lambda(f_M, f_2) = (M, O)(O, O)^\omega$ (the dotted path). The cost of this outcome is $\varGamma(\lambda(f_M, f_2)) = (6, 2)$. The outcome for the strategy profile $(f_L, f_2)$ is $\lambda(f_L, f_2) = (L, O)(O, P)(O, O)^\omega$ (the dashed path). The cost of this outcome is $\varGamma(\lambda(f_L, f_2)) = (7, 3)$. Note that although the strategy for player 2 remains the same, their move is different in the second step.

Let us now examine some of the strategy profiles for equilibria. Profile $(f_M, f_2)$ is a Nash equilibrium as no player can reduce their cost. Particularly, if player

1 uses $f_L$, he gets lower cost for the first step, but suffers even worse penalty in the second step. Other equilibria include $(f_L, f_1)$, $(f_M, f_1)$, and $(f_L, f_3)$. Profiles $(f_M, f_1)$ and $(f_M, f_3)$ are not Nash equilibria. Player 1 may switch to $f_L$ without any penalty. Profile $(f_H, f_3)$ is not a Nash equilibrium. While switching to $f_M$ does not do player 1 any good, switching to $f_L$ yields an immediate benefit that is greater than the received penalty from player 2.

*Problem 11 (Nash equilibria problem).* Given a PCGS and bound vector $(b_1 \dots b_K) \in \mathbb{N}_\infty^K$, characterize all pure-strategy Nash equilibria $F$ satisfying bounds $(b_1 \dots b_K)$.

*Problem 12 (Decision variant of Nash equilibria problem).* Given a PCGS and a bound vector $(b_1 \dots b_K) \in \mathbb{N}_\infty^K$, decide whether there is a Nash equilibrium $F$ satisfying bounds $(b_1 \dots b_K)$. If all $b_a$ are fixed to $\infty$, i.e. we decide whether there is some Nash equilibrium in general, we refer to this problem as the Decision variant of Nash equilibria problem without bounds.

We might want to consider only equilibria where each player reaches their goal, i. e. where each cost is finite. We would still want to be able to limit the individual costs.

*Problem 13 (Decision variant of Nash equilibria with finite costs problem).* Given a PCGS and a bound vector $(b_1 \dots b_K) \in \mathbb{N}_\infty^K$, decide whether there is a Nash equilibrium $F$ satisfying bounds $(b_1 \dots b_K)$, in which each cost is finite.

## 3 Temptation and Punishment

When looking for Nash equilibria, we are looking neither for best strategies, nor for any competition. The players are not opponents, but collaborators. A Nash equilibrium is a stable mutual cooperation, where no player is tempted to defect. The cooperation need not be the most effective one. In games with multiple equilibria, we can often find multiple or even infinite number of stable cooperations that are strictly worse than other stable cooperations (such as the equilibrium $(f_L, f_3)$ in the example from Fig. 1 with cost $(7, 3)$).

The *temptation* is the best possible result a player can achieve by defecting a cooperation from a particular transition. Without temptation, all cooperations would be Nash equilibria because players would have no incentive to defect. We assume that players can agree on any outcome that is not jeopardized by a better temptation for one of the players.

Reactive games such as CGS allow strategies to detect a defection and adjust their behaviour. However, the players always provide their whole strategy in advance. Therefore the defecting adversary is able to adjust to the cooperating players' *punishment*. The punishment are the moves of the coalition that they had decided to use after they detect a betrayal. The defecting player can take advantage of that and choose a way of defection that guarantees the best outcome, given the future punishment.

A strategy can never detect and punish a defection in advance. The reason for this is that a strategy is determined only by a history of moves. As long as the defector stays true to the deal, the others must be using the same moves.

On the other hand, the punishing coalition's strategies do not care about their own cost anymore. After the alliance is broken by the traitor, all they are trying to do is make things miserable for the traitor. Such behaviour might seem counter-intuitive at first, but it makes sense when we reconsider the very purpose of such punishment. It exists to minimize temptation, and never has to occur when the players keep the deal which they have no incentive of breaking.

Let us introduce punishment values $\Pi$ and temptation values $\tau$. First of all, we state our requirements for these values and show an easy example. Then we provide an algorithm for computing punishment and temptation values based on relations between them and finally we formally prove that the computed values correspond to our requirements. The *punishment value* $\Pi_a(q)$ for player $a$ and

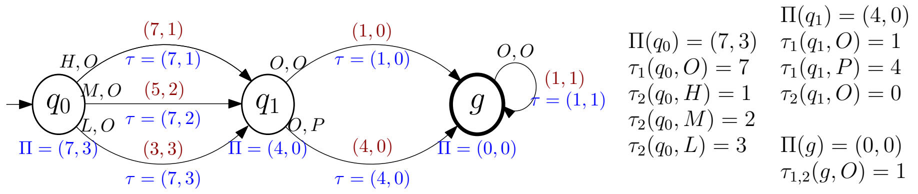


**Fig. 2.** PCGS from Fig. 1 with punishment and temptation values

state $q$ is the cost of the worst outcome for player $a$ which the coalition $\Omega \setminus \{a\}$ can enforce, starting in $q$. In other words, coalition $\Omega \setminus \{a\}$ has a strategy profile $F_{-a}$ to guarantee that the cost for player $a$ from $q$ will be at least $\Pi_a(q)$.

*Temptation value* $\tau_a(q, J_{-a})$ for player $a$, state $q$ and move $J_{-a}$ of the coalition $\Omega \setminus \{a\}$ is the cost of the best outcome for player $a$ starting from $q$, provided that $\Omega \setminus \{a\}$ use $J_{-a}$ as their first move and they are commited to their strategy. In other words, if players $\Omega \setminus \{a\}$ use profile $F_{-a}$ from $q$, starting with $J_{-a}$, player $a$ has a strategy to guarantee that their cost from $q$ will be at most $\tau_a(q, J_{-a})$. For the sake of simplicity, we use notation $\tau_a(q, J) = \tau_a(q, J_{-a})$.

Our algorithm is based on the following relations between punishment and temptation values:

$$\tau_a(q, J_{-a}) = \min_{j \in \Delta_a(q)} \gamma_a(q, j \rhd_a J_{-a}) + \Pi_a(\delta(q, j \rhd_a J_{-a}))$$

$$\text{if } g_a \notin \phi(q): \qquad \Pi_a(q) = \max_{J \in \Delta_{-a}(q)} \tau_a(q, J)$$

$$\text{if } g_a \in \phi(q): \qquad \Pi_a(q) = 0$$

It starts with an initial assignment of punishment values to $\infty$ for non-goal states and to 0 for goal states of a player. Then, in every iteration, it updates all the values according to these equations until reaching a fixed point. We can prove that no more than $|Q|$ iterations are needed and therefore the algorithm operates in polynomial time.

In the following lemmas consider $\Pi, \tau$ to be values computed by the algorithm, i.e. values satisfying the above equations. Here we show that they accord with their meaning provided in the first paragraphs.

**Lemma 14 (Punishment Lemma).** *For each state $q$ of a PCGS $T$, there is a strategy profile $F_{-a}$ from $q$ for coalition $\Omega \setminus \{a\}$, such that for each strategy $f$ from $q$ of player $a$, $\Gamma_a(\lambda(f \rhd_a F_{-a})) \geq \Pi_a(q)$.*

**Lemma 15 (Temptation Lemma).** *For each state $q$ and move $J_{-a} \in \Delta_{-a}(q)$ and for each strategy profile $F_{-a}$ from $q$ for coalition $\Omega \setminus \{a\}$ starting with the move $J_{-a}$, there is a strategy $f$ from state $q$ of player $a$, such that $\Gamma_a(\lambda(f \rhd_a F_{-a})) \leq \tau_a(q, J_{-a})$.*

## 4 Equilibrium Automaton

**Theorem 16.** *For a PCGS $T$, the set of all outcomes of Nash equilibria satisfying bounds $B \in \mathbb{N}_\infty^K$ is an $\omega-$regular language.*

We proceed with constructing a Büchi automaton accepting exactly the set of all equilibria outcomes. The idea of the construction is that we enhance states with local bounds for each player. Bounds $X = (x_1 \ldots x_K)$ in state $(q, X)$ mean that the cost of any infinite computation $\Lambda$ from $q$ represented by a run from $(q, X)$ must satisfy $X$, i.e. $\Gamma_a(\Lambda) \leq x_a$. We are also allowed to say that we no longer care about the cost for player $a$ from this state and let $x_a = \bot$.

Whenever there is a transition from $q$ to $r$ with cost $C$, there should be a transition from $(q, X)$ to $(r, Y')$, where $Y' = (y'_1 \ldots y'_K)$ and $y'_a = x_a - C_a$. However, whenever $q$ is a goal state for player $a$, then instead the local bound $y'_a$ for $a$ is set to $\bot$, because the cost of this run for $a$ has already been determined. This alone would allow us to observe the global bounds $B$.

On the other hand, we also have to account for the temptations of players to defect a potential equlibrium. Whenever we want to agree on a move $J$ with temptation $\tau_a(q, J)$, then the cost of the rest of the outcome for player $a$ must be lower than or equal to this temptation. Otherwise, player $a$ would defect the cooperation in this transition. Therefore we also update local bounds in $r$ to $y''_a = \tau_a(q, J) - C_a$.

We are interested in the lower of the two bounds $y', y''$. Thus finally,

$$y_a = \min(x_a, \tau_a(q, J)) - \gamma_a(q, J)$$

Note that such transition function guarantees that on any run of an equilibrium automaton, the local bound for any player $a$ is nonincreasing.

Additionally, whenever $y_a$ is lower than 0, we omit that transition. The transition function for the Equilibrium automaton is deterministic, but not total.

A choice of accepting states reflects the local bounds. If local bound $x_a$ is $\perp$, the cost is finite and respects the bounds. However, a computation $\Lambda$ which never reaches a state with goal $g_a$ has cost $\Gamma_a(\Lambda) = \infty$. Such computation can only be an equilibrium if local bounds for all states are $\infty$. Otherwise, there is a temptation for player $a$ to defect. Therefore we allow $\infty$ as a local bound for accepting states.

**Definition 17 (Equilibrium automaton $\mathcal{T}$).** *For a PCGS $T$ and $B \in \mathbb{N}_\infty^K$, the* Equilibrium automaton for $T$ and bounds $B$ *is the Büchi automaton* $(\Sigma, \mathcal{Q}, \delta', \{(q_0, B)\}, \mathcal{F})$ *with the following components:*

- Alphabet $\Sigma = \bigcup_{q \in Q} \Delta(q)$, *the set of all move vectors.*
- *The* state set $\mathcal{Q} = Q \times P_1 \times \ldots \times P_K$, *where $P_a$ denotes the set $\{\perp, \infty\} \cup \{0, 1 \ldots p_a\}$. If $B_a \neq \infty$, $p_a = B_a$, otherwise $p_a$ equals to the highest punishment value $\Pi_a$ for player $a$ lower than $\infty$.*
- *The* partial transition function $\delta' : \mathcal{Q} \times \Sigma \to \mathcal{Q}$, *defined as follows. For each state $q$ and local bounds $X = (x_1 \ldots x_K)$, let $Y = (y_1 \ldots y_K)$, s. t.*

$$y_a = \begin{cases} \perp & \textit{if } x_a = \perp \textit{ or } g_a \in \phi(q), \\ \min(x_a, \tau_a(q, J)) - \gamma_a(q, J) & \textit{otherwise.} \end{cases}$$

$$\textit{Then } \delta'((q, X), J) = \begin{cases} (\delta(q, J), Y) & \textit{if } y_a = \perp \textit{ or } y_a \geq 0 \textit{ for all } a, \\ \textit{undefined} & \textit{otherwise.} \end{cases}$$

- *The* initial state set $\{(q_0, B)\}$, *the initial state of $T$ augmented with bounds $B$.*
- *The* accepting state set $\mathcal{F} = Q \times \{\perp, \infty\}^K$.

**Lemma 18 (Correspondence).** *Let $T$ be a PCGS, $L_T$ be the language of all outcomes $\lambda(F)$, such that $F$ is a Nash equilibrium satisfying bounds $B \in \mathbb{N}_\infty^K$, and let $\mathcal{T}$ be an Equilibrium automaton for $T$ and bounds $B$. Then $\mathcal{L}(\mathcal{T}) = L_T$.*

*Proof.* The $\subseteq$ direction is given by Lemma 21, the $\supseteq$ is given by Lemma 22.

Theorem 16 is a corollary of the previous Lemma.

*Example 19.* In Fig. 3 we provide a second example which is not turn-based and includes cycles. On the left side there is PCGS $T$ with temptation and punishment values according to the previous section. The cost of each transition is $(1, 1)$, except for the transition $O, O$ from $q_k$ which is $(0, 1)$. On the right side, there is an Equilibrium automaton for $T$ and bounds $(\infty, \infty)$.

**Lemma 20.** *Let $\mathcal{T}$ be the Equilibrium automaton for a PCGS $T$ and bounds $B \in \mathbb{N}_\infty^K$. Then, if $\mathcal{T}$ has an accepting run $\rho$ over word $\Lambda$, then $\Lambda$ is a computation on $T$ starting in $q_0$ and the state component of the extended state always corresponds to the state of the computation, i.e. for $\rho(i) = (q_i, x_1^i \ldots x_K^i)$, $q_i = \hat{\delta}(q_0, \Lambda_i)$. Furthermore, $\rho$ satisfies local bounds in each state, i.e. for each player $a$, either $x_a^i = \perp$, or $\Gamma_a(\Lambda^i) \leq x_a^i$.*

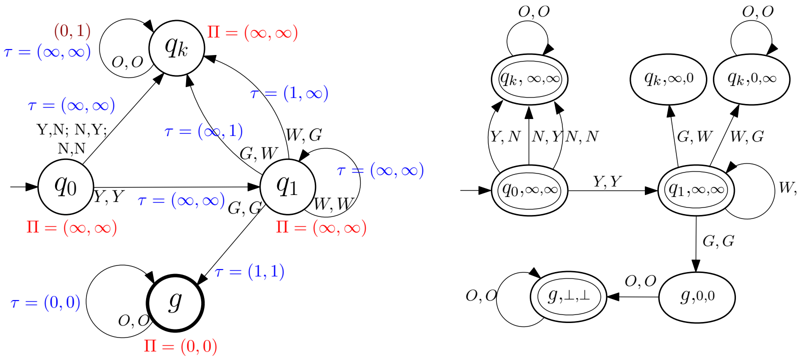


**Fig. 3.** Construction of the Equilibrium automaton for bounds $(\infty, \infty)$

**Lemma 21.** *Let $\mathcal{T}$ be the Equilibrium automaton for a PCGS $T$ and bounds $B \in \mathbb{N}_\infty^K$. If $\mathcal{T}$ accepts $\Lambda$, there exists a strategy profile $F$ from $q_0$, such that $\Lambda$ is an outcome $\lambda(F)$ and $F$ is a Nash equilibrium satisfying $B$.*

*Proof.* Firstly, we construct strategy profile $F = (f_1 \dots f_K)$ s. t. $\Lambda = \lambda(F)$. These strategies follow $\Lambda$ but as soon they detect a defection, they employ a punishing strategy according to the Punishment Lemma 14. Then we show that $F$ is a Nash equilibrium by contradiction. Assuming that player $a$ can reduce their cost by changing to $f'_a$, we find the last state $q_{i-1}$ before the defection and refer to the Punishment Lemma for the next state to show that the new cost for $a$ from $q_{i-1}$ is at least $x_a^{i-1}$. However, since the original cost for $a$ is at most $x_a^{i-1}$ thanks to Lemma 20, we reach a contradiction with the improvement of the cost.

Finally $F$ satisfies bounds $B$, as $B$ are local bounds for state $q_0$ and Lemma 20 gives an upper bound for the cost of $\lambda(F)$. □

**Lemma 22.** *For each strategy profile $F$ that is a Nash equilibrium on a PCGS $T$ satisfying bounds $B$, its outcome is in the language of $\mathcal{T}$, the Equilibrium automaton for $T$ and $B$. That is, $\lambda(F) \in \mathcal{L}(\mathcal{T})$.*

*Proof.* If $\Lambda$ is not accepted by $\mathcal{T}$, we find the last index $k$ s. t. condition $x_a^i = \perp$ or $\Gamma_a(\Lambda^i) \leq x_a^i$ is satisfied for each $i \leq k$. If such index does not exist because the run is infinite but the condition is always satisfied, then the cost for some player is $\infty$ which implies that the condition is not satisfied in some state (a contradiction). If such index does not exist because the condition is never satisfied, then the equilibrium does not meet the bounds.

Otherwise, we show that $\Lambda[k]$ is a move with a low temptation value for player $a$ and according to the Temptation Lemma 15, we can find a strategy $f$ defecting in this move, resulting in cost lower than the original. Thus, $F$ is not Nash equilibrium. □

The Equilibrium automaton provides a representation of all solutions to Problems 11 and 12. We can easily modify the automaton to solve Problem 13 by limiting the set of accepting states to those where all local bounds are $\perp$, as this corresponds to runs where each player reaches their goal. This also allow us to compute all Pareto optimal equilibria: the set of bounds $(b_1, \ldots, b_k) \in \mathbb{N}^k$ satisfied by a Nash equilibrium is upwards closed. Having just presented the solution to Problem 13 we can apply the result of Valk and Jantzen [8] for computing the finite (due to Dickson's lemma) minimal such bounds:

**Theorem 23.** *The set of Pareto optimal bounds $(b_1, \ldots, b_k) \in \mathbb{N}^k$ satisfied by a Nash equilibrium can be computed.*

## 5 Complexity of the Decision Variant

Consider the decision variant of Nash equilibria problem. With the construction of the Equilibrium automaton $\mathcal{T}$ for PCGS $T$ and bounds $B$, the problem is reduced to deciding the existence of an accepting run in the Büchi automaton. Although the size of the automaton is possibly exponential, we present the following result:

**Theorem 24.** *Decision variant of Nash equilibria problem is NP-complete.*

First we focus on showing that the problem is solvable in NP. The idea is that instead of constructing the Equilibrium automaton, we nondeterministically guess an accepting lasso in the automaton. We then verify the lasso in time linear to its length using the transition rules. The following two lemmas show that a lasso of polynomial length is sufficient for this.

**Definition 25 (Relation $\succsim$).** *States $X = (q_X, x_1 \ldots x_K), Y = (q_Y, y_1 \ldots y_K)$ are in relation $X \succsim Y$ iff $q_X = q_Y$ and for each player $a$, either $x_a = y_a = \perp$, or $x_a \geq y_a > \perp$. We say that $X$ is no more strict than $Y$.*

**Lemma 26.** *Let $X, Y$ be two states of the equilibrium automaton $\mathcal{T}$ such that $X \succsim Y$. Then, for every computation $\Lambda$ from $q$, whenever there is a run $\rho_Y$ from state $Y$ over $\Lambda$, then there also exists a run $\rho_X$ from $X$ over $\Lambda$. Furthermore, $\rho_X(i) \succsim \rho_Y(i)$ for all $i$.*

*Proof.* As all local bounds are lower in $Y$, whenever there is a transition from $Y$, the conditions of the transition function also hold in $X$. Furthermore, end states of the respective transitions are also in $\succsim$, so we can use induction. □

**Lemma 27.** *Given a PCGS with $K$ players and $|Q|$ states, if there is an accepting run in $\mathcal{T}$, there is also an accepting run which is a lasso of length at most $(K+2)|Q|$.*

*Proof.* Let $\rho$ be the shortest accepting lasso. We first show that the length of the cycle is at most $|Q|$. As the local bounds are nonincreasing, the values of the local bounds must be the same on all states on the cycle (either $\infty$ or $\perp$). Therefore they differ only in the base state. For cycle longer than $|Q|$ we find a repeating state and create a shorter accepting cycle, reaching a contradiction.

Now we show that the length of the nonrepeating path is at most $(K+1)|Q|$. If it is longer, some base state must repeat more than $K+1$ times. A local bound for player $a$ can change to $\perp$ only once. Therefore between at least one pair of those repetitions, no local bound changes to $\perp$. The previous lemma gives us a run $\rho'$ from the first of those states. Now take the first state $X$ of the cycle on $\rho$. Since all local bounds are either $\perp$ or $\infty$, $\rho'$ leads to this exact state and continues on the same accepting cycle. Joining $\rho'$ and the path to $X$ skips the steps between its one repetition, which contradicts that $\rho$ is the shortest. $\square$

**Lemma 28.** *Decision variant of Nash equilibria problem is NP-hard.*

*Proof.* We show a reduction from the subset sum problem [4], i.e. for every input instance of the subset sum problem, we construct a game structure and bounds, such that there is a Nash equilibrium meeting these bounds iff the input instance of the subset sum problem has a solution.

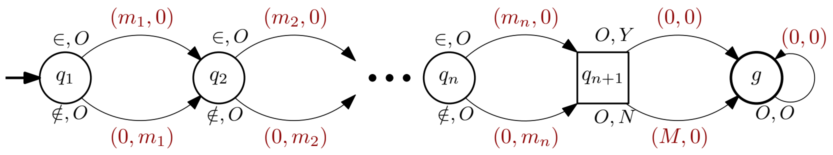


**Fig. 4.** Reduction from the subset sum problem

Let $S = \{m_1, m_2, \ldots, m_n\}$ be a set of positive integers and $m$ be the target sum. The input instance $(S, m)$ has a solution iff there exists a set $S' \subseteq S$, such that sum of the numbers of $S'$ is exactly $m$. Let $M = \sum_{s \in S} s$, the sum of all numbers. We construct a two-player turn-based game $G$ according to the Fig. 4. The initial state is $q_1$, $g$ is a goal state for both players and numbers above the transitions represent the prices. In circle (resp. square) states, player 1 (resp. player 2) chooses the next move. We set the bounds $(b_1, b_2) = (m, M-m)$.

For the first direction, suppose there is a solution to the subset sum problem $S'$. Now consider the following strategies for the players.

**Player 1.** In $q_i (1 \leq i \leq n)$, choose $\in$ if $m_i \in S'$, otherwise choose $\notin$.
**Player 2.** In $q_{n+1}$, choose $Y$ if the accumulated costs so far are $(m, M-m)$, otherwise choose $N$.

Outcome of these strategies has costs $(m, M-m) = (b_1, b_2)$. If player 1 changes his strategy such that the costs are different in $q_{n+1}$, his cost increases by $M > m$ in the last transition and thus she can not improve her cost. Player 2 can not influence her cost at all. The strategies are Nash equilibrium.

Now for the second direction, suppose there exists a Nash equilibrium meeting the bounds $(b_1, b_2)$. As the sum of the costs for both players is at least $m_1 + \ldots + m_n = M = m + (M-m) = b_1 + b_2$, the cost is exactly $(b_1, b_2)$. We consider the outcome and construct the set $S'$ as the set $\{m_i \mid \text{player 1 chooses the} \in \text{in } q_i\}$. Since the cost for player 1 is $m$, the sum of $S'$ must be exactly $m$ and it is a solution to the subset sum problem. $\square$

Theorem 24 is a corollary of the previous lemma and Lemma 27. We now show that the problem of deciding the existence of an equilibrium point is NP-hard even if we have no bounds on the possible equilibria.

**Theorem 29.** *Decision variant of Nash equilibria problem without bounds is NP-complete.*

*Proof.* Given a two-player PCGS $T$ and bounds $(b_1, b_2)$ we construct a two-player PCGS $T'$ according to Fig. 5. The new initial state is $q_0'$ and the state $g$ is a goal state of both players. We need to prove that there is an equilibrium in $T'$ iff there is an equilibrium in $T$ satisfying the bounds $(b_1, b_2)$.

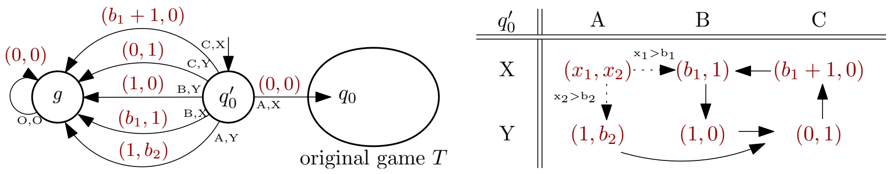


**Fig. 5.** Reduction from the problem with bounds $(b_1, b_2)$ to the problem without bounds

None of the added edges can be part of an equilibrium outcome as can be seen in the Table of choices in Fig. 5. The horizontal arrows indicate improvement for player 1, the vertical for player 2. Every equilibrium in $T$ satisfying bounds $(b_1, b_2)$ is preserved in $T'$, but the equilibria not satisfying the bounds are suppressed, because both players could change their first move and get a better cost. □

Lemma 28 shows NP-hardness even for turn-based games. However, [3] shows that for a special kind of games which roughly correspond to turn-based games where cost for each transition and player is 1, a Nash equilibrium always exists. Without elaborating on this further, we can confirm their result for any priced turn-based game with strategies that use history. Hence, the decision problem for this case is trivial and the answer is always positive.

## 6 Conclusion

We introduced the Nash equilibrium problem with bounds for priced concurrent games structures and provided a construction of a Büchi automaton accepting the set of all equilibria outcomes, characterizing the class of all Nash equilibria outcomes satisfying a bound vector as an $\omega$-regular set. We provided examples of similar problems that can be solved using our characterization, including finding equilibria satisfying LTL properties and describing pareto-optimal equilibria.

We also characterized the complexity of the decision variant of the problem as NP-complete. The problem remains NP-complete even if we consider either turn-based PCGS, or we omit bounds. If we do both, the problem becomes trivial as the equilibrium always exists.

## References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49, 672–713 (2002)
2. Bouyer, P., Brenguier, R., Markey, N.: Nash Equilibria for Reachability Objectives in Multi-player Timed Games. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 192–206. Springer, Heidelberg (2010)
3. Brihaye, T., Bruyère, V., De Pril, J.: Equilibria in Quantitative Reachability Games. In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 72–83. Springer, Heidelberg (2010)
4. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to algorithms, vol. 7, pp. 1162–1171. MIT Press and McGraw-Hill Book Company (1976)
5. Felegyhazi, M., Hubaux, J., Buttyan, L.: Nash equilibria of packet forwarding strategies in wireless ad hoc networks. IEEE Transactions on Mobile Computing, 463–476 (2006)
6. Nash, J.: Equilibrium points in n-person games. Proceedings of the National Academy of Sciences of the United States of America 36(1), 48–49 (1950)
7. Ummels, M., Wojtczak, D.: The Complexity of Nash Equilibria in Limit-Average Games. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 482–496. Springer, Heidelberg (2011)
8. Valk, R., Jantzen, M.: The residue of vector sets with applications to decidability problems in Petri nets. In: Application and Theory of Petri Nets, pp. 234–258 (1984)

# Computing by Observing Insertion

Alexander Krassovitskiy and Peter Leupold[*]

Research Group on Mathematical Linguistics – GRLMC
Rovira i Virgili University, Tarragona, Spain
`akrassovitskiy@gmail.com, Peter.Leupold@web.de`

**Abstract.** Computing by Observing is a theoretical model for computation that tries to formalize the standard setup of experiments in natural sciences. We establish that insertion systems with empty contexts and only one inserted letter suffice in this architecture to accept all recursively enumerable languages. While so far in most cases context-free power was needed, here a sub-regular system leads to computational completeness in this context. Further, we investigate more complicated insertion systems in a model with less powerful observer called Observing Change.

## 1 Computing by Observing

Much of the recent work in Formal Language Theory has stood in some relation to biochemistry. The original motivation for this were hopes of building actual biocomputers based on the theoretical models that have been developed. Nearly all of these models in the area of DNA computing follow the classical computer science paradigm of processing an input directly to an output, which is the result of the computation. Only the mechanisms of processing are different from conventional models; instead of a finite state control or a programming language it is biomolecular mechanisms that are used, or rather abstractions of such mechanisms.

However, in many experiments in biology and chemistry the setup is fundamentally different. The matter of interest is not some product of the system but rather the change over time observed in certain, selected quantities. To cite two simple examples that might be well-known from High School biology and chemistry classes:

– The *predator-prey relationship.* It is not of much use to know the numbers of predators and prey in one single moment. The interesting feature here is how the increase or decrease in one of the two populations affects the other population.
– A chemical *reaction with a catalyst* often has the same product as without, but the energy curves during the reactions are different. In the presence of a catalyst the reaction will occur with less energy consumption and possibly also faster.

[*] Peter Leupold's work was done while he was funded as a Beatriu de Pinós researcher by the Departament d'Universitats, Recerca i Societat de la Informació de la Generalitat de Catalunya under contract number 2009 BP-B 00052.

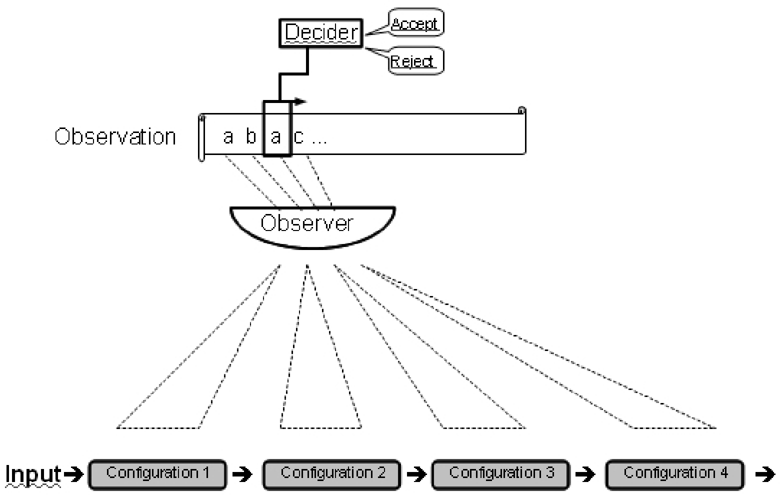


**Fig. 1.** The architecture of an accepting observer system [6]

The objective of Computing by Observing was to formalize this approach of investigators who have been dealing with biological and chemical systems for ages in a paradigm for computation. The resulting architecture consists of an underlying system, which evolves in discrete steps from one configuration to the next. The second element is an observer, which reads these configurations and transforms them into single letters; a type of classification, if we take the finite number of letters to represent a finite number of classes. In this way, a sequence of configurations is transformed into a simple sequence of symbols, i.e. a string. This corresponds to the protocol of an experiment in biology or chemistry, and for us it is the main result of the computation. Figure 1 depicts this setup, where acceptance is decided based on this *observation*.

In the initial work on the topic, membrane systems were used as underlying systems [4]. Then plain string-rewriting systems came into the focus [5,6]; because of their simplicity and generality they seemed suited for identifying key features of the underlying systems that would be crucial for obtaining higher computational power. At the same time, several types of models of bio-computing were investigated as underlying systems, namely splicing systems [3] and sticker systems [2]. The main results of these investigations presented a common pattern: systems that by themselves characterize the context-free languages together with regular observers result in computational completeness.

From the numerous DNA-inspired models of computation [14,8], here we investigate insertion systems as underlying systems in the Computing by Observing architecture. Insertion is the operation of inserting a new factor between specified left and right contexts in a string. Motivated from linguistics, systems of rules of this type were first investigated by Galiukschov [9]. Later, more intensive investigations resulted from the fact that insertion occurs in DNA strands. Thus the power of this operation in the field of DNA computing was investigated, mostly in combination with the complementary operation of deletion [10,12]. An up-to-date and complete overview of work on these systems can be found in the recent dissertation by one of the current authors [13].

Since every insertion necessarily increases a string's length, the generative power of insertion systems is limited; in some sense they have no auxiliary memory available. However, via morphisms that delete some auxiliary symbols, all recursively enumerable languages can be obtained from languages generated by insertion systems, whose rules have inserted strings, left and right contexts all of length two. Systems with smaller rules are less powerful.

In the light of the results mentioned above, our main result is slightly surprising: while so far, with the exception of sticker systems, context-free power was necessary in the underlying systems to achieve computational completeness, we show that insertion systems of sub-regular power suffice for this purpose. In fact, the insertion systems we use are the simplest ones possible: they insert one letter independent of the context. Here the observer's implicit ability to verify the correct context for an insertion combines very efficiently with the simple insertion rules.

The fact that computational completeness is obtained already with the simplest insertion systems suggests using less powerful models of observation. One such model is *observing change* [7], where the observer sees only which type of change (i.e. rule) has occurred, but does not see the entire configuration. We establish that such systems are significantly less powerful than the ones with omnivident observers.

## 2 Insertion Observer Systems

Now we proceed to define first the two main components that will make up our observer systems. Then we formally describe their interplay. The reader is assumed familiar with standard concepts from Formal Language Theory such as languages and finite automata; for more details standard textbooks can be consulted [15]. Note that we denote the empty string by $\lambda$.

### 2.1 Insertion Systems

We now formally specify what insertion systems are and how they implement the insertions described informally above.

**Definition 1.** *An insertion system is a triple* $(\Sigma, A, R)$*, where* $\Sigma$ *is an alphabet, the set of* axioms $A$ *is a finite language over* $\Sigma$*, and the set of* insertion rules $R$ *is a finite set of triples of the form* $(u, \alpha, v)$*, where* $u$*,* $\alpha$*, and* $v$ *are strings over* $\Sigma$*,* $\alpha \neq \lambda$*.*

An insertion rule $(u, \alpha, v) \in R$ indicates that the string $\alpha$ can be inserted between $u$ and $v$; the latter two are therefore called *contexts*. Stated otherwise, $(u, \alpha, v) \in R$ corresponds to the rewriting rule $uv \to u\alpha v$. We denote by $\Rightarrow$ the relation defined by an insertion rule. Formally, $x \Rightarrow y$ iff $x = x_1uvx_2, y = x_1u\alpha vx_2$, for some $(u, \alpha, v) \in R$ and $x_1, x_2 \in \Sigma^*$. We denote by $\Longrightarrow^*$ the reflexive and transitive closure of $\Rightarrow$, and $\Rightarrow^+$ denote its transitive closure. The language generated by the system is defined by

$$L = \{w \in \Sigma^* : x \Rightarrow^* w, x \in A\}.$$

In what follows, we will mainly use very simple insertion systems, namely ones that do not use contexts and insert only one letter at a time. To facilitate comparison later on, we start by characterizing their generative power.

**Lemma 2.** *Every language over the alphabet $\Sigma$ generated by an insertion system with insertion rules that have empty contexts and insert only one letter is a finite union of languages of the form $X^*x_1X^*x_2X^* \cdots X^*x_nX^*$ where for some $n \geq 0$ we have $x_1, x_2, \ldots, x_n \in \Sigma$ and $X \subseteq \Sigma$.*

*Proof.* We take a look at what kinds of words an insertion system $(\Sigma, A, R)$ can generate. We divide the letters from the alphabet $\Sigma$ into two classes: $X$ is the set $\{x : (\lambda, x, \lambda) \in R\}$, $Y$ is the rest of letters. Obviously, letters from $Y$ can only occur in a word that is generated by the system, if they were already present in the axiom. On the other hand, letters from $X$ can be inserted in arbitrary numbers at arbitrary places. Thus each axiom generates a language of the form in the lemma's statement. □

## 2.2 Observers

The observer's task is mapping a string to a single symbol. As in all the preceding work on observer systems, where the configurations were strings, we will use monadic transducers for this. They are simply finite automata that additionally have an output function $\phi$ that outputs one letter depending on the state the transducer stops in.

**Definition 3.** *A* monadic transducer *is a tuple $[Q, \Sigma, \Gamma, \delta, q_0, \phi]$ where the state set $Q$, the alphabet $\Sigma$, the transition function $\delta$, and the initial state $q_0$ are the same as for deterministic finite automata. $\Gamma$ is the* output alphabet*, and $\phi$ is the output function, a mapping $Q \mapsto \Gamma \cup \{\lambda\}$ which assigns an output letter or the empty word to each state.*

## 2.3 Insertion Observer Systems

Combining the devices introduced in the preceding definitions we now give a formal description of the setup depicted in Figure 1.

**Definition 4.** *An* insertion observer system *is a quintuple $\Omega = [\Delta, R, \mathcal{O}, D, \odot]$, where $\mathcal{O}$ is a monadic transducer, $R$ is an insertion system over the input alphabet $\Sigma$ of $\mathcal{O}$, the system's input alphabet $\Delta$ is a subset of $\Sigma$, and $D$ is a regular language over the output alphabet of $\mathcal{O}$; all words of $D$ end in the* termination symbol *$\odot$.*

The main difference to general accepting observer systems is the termination symbol $\odot$. So far, the observations always ended when the underlying system's computation terminated. However, insertion systems often do no terminate. Especially systems with very simple rules will run forever: if there is an insertion rule without context, it will always be applicable; also rules with just one symbol of context on just one side will always be applicable, if they have been applicable once — the context symbol will never disappear. This is why we introduce this explicit way of stopping the observation.

**Definition 5.** *The language accepted by an insertion observer system $\Omega = [\Delta, R, \mathcal{O}, D, \odot]$ is the set of all words $w \in \Delta^*$ such that there exists a derivation sequence $s : w \Rightarrow w_1 \Rightarrow \cdots \Rightarrow w_n$ with insertion rules from $R$, whose observation is in the language $D$, i.e. $\mathcal{O}(w) \cdot \mathcal{O}(w_1) \cdots \mathcal{O}(w_n) \in D$, where $w_n$ must either be a string to which no rule of $R$ can be applied, or it must be equal to $\odot$ with $\odot$ not appearing before in $s$..*

## 3 Computing by Observing Insertion Systems

In order to illustrate the ideas behind the definitions of Section 2 we start out with an example. Already this example will show that the combination of simple components can lead to considerable computational power.

*Example 6.* With a very simple insertion system we can generate the language $\{a^n b^n c^n : n \geq 0\}$, which is not context-free. More specifically, all the rules have empty contexts and insert only one letter each. The insertion rules that are used are the ones in the set

$$\{(\lambda, A, \lambda), (\lambda, \overline{A}, \lambda), (\lambda, B, \lambda), (\lambda, \overline{B}, \lambda), (\lambda, C, \lambda), (\lambda, \overline{C}, \lambda)\}.$$

In combination with the observer, these rules are used to implement the following algorithm:

(i). Check, if the input word is from the language $a^+b^+c^+$.
(ii). Insert an $A$ after the first letter $a$ that is not followed by $A$.
(iii). Insert a $B$ after the first letter $b$ that is not followed by $B$.
(iv). Insert a $C$ after the first letter $c$ that is not followed by $c$.
(v). Insert an $\overline{A}$ after the first letter $A$ that is not followed by $\overline{A}$.
(vi). Insert a $\overline{B}$ after the first letter $B$ that is not followed by $\overline{B}$.
(vii). Insert a $\overline{C}$ after the first letter $C$ that is not followed by $\overline{C}$.
(viii). If there is some lower case letter left that is not followed by the corresponding upper case letter: go back to Step (ii).
(ix). Accept the input word.

This algorithm recognizes the language $\{a^n b^n c^n : n \geq 0\}$ by first checking the sequence of letters in the word, i.e. there should only be $a$ followed by a block of $b$ followed by a block of $c$. This is done implicitly by the observer. Steps (ii) – (vii) in some sense mark one each of the letters $a$, $b$ and $c$ by inserting their upper case counterparts just after them. Thus, if after some iteration there is no unmarked letter left, we can conclude that the input was from $\{a^n b^n c^n : n \geq 0\}$. This is again done by the observer which will end the observation by outputting $\odot$. Its mapping is:

$$\mathcal{O}(w) = \begin{cases} 1 & \text{if } w \in (aA\overline{A})^* a^+ (bB\overline{B})^* b^+ (cC\overline{C})^* c^+, \\ 2 & \text{if } w \in (aA\overline{A})^* aAa^* (bB\overline{B})^* b^+ (cC\overline{C})^* c^+, \\ 3 & \text{if } w \in (aA\overline{A})^* aAa^* (bB\overline{B})^* bBb^* (cC\overline{C})^* c^+, \\ 4 & \text{if } w \in (aA\overline{A})^* aAa^* (bB\overline{B})^* bBb^* (cC\overline{C})^* cCc^*, \\ 5 & \text{if } w \in (aA\overline{A})^+ a^* (bB\overline{B})^* bBb^* (cC\overline{C})^* cCc^*, \\ 6 & \text{if } w \in (aA\overline{A})^+ a^* (bB\overline{B})^+ b^* (cC\overline{C})^* cCc^*, \\ \odot & \text{if } w \in (aA\overline{A})^+ (bB\overline{B})^+ (cC\overline{C})^+, \\ \perp & \text{else.} \end{cases}$$

The observations 1 to 6 correspond to the six steps (ii) – (vii) from the algorithm above. Now the deciders task consists in checking whether these steps have always been executed in the correct order. To this end the decider accepts the language $(123456)^+\odot$.

This example already exhibits the key technique that we will use further down. Obviously, insertion rules cannot delete or change any symbols. Thus, what forms part of the string at some point will stay in the string until the end of the computation. However, for computing it is usually convenient to discard information that will not be used anymore. Obviously this makes it easier to find the relevant pieces of information. What we did in Example 6 with the symbols that had already been counted was to put a "complementary" symbol behind them to signal that they had no relevance for the remainder of the computation. This complementing instead of rewriting allowed the observer to identify the place where the computation was proceeding: the only place where uncomplemented $A$, $B$ and $C$ appeared.

**Theorem 7.** *Every recursively enumerable language can be accepted by an insertion observer system with insertion rules that have empty contexts and insert only one letter.*

*Proof.* We will simulate an off-line Turing Machine $\mathcal{M}$ with one working tape and with transition function $\delta : Q \times \Phi \to Q \times \Phi \times \{+, -\}$, where $Q$ is the set of states, $\Phi$ is the tape alphabet, and $+$ or $-$ denote a move to the right or left, respectively. Every transition in $\delta$ is associated to an unique label $t$, and $T$ is the set of all these labels. The special letter $\square \in \Phi$ denotes an empty tape cell. An input string is accepted, if $\mathcal{M}$ stops in the final state $q_f \in Q$. Its initial state $q_0 \in Q$ appears at the beginning of input string in the initial configuration.

We denote by $\widehat{T} = T \cup \{\boxplus, \boxminus, \oplus, \ominus, \vdash\}$. $\overline{Q}$ and $\overline{\Phi}$ are new alphabets with overlined symbols $\overline{a}, a \in Q$, and $a \in \Phi$, correspondingly and by $\Sigma$ the alphabet $\Sigma = Q \cup \Phi \cup \overline{Q} \cup \overline{\Phi} \cup \widehat{T}$. We denote by $S^m$ and $S^f$ the following regular expressions $S^m = (Q\overline{Q} \cup \Phi\overline{\Phi})^*$, $S^f = (\Phi \cup S^m)^*$, respectively.

Now we construct the insertion observer system $\Omega = (T, R, \mathcal{O}, D)$ that simulates $\mathcal{M}$, where the set of insertion rules $R$ contains the following rules: $R =$

$\{(\lambda, a, \lambda) : a \in \Sigma\}$. The output alphabet for the observer $\mathcal{O}$ consists of the union of the sets $\{I_1, I_2, \odot, \perp\}$, $\{T_{t,l} : 1 \leq l \leq 6, t \in T\}$, $\{T_{i,l} : 1 \leq l \leq 8, i \in 1, \dots, |Q|\}$ and $\{T_l : 9 \leq l \leq 14\}$. Its mapping for the initialization phase is:

$$\mathcal{O}(w) = \begin{cases} I_1, & \text{if } w \in \Phi^*, \\ I_2, & \text{if } w \in q_0\Phi^*. \end{cases}$$

For every transition $t : (q_i, a) \xrightarrow{\delta} (q_j, b, +)$ (which performs the rewriting step $w'q_iaw'' \to w'bq_jw''$, $w', w'' \in \Phi^*$) the observer is defined by the mapping:

$$\mathcal{O}(w) = \begin{cases} T_{t,1}, & \text{if } w \in t\widehat{T}^* S^f q_i a S^f, \\ T_{t,2}, & \text{if } w \in t\widehat{T}^* S^f q_i a q_j S^f, \\ T_{t,3}, & \text{if } w \in t\widehat{T}^* S^f q_i a \overline{a} q_j S^f, \\ T_{t,4}, & \text{if } w \in t\widehat{T}^* S^f q_i a \overline{a} b q_j S^f, \\ T_{t,5}, & \text{if } w \in t\widehat{T}^* S^f q_i \overline{q_i} a \overline{a} b q_j S^f, \\ T_{t,6}, & \text{if } w \in \vdash t\widehat{T}^* S^f q_i \overline{q_i} a \overline{a} b q_j S^f. \end{cases}$$

Similarly, for every transition $t : (q_i, a) \xrightarrow{\delta} (q_j, b, -)$ (which performs rewriting $w'xq_iaw'' \to w'q_jxbw''$, $x \in \Phi$, $w', w'' \in \Phi^*$) we define

$$\mathcal{O}(w) = \begin{cases} T_{t,1}, & \text{if } w \in t\widehat{T}^* S^f \Phi q_i a S^f, \\ T_{t,2}, & \text{if } w \in t\widehat{T}^* S^f q_j \Phi q_i a S^f, \\ T_{t,3}, & \text{if } w \in t\widehat{T}^* S^f q_j \Phi q_i a \overline{a} S^f, \\ T_{t,4}, & \text{if } w \in t\widehat{T}^* S^f q_j \Phi q_i b a \overline{a} S^f, \\ T_{t,5}, & \text{if } w \in t\widehat{T}^* S^f q_j \Phi q_i \overline{q_i} b a \overline{a} S^f, \\ T_{t,6}, & \text{if } w \in \vdash t\widehat{T}^* S^f q_j \Phi q_i \overline{q_i} b a \overline{a} S^f. \end{cases}$$

We can assume that for every transition $t : (q_i, a) \xrightarrow{\delta} (q_j, b, \mu)$ we have $q_i \neq q_j$. Hence, the strings of the form $S^f q_i S^m q_i S^f$ may be used in order to "replace" $q_i$ over marked symbols. We do so by inserting a new copy of $q_i$ and then by marking the old symbol $q_i$. Formally, for every state $q_i \in Q$ the observer $\mathcal{O}(w)$ is defined by the following mapping:

$$\mathcal{O}(w) = \begin{cases} T_{i,1}, & \text{if } w \in \boxplus\widehat{T}^* S^f q_i S^m S^f, \\ T_{i,2}, & \text{if } w \in \boxplus\widehat{T}^* S^f q_i S^m q_i S^f, \\ T_{i,3}, & \text{if } w \in \boxplus\widehat{T}^* S^f q_i \overline{q_i} S^m q_i S^f, \\ T_{i,4}, & \text{if } w \in \vdash \boxplus\widehat{T}^* S^f q_i \overline{q_i} S^m q_i S^f, \\ T_{i,5}, & \text{if } w \in \boxminus\widehat{T}^* S^f S^m q_i S^f, \\ T_{i,6}, & \text{if } w \in \boxminus\widehat{T}^* S^f q_i S^m q_i S^f, \\ T_{i,7}, & \text{if } w \in \boxminus\widehat{T}^* S^f q_i S^m q_i \overline{q_i} S^f, \\ T_{i,8}, & \text{if } w \in \vdash \boxminus\widehat{T}^* S^f q_i S^m q_i \overline{q_i} S^f, \end{cases}$$

where $T_{i,1}, T_{i,2}, T_{i,3}, T_{i,4}$ are used to replace $q_i$ to the right and $T_{i,5}, T_{i,6}, T_{i,7}, T_{i,8}$ are used to replace $q_i$ to the left.

Insertion of the blank tape symbol $\Box$ is controlled by the following clauses:

$$\mathcal{O}(w) = \begin{cases} T_9, & \text{if } w \in \oplus\widehat{T}^* S^f Q, \\ T_{10}, & \text{if } w \in \oplus\widehat{T}^* S^f Q\Box, \\ T_{11}, & \text{if } w \in \vdash \oplus\widehat{T}^* S^f Q\Box, \\ T_{12}, & \text{if } w \in \ominus\widehat{T}^* Q(\Phi \cup Q)S^f, \\ T_{13}, & \text{if } w \in \ominus\widehat{T}^* \Box Q(\Phi \cup Q)S^f, \\ T_{14}, & \text{if } w \in \vdash \ominus\widehat{T}^* \Box Q(\Phi \cup Q)S^f, \end{cases}$$

where the first three conditions are used to insert the blank at the end while the remaining three conditions are used to insert the blank at the beginning of the factor in $QS^f$. In the latter case, we additionally require that the first state symbol in $Q$ has not been marked (hence it is followed by either symbol in $\Phi$ or in $Q$).

Finally, the observer $\mathcal{O}(w)$ is defined for the final configuration of $\mathcal{M}$ and those computations that have not been considered.

$$\mathcal{O}(w) = \begin{cases} \odot, & \text{if } w \in \widehat{T} S^f q_f S^f, \\ \perp, & \text{else.} \end{cases}$$

All the languages, i.e. types of configurations, described in the observer's clauses above are disjoint. Therefore we can take the union of all of the parts defined above to obtain the complete observer.

The decider $D$ mainly has to guarantee that always one transition of the Turing Machine at a time is simulated completely. To this end it accepts the language

$$\begin{aligned} D = I_1 I_2 \Big( & \bigcup_{t \in T} \; T_{t,1}T_{t,2}T_{t,3}T_{t,4}T_{t,5}T_{t,6} \\ & \bigcup_{i \in \{1,\ldots,|Q|\}} (T_{i,1}T_{i,2}T_{i,3}T_{i,4} \cup T_{i,5}T_{i,6}T_{i,7}T_{i,8}) \\ & \bigcup \; T_9 T_{10} T_{11} \cup T_{12} T_{13} T_{14} \Big)^* \odot . \end{aligned}$$

The prefix of computed strings over $\widehat{T}$ stores a *history* of computation steps performed by observer system in the form $(\vdash (\widehat{T} \setminus \vdash))^*$, where $t \in T \subset \widehat{T}$ means corresponding instruction performed by $\mathcal{M}$; symbols $\boxplus, \boxminus \in \widehat{T}$ are used for right and left moves of $q_0$ over marked symbols; and $\oplus, \ominus \in \widehat{T}$ are used for insertion of blank cell at the end of the string or at the beginning of the factor in $QS^f$. We use these letters over $\widehat{T}$ in order to indicate which computation part performs the observer. Insertion of $\vdash$ means that the observer is ready for the next computation part.

The computation of $\Omega$ realizes the following algorithm:

(1) Insert $q_0$ at the beginning of the input string.
(2) Go randomly to Step (3), (4) or (5) (guess next operation).
(3) Simulate a rule $t : (q_i, a) \xrightarrow{\delta} (q_j, b, \mu), \mu \in \{+, -\}$.
  – Insert symbol $t$ at the beginning of the string.
  – Insert new state symbol $q_j$.

  – Mark symbol $a$.
  – Insert symbol $b$.
  – Mark $q_i$.
  – Insert $\vdash$ .
  – Go to (6).

(4) "Replace" state symbol $q_i$ to the left or to the right over marked symbols
  – Insert $\boxplus$ (or $\boxminus$) at the beginning.
  – Insert a copy of $q_i$ to the right(to the left) over marked symbols, correspondingly.
  – Mark the old $q_i$.
  – Insert $\vdash$ .
  – Go to (6).

(5) Insert a blank cell.
  – Insert $\oplus$(or $\ominus$) at the beginning.
  – Insert the blank at the end (or after $\widehat{T}^*$, at the beginning of factor in $QS^f$, correspondingly).
  – Insert $\vdash$ .

(6) If there is no symbol $q_f$ then go back to Step (2).

(7) Accept the input word.

The very specific form of the observer's clauses guarantees that the order of these steps is followed. Every simulation of a step of the computation runs through a sequence of configurations that belong to clauses specific to only that step. Thus for every string accepted by $\mathcal{M}$ there is an accepting computation of $\Omega$. Indeed, consider the computation of $\mathcal{M}$, Then the corresponding computation of the observer system mimics it by the algorithm shown above. Step (4) is performed whenever the replacement of $q_i$ over marked symbols needed, while step (5) inserts blanks.

In order to show that for every word accepted by $\Omega$ there an accepting computation of $\mathcal{M}$ we note that the decider $D$ is defined in such a way that it allows to consider only those computations of $\Omega$ that are given by the algorithm. □

So already the simplest possible form of insertion rules suffices to obtain computational completeness in the framework of insertion observer systems. This suggests considering variants of observer systems with less powerful modes of observation. First, however, we want to remark how big the leap in computational power in Theorem 7 was. Looking at the power of the corresponding class of insertion systems by themselves, we observe that they only generate the very simple class of subregular languages described in Lemma 2.

## 4 Observing only Change

Section 3 has shown us that even the simplest insertion systems suffice to obtain computational completeness in the Computing by Observing framework. One essential point for this was the observer's power to see and process the entire configuration in each step. This is a very strong assumption; as there is no bound on the size of a configuration, such an observation might take a very long time. Therefore variants with less powerful observers have been investigated, too. We take a look at what was called *observing change* [7]: the observer does not see the configuration, but rather sees what has happened. In our case, this means that the observation consists in the rule that has been applied, without any knowledge as to where it has been applied.

**Definition 8.** *An* insertion change-observing acceptor *is a quintuple $\Omega = [\Delta, R, \mathcal{O}, D, \odot]$, where all components are the same as in Definition 4 except for the observer. It is not a monadic transducer but a function from $R$ into the alphabet of $D$.*

We refrain from giving a formal definition of the mode of operation here, because the functioning is very straightforward to understand after the treatment of insertion observer systems in the preceding sections. With such an insertion change-observing acceptor we now recognize the same language as in Example 6. We will see how the knowledge about positions of insertions must now be coded in the rules' contexts, because the observer cannot do this part of the job anymore.

*Example 9.* The key observation for this example is an alternative characterization of words from the language $\{a^n b^n c^n : n \geq 0\}$ based on the possible factors of length two that such words can have. It is relatively straight-forward to see that the following three conditions characterize exactly the words from our language:

(i). There is exactly one occurrence of each $ab$ and $bc$.
(ii). The numbers of $aa$, $bb$ and $cc$ are equal.
(iii). Other blocks of two letters ($ac$, $ba$, $ca$, $cb$) do not occur.

For every possible factor of two letters we use one insertion rule to mark it: for $x, y \in \{a, b, c\}$ we use the rule $(x, \triangle, y)$ and give it the label $x_y$; we use the labels as observations for better readability. Thus between any two input letters the symbol $\triangle$ can be inserted. As a result, the two letters are not adjacent anymore. Thus every factor of length two leads to exactly one rule application. An input string $x_1 x_2 \ldots x_{n-1} x_n$ is converted into the string $x_1 \triangle x_2 \triangle \cdots \triangle x_{n-1} \triangle x_n$.

In this way, the sequence of rules can be used to count the numbers of occurrences of the different types of factors. With the decider $a_b b_c (a_a b_b c_c)^*$ on the labels of rules we validate exactly the characterization from above. Note that the fact that the derivation has to terminate also guarantees that none of the forbidden factors occurs.

While this example shows that some non-trivial languages can be generated, we do not achieve computational completeness as above. In fact, even some finite languages cannot be accepted.

**Proposition 10.** *The singleton language $\{a^2 ba\}$ cannot be accepted by any insertion change-observing acceptor with insertion rules that have left and right contexts of length at most one each and insert an arbitrary number of letters.*

*Proof.* Let us look at a factor $xy$ of an input string. Whatever is inserted between $x$ and $y$ during a derivation does not at all depend on the parts of the string left of $x$ and right of $y$. Consequently, for a change-observing acceptor it is irrelevant where in the string $xy$ is located. Every such pair generates a sequence of rule applications, and the possible sequences for the entire input string depend only on the numbers of pairs of different types. This means that the strings $a^2ba$ and $aba^2$ generate exactly the same sequences of rule applications. This means that in any language accepted by a change-observing acceptor as specified above there are either both of these strings or none. □

The proof shows that the main limitation of power comes from the fact that it is impossible to distinguish applications of the same rule in different places. To some extent this could be compensated by using more context in the rules. This is just what took us from Example 6 to Example 9, where in the latter the same language is accepted as in the former by using some context. Based on prior experience with graph-controlled insertion systems [1] we conjecture the following:

*Conjecture 11.* Insertion change-observing acceptors with insertion rules that have left and right contexts of length two each and insert strings of length two can accept all recursively enumerable languages.

In the reference cited above, the *mark and migration* technique as used by Kari and Sosík [11] was combined with a graph control of derivations. In this way, insertion rules as in the conjecture sufficed to achieve computational completeness. Since the graph control is essentially based on admitting only certain sequences of rules, very similar arguments should work for our acceptors.

## 5 Conclusions

Theorem 7 has shown that insertion systems are very well-suited for the use in the Computing by Observing architecture: even sub-regular systems suffice to accept all recursively enumerable languages. As the systems used are the simplest ones possible, this leaves few open questions.

On the other hand, the results from Section 4 provide a host of open questions: the most obvious one is the explicitly formulated Conjecture 11. If it turns out to be true, then the next question is its optimality. That is, can computational completeness also be achieved with rules of smaller sizes? And can the language classes that are accepted with insertion rules of smaller size be characterized in nice ways?

## References

1. Alhazov, A., Krassovitskiy, A., Rogozhin, Y., Verlan, S.: Small Size Insertion and Deletion Systems. In: Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory. Scientific Applications of Language Methods, vol. 2, pp. 459–524. World Scientific (2010)

2. Alhazov, A., Cavaliere, M.: Computing by Observing Bio-systems: The Case of Sticker Systems. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 1–13. Springer, Heidelberg (2005)
3. Cavaliere, M., Jonoska, N., Leupold, P.: Recognizing DNA splicing. Natural Computing 9(1), 157–170 (2009)
4. Cavaliere, M., Leupold, P.: Evolution and Observation: A New Way to Look at Membrane Systems. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 70–87. Springer, Heidelberg (2004)
5. Cavaliere, M., Leupold, P.: Evolution and observation — a non-standard way to generate formal languages. Theoretical Computer Science 321, 233–248 (2004)
6. Cavaliere, M., Leupold, P.: Observation of string-rewriting systems. Fundamenta Informaticae 74(4), 447–462 (2006)
7. Cavaliere, M., Leupold, P.: Computing by Observing Changes. In: Peper, F., Umeo, H., Matsui, N., Isokawa, T. (eds.) IWNC 2009. PICT, vol. 2, pp. 133–140. Springer, Heidelberg (2010)
8. Dassow, J., Mitrana, V., Salomaa, A.: Operations and language generating devices suggested by the genome evolution. Theor. Comput. Sci. 270(1-2), 701–738 (2002)
9. Galiukschov, B.S.: Semicontextual grammars. Matem. Logica i Matem. Lingvistika, 38–50 (1981) (in Russian)
10. Kari, L., Păun, G., Thierrin, G., Yu, S.: At the crossroads of DNA computing and formal languages: Characterizing RE using insertion-deletion systems. In: Proceedings of 3rd DIMACS Workshop on DNA Based Computing, Philadelphia, pp. 318–333 (1997)
11. Kari, L., Sosík, P.: On the weight of universal insertion grammars. Theoretical Computer Science 396(1-3), 264–270 (2008)
12. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. Information and Computation 131(1), 47–61 (1996)
13. Krassovitskiy, A.: Complexity and Modeling Power of Insertion-Deletion Systems. Ph.D. thesis, Universitat Rovira i Virgili, Tarragona (2011)
14. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing – New Computing Paradigms. Springer, Heidelberg (1998)
15. Salomaa, A.: Formal Languages. Academic Press, New York (1973)

# On the Parameterized Complexity of Default Logic and Autoepistemic Logic[⋆]

Arne Meier[1], Johannes Schmidt[2], Michael Thomas[3,⋆⋆], and Heribert Vollmer[1]

[1] Universität Hannover, Germany
{meier,vollmer}@thi.uni-hannover.de
[2] Université de la Méditerranée, France
johannes.schmidt@lif.univ-mrs.fr
[3] TWT GmbH, Germany
michael.thomas@twt-gmbh.de

**Abstract.** We investigate the application of Courcelle's Theorem and the logspace version of Elberfeld et al. in the context of the implication problem for propositional sets of formulae, the extension existence problem for default logic, as well as the expansion existence problem for autoepistemic logic and obtain fixed-parameter time and space efficient algorithms for these problems.

On the other hand, we exhibit, for each of the above problems, families of instances of a very simple structure that, for a wide range of different parameterizations, do not have efficient fixed-parameter algorithms (even in the sense of the large class $\mathrm{XP}_{\mathrm{nu}}$), unless $\mathrm{P} = \mathrm{NP}$.

## 1 Introduction

Non-monotonic reasoning formalisms were introduced in the 1970s as a formal model for human reasoning and have developed into one of the most important topics in computational logic and artificial intelligence. However, as it turns out, most interesting reasoning tasks are computationally intractable already for propositional versions of non-monotonic logics [7], in fact presumably much harder than for classical propositional logic. Because of this, a lot of effort has been spent to identify fragments of the logical language for which at least some of the algorithmic problems allow efficient algorithms; a survey of this line of research can be found in [13].

In this paper a different approach is chosen to deal with hard problems, namely the framework of parameterized complexity. Gottlob et al. [8] made it clear that the tree width of a (suitable graph theoretic encoding of a) given knowledge base is a useful parameter in this context: making use of Courcelle's Theorem it was shown that many reasoning tasks for logical formalisms such as circumscription, abduction and logic programming become tractable in the case of bounded tree

⋆ Work supported by DFG grant VO 630/6-2.

⋆⋆ Work performed while employed at the Gottfried Wilhelm Leibniz Universität, Appelstr. 4, 30167 Hannover, Germany.

width. Here we examine a family of non-monotonic logics where the semantics of a given set of formulae (axioms, knowledge base) is defined in terms of a fixed-point equation. In particular we turn to default logic [11] and autoepistemic logic [9]. In the first, human reasoning is mimicked using so called "default rules" (in the absence of contrary information, assume this and that); in the second, a modal operator is introduced to model the beliefs of a perfect rational agent. For both logics the algorithmic tasks of satisfiability and reasoning have been shown to be complete in the second level of the polynomial hierarchy [7].

Much in the vein of [8] we here examine the parameterized complexity of these problems and, making again use of Courcelle's Theorem and a recent improvement by Elberfeld et al., we obtain time and space efficient algorithms if the tree width of the given knowledge base is bounded. This proves once again how important this parameter is.

A second contribution of our paper concerns lower bounds: Under the assumption $\mathrm{P} \neq \mathrm{NP}$ we show that, even for certain families of very simple knowledge bases and for any parameterization taken from a broad variety, no efficient fixed-parameter algorithms exist, not even in the sense of the quite large parameterized class $\mathrm{XP}_{\mathrm{nu}}$. These simple families of knowledge bases are defined in terms of severe syntactic restrictions, e.g., using default rules with literals or propositions only. Restricting the input structure even further we obtain that no fixed-parameter algorithm in the sense of the space-bounded class $\mathrm{XL}_{\mathrm{nu}}$ (the logarithmic space analogue of $\mathrm{XP}_{\mathrm{nu}}$) exists, unless $\mathrm{L} = \mathrm{NL}$.

Unfortunately, tree width is not among the parameters for which our lower bound can be proven—otherwise we would have proven $\mathrm{P} \neq \mathrm{NP}$. In a third part of our paper, we show that those structurally very simple families of knowledge bases, for which we gave our lower bounds, already have unbounded tree width. For this result, we introduce the notion of pseudo-cliques and show how to embed these into our graph-theoretic encodings of knowledge bases.

Due to space reasons the proof of Theorem 13 has to be omitted and can be read in the CoRR version "Meier, A., Schmidt, J., Thomas, M., Vollmer, H.: *On the parameterized complexity of default logic and autoepistemic logic.* CoRR abs/1110.0623 (2011)".

## 2 Preliminaries

*Complexity Theory.* In this paper we will make use of several standard notions of complexity theory such as the complexity classes L, $\oplus$L, NL, P, NP, coNP, and $\Sigma_2^{\mathrm{p}}$ and their completeness notions under logspace-many-one $\leq_{\mathrm{m}}^{\log}$ reductions.

Given a problem $P$ and a parameterization $\kappa$, $(P, \kappa)$ belongs to the class FPT iff there is a deterministic algorithm solving $P$ in time $f(\kappa(x)) \cdot |x|^{O(1)}$; $(P, \kappa)$ is said to be *fixed parameter tractable* then. If $(P, \kappa)$ is a parameterized problem, then $(P, \kappa)_\ell := \{x \in P \mid \kappa(x) = \ell\}$ is the $\ell$-th slice of $(P, \kappa)$. Define $(P, \kappa)$ to be a member of $\mathrm{XP}_{\mathrm{nu}}$ (in words, XP nonuniform) iff $(P, \kappa)_\ell \in \mathrm{P}$ for all $\ell \in \mathbb{N}$. For background on parameterized complexity we recommend [6].

Furthermore, we require space parameterized complexity classes which have been defined in [12] recently. Given a parameterized problem $(P, \kappa)$, we say

$(P, \kappa) \in$ PLS iff there exists a deterministic algorithm deciding $P$ in space $O(\log \kappa(x) + \log |x|)$, $(P, \kappa) \in$ FPL iff there exists a deterministic algorithm deciding $P$ in space $O(\log f(\kappa(x)) + \log |x|)$ for some recursive funktion $f$, and $(P, \kappa) \in \mathrm{XL_{nu}}$ iff $(P, \kappa)_\ell \in$ L for all $\ell \in \mathbb{N}$. It holds that PLS $\subseteq$ FPL $\subseteq$ FPT $\subseteq$ $\mathrm{XP_{nu}}$ as well as FPL $\subseteq \mathrm{XL_{nu}} \subseteq \mathrm{XP_{nu}}$.

*Tree width.* A *tree decomposition* of a graph $G = (V, E)$ is a pair $(T, X)$, where $X = \{B_1, \ldots, B_r\}$ is a family of subsets of $V$ (the set of *bags*), and $T$ is a tree whose nodes are the bags $B_i$, satisfying the following conditions: *(i)* $\bigcup X = V$, i.e., every node appears in at least one bag, *(ii)* $\forall (u, v) \in E\ \exists B \in X$: $u, v \in B$, i.e., every edge is 'contained' in a bag, and *(iii)* $\forall u \in V$: $\{B \mid u \in B\}$ is connected in $T$, i.e., for every node $u$ the set of bags containing $u$ is connected in $T$.

The *width* of a decomposition $(T, X)$, *width*$(T, X)$, is the number $\max\{\,|B| \mid B \in X\} - 1$, i.e., the size of the largest bag minus 1. The *tree width* of a graph $G$, *tw*$(G)$, is the minimum of the widths of the tree decompositions of $G$.

*Propositional Logic.* Let $\varphi, \psi$ be propositional formulae. We say $\psi$ can be deduced from $\varphi$, in symbols $\varphi \models \psi$, if for every assignment $\theta$ such that $\theta \models \varphi$ it holds $\theta \models \psi$. Further if $A$ is a set of propositional formulae, then we define $\mathrm{Th}(A) := \{\psi \mid \varphi \models \psi, \varphi \in A\}$ as the set of all consequences of the set $A$.

*Default Logic.* Following Reiter [11], a *default rule* is a triple $\frac{\alpha:\beta}{\gamma}$; $\alpha$ is called the *prerequisite*, $\beta$ is called the *justification*, and $\gamma$ is called the *conclusion.* If $B$ is a set of Boolean functions, then $d = \frac{\alpha:\beta}{\gamma}$ is a $B$-default rule if $\alpha, \beta, \gamma$ are $B$-formulae, i.e., formulae that use only connectors for functions in $B$. A $B$-*default theory* $(W, D)$ consists of a set of propositional $B$-formulae $W$ and a set of $B$-default rules $D$.

Let $(W, D)$ be a default theory and $E$ be a set of formulae. Now define $\Gamma(E)$ as the smallest set of formulae such that *(i)* $W \subseteq \Gamma(E)$, *(ii)* $\Gamma(E)$ is closed under deduction, i.e., $\Gamma(E) = \mathrm{Th}(\Gamma(E))$, and *(iii)* for all defaults $\frac{\alpha:\beta}{\gamma} \in D$ with $\alpha \in \Gamma(E)$ and $\neg\beta \notin E$, it holds that $\gamma \in \Gamma(E)$. Then a *stable extension* of $(W, D)$ is a fix-point of $\Gamma$, i.e., a set $E$ such that $E = \Gamma(E)$.

A definition for stable extensions beyond fix-point semantics which was introduced by Reiter [11] uses the principle of a stage construction: for a given default theory $(W, D)$ and a set $E$ of formulae, define $E_0 = W$ and $E_{i+1} = \mathrm{Th}(E_i) \cup \{\gamma \mid \frac{\alpha:\beta}{\gamma} \in D, \alpha \in E_i \text{ and } \neg\beta \notin E\}$. Then $E$ is a *stable extension* of $(W, D)$ if and only if $E = \bigcup_{i \in \mathbb{N}} E_i$, and the set $G = \{\frac{\alpha:\beta}{\gamma} \in D \mid \alpha \in E \wedge \neg\beta \notin E\}$ is called the set of *generating defaults.*

The so to speak satisfiability problem for default logic, here called *extension existence problem*, Ext, is the problem, given a theory $(W, D)$, to decide if it has a stable extension. Gottlob [7] proved that this problem is $\Sigma_2^{\mathrm{p}}$-complete.

*Autoepistemic Logic.* Moore in 1985 introduced a new modal operator $L$ stating that its argument is "believed" as an extension of propositional logic [9]. Further the expression $L\varphi$ is treated as an atomic formula with respect to the consequence relation $\models$. Given a set of Boolean functions $B$, we define with $\mathcal{L}_{\mathrm{ae}}(B)$

the set of all autoepistemic $B$-formulae through $\varphi ::= p \mid f(\varphi, \dots, \varphi) \mid L\varphi$ for $f$ being a Boolean functions in $B$ and a proposition $p$. If $\Sigma \subseteq \mathcal{L}_{\mathrm{ae}}(B)$, then a set $\Delta \subseteq \mathcal{L}_{\mathrm{ae}}(B)$ is a *stable expansion* of $\Sigma$ if it satisfies the condition $\Delta = \mathrm{Th}(\Sigma \cup L(\Delta) \cup \neg L(\overline{\Delta}))$, where $L(\Delta) := \{L\varphi \mid \varphi \in \Delta\}$ and $\neg L(\overline{\Delta}) := \{\neg L\varphi \mid \varphi \notin \Delta\}$, and $L(\Delta), \neg L(\overline{\Delta}) \subseteq \mathcal{L}_{\mathrm{ae}}(B)$.

Let $\mathrm{SF}(\varphi)$ denote the set of subformulae of $\varphi$, let $\mathrm{SF}^L(\varphi)$ denote the set of those subformulae of $\varphi$ that have prefix $L$, and let us use the shorthand $\neg S = \{\neg\varphi \mid \varphi \in S\}$ for a set of (autoepistemic) formulae $S$. Given a set of autoepistemic $B$-formulae $\Sigma \subseteq \mathcal{L}_{\mathrm{ae}}(B)$, we say a set $\Lambda \subseteq \mathrm{SF}^L(\Sigma) \cup \neg\mathrm{SF}^L(\Sigma)$ is *$\Sigma$-full* if for each $L\varphi \in \mathrm{SF}^L(\Sigma)$ it holds $\Sigma \cup \Lambda \models \varphi$ iff $L\varphi \in \Lambda$, and $\Sigma \cup \Lambda \not\models \varphi$ iff $\neg L\varphi \in \Lambda$. The connection of $\Sigma$-full sets and stable expansions of $\Sigma$ has been observed by Niemelä [10]: if $\Sigma \subseteq \mathcal{L}_{\mathrm{ae}}$ is a set of autoepistemic formulae and $\Lambda$ is a $\Sigma$-full set, then for every $L\varphi \in \mathrm{SF}^L(\Sigma)$ either $L\varphi \in \Lambda$ or $\neg L\varphi \in \Lambda$. The stable expansions of $\Sigma$ and $\Sigma$-full sets are in one-to-one correspondence.

The *expansion existence problem*, Exp, is the problem, given a set of autoepistemic formulae $\Sigma$, to decide if it has a stable expansion. Again, Gottlob proved that this problem is complete for the class $\Sigma_2^{\mathrm{p}}$.

## 3 MSO-Encodings

The aim of this paper is the application of Courcelle's theorem for obtaining fixed-parameter algorithms in the context of default and autoepistemic logic. For this, we will have to describe the relevant decision problems by monadic second-order formulae. In this section we will explain how to do this and obtain a preliminary result for the implication problem. Our approach is similar to the one of Gottlob, Pichler, and Wei [8] where MSO encodings for algorithmic problems from logic programming, abduction, and circumscription were developed.

Now fix a finite set $B$ of Boolean functions. Denote by $\tau_B$ the vocabulary $\{\mathit{const}_f^1 \mid f \in B, \mathrm{arity}(f) = 0\} \cup \{\mathit{conn}_{f,i}^2 \mid f \in B, 1 \leq i \leq \mathrm{arity}(f)\}$. With respect to a set $\Gamma$ of propositional $B$-formulae we associate a $\tau_{B,prop}$-structure $\mathcal{A}_\Gamma$ where $\tau_{B,prop} := \tau_B \cup \{\mathit{var}^1, \mathit{repr}^1\}$ such that the universe of $\mathcal{A}_\Gamma$ is the set of subformulae of the formulae in $\Gamma$, and *(i)* $\mathit{var}(x)$ holds iff $x$ represents a variable, *(ii)* $\mathit{repr}(x)$ holds iff $x$ represents a formula from $\Gamma$, *(iii)* $\mathit{const}_f^1(x)$ holds iff $x$ represents the constant $f$, and *(iv)* $\mathit{conn}_{f,i}^2(x, y)$ holds iff $x$ represents the $i$th argument of the function $f$ at the root of the formula tree represented by $y$.

**Lemma 1.** *Let $B$ be a finite set of Boolean functions. Then there exists an MSO-formula $\theta_{sat}$ over $\tau_{B,prop}$ such that for any $\Gamma \subseteq \mathcal{L}(B)$ it holds that $\Gamma$ is satisfiable iff $\mathcal{A}_\Gamma \models \theta_{sat}$.*

*Proof.* The formula $\theta_{struc}$ defined as follows states that if an individual is not representing a formula $\varphi \in \Gamma$, then there must be at least one subformula in which it occurs. If an individual is not a variable, then it represents either a constant or a Boolean function $f \in B$ and needs to have corresponding arity $f$ individuals.

$$\theta_{struc} := \forall x\Big(\neg repr(x) \to \exists y\big(\neg var(y) \land \bigvee_{\substack{f\in B,\\ 1\le i\le \mathrm{arity}(f)}} conn_{f,i}(x,y)\big)\Big) \land \forall x\Big(\neg var(x) \to$$

$$\bigvee_{\substack{f\in B,\\ \mathrm{arity}(f)=0}} const_f(x) \oplus \bigvee_{f\in B} \bigwedge_{1\le i\le \mathrm{arity}(f)} \exists y\big(conn_{f,i}(y,x) \land \forall z(conn_{f,i}(z,x) \to z = y)\big)\Big).$$

Let $n$ denote the maximal arity of $B$, i.e., $n := \max\{\mathrm{arity}(f) \mid f \in B\}$.

$$\theta_{assign}(M) := \forall x \forall y_1 \cdots \forall y_n \bigwedge_{f\in B} \Big( \bigwedge_{\mathrm{arity}(f)=0} const_f(x) \to (M(x) \leftrightarrow f) \land$$

$$\bigwedge_{1\le i\le \mathrm{arity}(f)} conn_{f,i}(y_i, x) \to \big(M(x) \leftrightarrow f([\![y_1 \in M]\!], \ldots, [\![y_{\mathrm{arity}(f)} \in M]\!])\big)\Big),$$

where $[\![x \in M]\!]$ is $\top$ iff $x \in M$ holds and $\bot$ otherwise. Now define

$$\theta_{\exists assign} := \exists M\big(\theta_{assign}(M) \land \forall x\Big(repr(x) \to M(x)\Big)$$

It is easy to verify that $\theta_{sat} := \theta_{struc} \land \theta_{\exists assign}$ satisfies the lemma. □

Let $B$ be a finite set of Boolean functions and $F, G$ be sets of $B$-formulae. Answering the implication problem of sets of propositional formulae, i.e., the question whether $F \models G$, requires to extend our vocabulary $\tau_{B,prop}$ to $\tau_{B,imp} := \tau_{B,prop} \cup \{repr^1_{prem}, repr^1_{conc}\}$ as well as our structure which we will denote by $\mathcal{A}_{F,G}$: $repr_{prem}(x)$ is true iff $x$ represents a formula from $F$, and $repr_{conc}(x)$ is true iff $x$ represents a formula from $G$. Now it is straightforward to formalize implication.

**Lemma 2.** *Let $B$ be a finite set of Boolean functions. Then there exists an MSO-formula $\theta_{imp}$ over $\tau_{B,imp}$ such that for any $\Gamma \subseteq \mathcal{L}(B)$ and any $F, G \subseteq \Gamma$ it holds that $F \models G$ iff $\mathcal{A}_{F,G} \models \theta_{imp}$.*

*Proof.* Define the MSO-formulae $\theta_{premise}(M)$, $\theta_{conclusion}(M)$, and $\theta_{implies}$ as follows:

$$\begin{aligned}
\theta_{premise}(M) &:= \forall x(repr_{prem}(x) \to M(x))\\
\theta_{conclusion}(M) &:= \forall x(repr_{conc}(x) \to M(x)))\\
\theta_{implies} &:= \forall M\Big(\big(\theta_{assign}(M) \land \theta_{premise}(M)\big) \to \theta_{conclusion}(M)\Big)
\end{aligned}$$

Then, we can define the formula $\theta_{imp}$ as $\theta_{imp} := \theta_{struc} \land \theta_{implies}$, where $\theta_{struc}$ and $\theta_{assign}$ are defined as above in Lemma 1. □

The application of Courcelle's Theorem [3] and the logspace version of Elberfeld et al. [5] directly leads to the following theorem.

**Theorem 3.** *Let $B$ be a finite set of Boolean functions, let $k \in \mathbb{N}$ be fixed, and let $F, G$ be sets of $B$-formulae such that $\mathcal{A}_{F,G}$ has tree width bounded by $k$. Then the implication problem for sets of $B$-formulae is solvable in time $O(f(k)\cdot(|F|+|G|))$ and space $O(\log(f(k)) + \log(|F| + |G|))$.*

In other words, the implication problem of sets of formulae parameterized by the tree width of $\mathcal{A}_{F,G}$ is fixed-parameter tractable, and even in PLS. In the following sections we will extend this result to default logic and autoepistemic logic.

## 4 Default Logic

Let $B$ be a finite set of Boolean functions. Write $W \Cup D$ as a shorthand for the set of formulae $W \cup \{\alpha, \beta, \gamma \mid \frac{\alpha:\beta}{\gamma} \in D\}$. To any $B$-default theory $(W, D)$, we associate a $\tau_{B,dl} := \tau_{B,prop} \cup \{kb^1, def^1, prem^2, just^2, concl^2\}$-structure $\mathcal{A}_{(W,D)}$ such that the universe of $\mathcal{A}_{(W,D)}$ is the union of the set of subformulae of $W \Cup D \cup \{\neg\beta \mid \frac{\alpha:\beta}{\gamma} \in D\}$ together with a set corresponding to the defaults in $D$, the relations from $\tau_{B,prop}$ are interpreted as in Section 3, and $kb(x)$ holds iff $x$ represents a formula from the knowledge base $W$, $def(x)$ holds iff $x$ represents a default $d \in D$, $prem(x, y)$ (resp. $just(x, y)$, $concl(x, y)$) holds iff $x$ represents the premise $\alpha$ (resp. justification $\beta$, conclusion $\gamma$) and $y$ represents the rule $\frac{\alpha:\beta}{\gamma}$.

**Lemma 4.** *Let $B$ be a finite set of Boolean functions and let $(W, D)$ be a $B$-default theory. There exists an MSO-formula $\theta_{extension}$ such that $(W, D)$ possesses a stable extension iff $\mathcal{A}_{(W,D)} \models \theta_{extension}$.*

*Proof.* First the formula $\theta_{isneg}$ expresses the fact that one formula is the negation of another formula: $\theta_{isneg}(\varphi, \overline{\varphi}) := \theta_{struc} \wedge \forall M\Big(\theta_{assign}(M) \rightarrow \big(M(\varphi) \leftrightarrow \neg M(\overline{\varphi})\big)\Big)$. Observe that $\varphi$ and $\overline{\varphi}$ are not formulae but placeholders for individuals. The following two formulae define the applicability of defaults, i.e., whether a premise $\alpha$ is entailed or a justification $\beta$ is violated which uses the shortcut $\chi(C, M, x) := (kb(x) \vee C(x)) \rightarrow M(x)$:

$$\theta_{W\cup C\models\alpha}(C, \alpha) := \forall M\Big(\theta_{assign}(M) \rightarrow \forall x\Big(\chi(C, M, x) \rightarrow M(\alpha)\Big)\Big),$$

$$\theta_{W\cup C\models\neg\beta}(C, \beta) := \exists\overline{\beta}\exists M\Bigg(\theta_{assign}(M) \rightarrow \forall x\Big(\chi(C, M, x) \wedge M(\overline{\beta}) \wedge \theta_{isneg}(\beta, \overline{\beta})\Big)\Bigg).$$

Now we can define the MSO-formulae $\theta_{app}$ (a default $d$ is applicable), $\theta_{stable}$ (a set of defaults is stable), $\theta_{gd}$ (a set of defaults is generating) as follows.

$$\begin{aligned}\theta_{app}(d, G) :=\ & \exists\alpha\exists\beta\exists C\Big(prem(\alpha, d) \wedge just(\beta, d) \wedge \\ & \quad \forall x\big(C(x) \leftrightarrow \exists y(G(y) \wedge concl(x, y))\big) \wedge \theta_{W\cup C\models\alpha}(C, \alpha) \wedge \neg\theta_{W\cup C\models\neg\beta}(C, \beta)\Big) \\ \theta_{stable}(G) :=\ & \forall d\big(def(d) \wedge (G(d) \leftrightarrow \theta_{app}(d, G))\big) \\ \theta_{gd}(G) :=\ & \theta_{stable}(G) \wedge \forall G'(G' \subsetneq G \rightarrow \neg\theta_{stable}(G'))\end{aligned}$$

Then $\theta_{extension} := \theta_{struc} \wedge \exists G(\theta_{gd}(G))$ is true under $\mathcal{A}_{(W,D)}$ iff $(W, D)$ has a stable extension. □

As a consequence of Lemma 4, we obtain from Courcelle's Theorem [3] and the logspace version of Elberfeld et al. [5] that, given the tree width of $\mathcal{A}_{(W,D)}$ as a parameter, the extension existence problem for default logic is fixed-parameter tractable, and in fact, in PLS.

**Theorem 5.** *Let $B$ be a finite set of Boolean functions, let $k \in \mathbb{N}$ be fixed, and let $(W, D)$ be a $B$-default theory such that $\mathcal{A}_{(W,D)}$ has tree width bounded by $k$. Then the extension existence problem for $B$-default logic is solvable in time $O(f(k) \cdot |(W, D)|)$ and space $O(\log(f(k)) + \log |(W, D)|)$.*

So again and maybe with no big surprise, similar to the study by Gottlob et al. [8] for different nonmonotonic formalisms, we see here that bounding the tree width of a default theory yields time and space efficient algorithms for satisfiability. In the following we want to contrast this with a strong lower bound. We consider knowledge bases with very simple defaults rules, namely consisting only of literals (and in a second step even only propositions). Then we consider any parameterization of the extension existence problem that is bounded for all knowledge bases that obey this restriction. It follows that even for these very restricted knowledge bases, the parameterized extension existence problem is not even in the class $\mathrm{XP}_{\mathrm{nu}}$, unless $\mathrm{P} \neq \mathrm{NP}$.

We want to point out that this theorem comprises for example the usual parameterizations for SAT (in terms of, e.g., backdoor sets or formula tree width): For all these, we have FPT-algorithms for propositional satisfiability, but still the extension existence problem is not in $\mathrm{XP}_{\mathrm{nu}}$.

**Theorem 6.** *Let $B$ be a finite set of Boolean functions such that $\neg \in [B \cup \{\top\}]$ and let $\mathbf{D}$ be the set of sets $D$ of default rules such that each default $d \in D$ is composed of literals only. Further let $\kappa$ be a parameterization function for which there exists a $c \in \mathbb{N}$ such that $\kappa\big((\emptyset, D)\big) < c$ for all $D \in \mathbf{D}$. If $\mathrm{P} \neq \mathrm{NP}$, then the extension existence problem for $B$-default logic, parameterized by $\kappa$, is not contained in $\mathrm{XP}_{\mathrm{nu}}$.*

*Proof.* The reduction from SAT to default logic restricted to default theories with $W = \emptyset$ and default rules composed of literals only, shown in Lemma 5.6 of [2], proves that the extension existence problem of default logic restricted to theories of this kind (which will be denoted by $\textsc{Ext}'$) is NP-hard. Now let $\kappa$ be such a parameterization and suppose $\mathrm{P} \neq \mathrm{NP}$. For contradiction assume $(\textsc{Ext}', \kappa) \in \mathrm{XP}_{\mathrm{nu}}$. Hence, by definition of $\mathrm{XP}_{\mathrm{nu}}$, it holds $(\textsc{Ext}', \kappa)_\ell \in \mathrm{P}$ for every $\ell \in \mathbb{N}$. As also $\ell < c$ holds we can compose a deterministic polynomial time algorithm which solves $\textsc{Ext}'$. This contradicts $\mathrm{P} \neq \mathrm{NP}$ and concludes the proof. □

**Theorem 7.** *Let $B$ be a finite set of Boolean functions such that $\bot \in [B]$ and let $\mathbf{D}$ be the set of sets $D$ of default rules such that each default $d \in D$ is composed of propositions or the constant $\bot$ only. Further let $\kappa$ be a parameterization function for which there exists a $c \in \mathbb{N}$ such that $\kappa\big((W, D)\big) < c$ for all $D \in \mathbf{D}$ and all $W$ that consists of at most one proposition. If $\mathrm{L} \neq \mathrm{NL}$, then the extension existence problem for $B$-default logic, parameterized by $\kappa$, is not contained in $\mathrm{XL}_{\mathrm{nu}}$.*

*Proof.* The reduction from GAP to default logic restricted to default theories with $|W| \leq 1$ and default rules composed of propositions or the constant $\bot$ only, shown in Lemma 5.8 of [2], proves that the extension existence problem of default logic restricted to theories of this kind (which will be denoted by $\textsc{Ext}'$) is NL-hard. Following the argumentation in the proof of Theorem 6, we conclude for $\mathrm{L} \neq \mathrm{NL}$ and $(\textsc{Ext}', \kappa) \in \mathrm{XL}_{\mathrm{nu}}$ that $(\textsc{Ext}', \kappa)_\ell \in \mathrm{L}$ holds for every $\ell$. This eventually leads to the desired contradiction proving the theorem. □

## 5 Autoepistemic Logic

Let $B$ be a finite set of Boolean functions. To any set $\Sigma$ of autoepistemic $B$-formulae, we associate a $\tau_{B,ae} := \tau_B \cup \{L^1, repr^1\}$-structure $\mathcal{A}_\Sigma$ such that the universe of $\mathcal{A}_\Sigma$ is the union of the set of subformulae of $\Sigma \cup \{\neg L\varphi \mid L\varphi \in \mathrm{SF}(\Sigma)\}$, the relations from $\tau_B$ are interpreted as in Section 3, and $L(x)$ holds iff the subformulae represented by $x$ is prefixed by an $L$, and $repr(x)$ holds iff $x$ represents a formula in $\Sigma$.

**Lemma 8.** *Let $B$ be a finite set of Boolean functions and let $\Sigma$ be a set of autoepistemic $B$-formulae. There exists an MSO-formula $\theta$ such that $\Sigma$ possesses a stable expansion iff $\mathcal{A}_\Sigma \models \theta$.*

*Proof.* For a set of formulae $G$ and a formula $\varphi$, similar to $\theta_{W\cup C\models\alpha}(C,\alpha)$ in the proof of Lemma 4, define be the MSO-formula

$$\theta_{\Sigma\cup\Lambda\models\varphi}(\Lambda,\varphi) := \forall M\Big(\theta_{assign}(M) \to \forall x\Big(\big((repr(x) \vee \Lambda(x)) \to M(x)\big) \to M(\varphi)\Big)\Big)$$

to test for $\Sigma \cup \Lambda \models \varphi$. Now define the MSO-formula $\theta_{full}$ as

$$\begin{aligned}\theta_{full}(\Lambda) :=& \forall x\Big(L(x) \to \big(\Lambda(x) \oplus \exists y(conn_\neg(x,y) \wedge \Lambda(y))\big)\Big) \wedge \\ & \forall x\Big(L(x) \to \big(\Lambda(x) \leftrightarrow \theta_{\Sigma\cup\Lambda\models\varphi}(\Lambda,x)\big)\Big)\end{aligned}$$

Then $\theta := \theta_{struc} \wedge \exists\Lambda(\theta_{full}(\Lambda))$ is true under $\mathcal{A}_\Sigma$ iff $\Sigma$ has a $\Sigma$-full set $\Lambda$, which is the case iff $\Sigma$ has a stable expansion. □

As above we obtain from Lemma 8 that, given the tree width of $\mathcal{A}_\Sigma$ as a parameter, the expansion existence problem for autoepistemic logic is fixed-parameter tractable, and in fact in PLS.

**Theorem 9.** *Let $B$ be a finite set of Boolean functions, let $k \in \mathbb{N}$ be fixed, and let $\Sigma$ be a set of autoepistemic $B$-formulae such that $\mathcal{A}_\Sigma$ has tree width bounded by $k$. Then the expansion problem is solvable in time $O(f(k) \cdot |\Sigma|)$ and space $O(\log(f(k)) + \log|\Sigma|)$.*

On the other hand, analogues of Theorems 6 and 7 are easily obtained:

**Theorem 10.** *Let $B$ be a finite set of Boolean functions such that $\vee \in [B \cup \{\bot, \top\}]$ and let $\mathbf{\Sigma}$ be the set of sets $\Sigma$ of autoepistemic $B$-formulae such that all $\varphi \in \Sigma$ are disjunctions of propositions or $L$-prefixed propositions. Further let $\kappa$ be a parameterization function for which there exists a $c \in \mathbb{N}$ such that $\kappa(\Sigma) < c$ for all $\Sigma \in \mathbf{\Sigma}$. If $\mathrm{P} \neq \mathrm{NP}$, then the expansion existence problem for sets of autoepistemic $B$-formulae, parameterized by $\kappa$, is not contained in $\mathrm{XP}_{\mathrm{nu}}$.*

*Proof.* Observe that there exists a reduction $f$ from 3-Sat to autoepistemic logic restricted to $B$-formulae shown in Lemma 4.5 of [4]. This implies our claim, as membership in $\mathrm{XP}_{\mathrm{nu}}$ implies a polynomial-time algorithm for any fixed $\kappa$. □

**Theorem 11.** *Let $B$ be a finite set of Boolean functions such that $\oplus, \top \in [B]$ Further let $\kappa$ be a parameterization function for which there exists a $c \in \mathbb{N}$ such that $\kappa(\Sigma) < c$ for all $\Sigma$. If $\mathrm{L} \neq \oplus\mathrm{L}$, then the expansion existence problem for sets of autoepistemic $B$-formulae, parameterized by $\kappa$, is not contained in $\mathrm{XL}_{\mathrm{nu}}$.*

*Proof.* Observe that there exists a reduction $f$ from the implication problem restricted to $B$-formulae shown in Lemma 4.8 of [1]. This implies our claim, as membership in $\mathrm{XL}_{\mathrm{nu}}$ implies a logspace algorithm for any fixed $\kappa$. □

We remark that similar lower bounds as given for default logic in the previous section and for autoepistemic logic here hold for the implication problem as well.

## 6 Pseudo-cliques

Looking at Theorems 6 and 7 one might hope that the syntactic restriction imposed there, namely allowing only defaults that involve literals or propositions, is so severe that it will bound the tree width of every such input structure. Combining this with Theorem 5 would then yield $\mathrm{P} = \mathrm{NP}$ (or $\mathrm{L} = \mathrm{NL}$, resp.). Stated the other way round, if $\mathrm{P} \neq \mathrm{NP}$ then the tree width of $\mathcal{A}_{(W,D)}$ is a non-trivial parameterization, i.e., a parameterization $\kappa$ for which there exists no $c \in \mathbb{N}$ such that $\kappa((\emptyset, D)) < c$ holds for all $D$ consisting of defaults rules involving only literals.

In the following we directly prove the non-triviality of the parameterization by tree width (i.e., without any complexity hypothesizes). As a tool we utilize the subsequent definition of *pseudo-cliques.*

**Definition 12.** *Let $G = (V, E)$ be an undirected graph. A* pseudo-clique *is a set of vertices $V' \subseteq V$ that can be partitioned into the set of* main-nodes *$V_{main}$ and sets of* edge-nodes *$V_{u,v}$ for each $u \neq v \in V_{main}$ such that the following holds: for $v_1, \ldots, v_m \in V_{u,v}$ the nodes in $V_{u,v}$ form a simple path from $u$ to $v$, i.e., it holds that $(u, v_1), (v_1, v_2), \ldots, (v_{m-1}, v_m), (v_m, v) \in E$ and no other edges are present.*

*The* size *of a pseudo-clique is $|V_{main}|$, i.e., the number of main-nodes. The* cardinality *of a pseudo-clique is $\max_{u \neq v \in V_{main}} |V_{u,v}|$, i.e., the length of the longest simple path between edge-nodes. A pseudo-clique is said to have* exact cardinality *$k$ if $\forall u, v \in V_{main}$: $|V_{u,v}| = k$.*

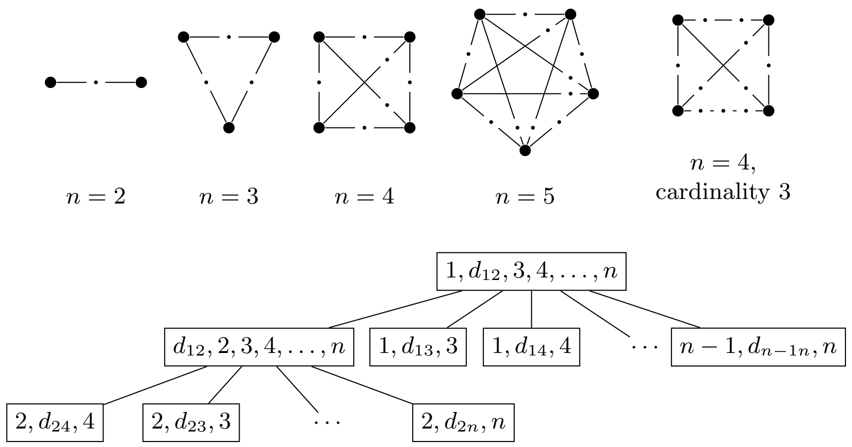


**Fig. 1.** Pseudo-cliques of exact card. 1 and size $\in \{2, \dots, 5\}$, one of card. 3 and size 4, and a tree decomposition of a pseudo-clique of exact card. 1 and size $n$.

The first four pseudo-cliques of exact cardinality 1, and one of cardinality 3 are visualized in Figure 1. The thick vertices correspond to the main-nodes whereas the small dots correspond to the edge-nodes.

The important fact for us is the observation that pseudo-cliques of size $n$ have the same tree width as the clique of size $n$.

**Theorem 13.** *Let $G = (V, E)$ be a pseudo-clique of size $n \geq 3$ and cardinality $k \geq 0$. Then the tree width of $G_n$ is $n - 1$.*

Whenever one wants to show that a parametrization by tree width is non-trivial, the most obvious method is to show that the family of graphs has (sub-) cliques of arbitrary size. Now Theorem 13 provides an alternative when this method is prohibited: it suffices to construct pseudo-cliques. Corollary 14 (1.) shows that, for families used for the lower bounds in the previous sections, it is not possible to use cliques in order to prove unbounded tree width and therefore additionally motivates the definition and purpose of pseudo-cliques.

**Corollary 14.** *Let $B$ be a finite set of Boolean functions such that $\bot \in [B]$ and let $(\emptyset, D)$ be a $B$-default theory in the sense of Theorem 6, i.e., each default in $D$ is composed of literals only. Then there exists an MSO formula $\theta$ fulfilling the property $(W, D) \in \textsc{Ext}(B)$ iff $\mathcal{A}_{(W,D)} \models \theta$, and*

1. *$\mathcal{A}_{(W,D)}$ is neither $\ell$-connected nor contains a clique of size $\ell$ for any $\ell \geq 3$.*
2. *There exists a family of default theories $(\emptyset, D)_k$ such that the tree width of $\mathcal{A}_{(\emptyset,D)_k}$ is not constant.*

*Proof.* For (1.) we construct the MSO formula $\theta$ according to Lemma 4. At first observe that the universe of $\mathcal{A}_{(W,D)}$ comprises only literals and defaults. Further, there are no edges between literals, and no edges between defaults. Every default can be connected to at most three different literals. Obviously the graph does not contain a clique of size $\ell \geq 3$. Furthermore, the graph is not $\ell$-connected for any $\ell \geq 3$ by the following observation. Let $x_d$ be some individual representing the default $d = \frac{\alpha:\beta}{\gamma}$. Then there are individuals $x_\alpha, x_\beta, x_\gamma$ to represent the respective parts of $d$ which are all connected to $x_d$. If now $x_\alpha, x_\beta$ and $x_\gamma$ are removed from the graph, then there is no other individual to which $x_d$ is connected yielding a contradiction to the connectivity.

Turning to (2.) observe that (1.) prohibits using $\ell$-cliques or $\ell$-connectivity for any $\ell \geq 3$ to measure the tree width of $\mathcal{A}_{(W,D)_k}$. Now define a default theory $(\emptyset, D)$ complying with Theorem 10, where $D := \left\{d_{ij} = \frac{x_i:y_j}{\bot} \;\middle|\; 1 \leq i \leq j \leq n\right\}$, and $x_i, y_j$ are variables for $1 \leq i \leq j \leq n$. Consisting only of this kind of default rules implies that the structure forms a pseudo-clique whence the application of Theorem 13 concludes the proof. $\square$

An analogous result holds for autoepistemic logic.

**Corollary 15.** *Let $B$ be a finite set of Boolean functions. There exists a family of autoepistemic $B$-formulae $\Sigma_k$ and all $\varphi \in \Sigma_k$ are disjunctions of propositions or $L$-prefixed propositions such that there exists an MSO formula $\theta$ fulfilling the property $\Sigma_k \in \text{Exp}(B)$ iff $\mathcal{A}_{\Sigma_k} \models \theta$ and the tree width of $\mathcal{A}_{\Sigma_k}$ is not constant.*

*Proof.* Define $\Sigma_k$ as $\Sigma_k := \{x_i \vee x_j \mid 1 \leq i \leq j \leq k\}$. Then the structure $\mathcal{A}_{\Sigma_k}$ consist of cliques of size $k$, in fact. $\square$

**Corollary 16.** *Let $B$ be a finite set of Boolean functions such that $\wedge, \vee \in [B]$. Let $\mathbf{\Gamma}_1$ be the set of sets $\Gamma$ of formulae in monotone 2-CNF and let $\mathbf{\Gamma}_2$ be the set of sets $\Gamma$ of formulae in DNF. There exists a family of sets of $B$-formulae $(F,G)_k$ with $F \in \mathbf{\Gamma}_1, G \in \mathbf{\Gamma}_2$ such that there exists an MSO formula $\theta$ fulfilling the property $(F,G)_k \in \text{Imp}(B)$ iff $\mathcal{A}_{(F,G)_k} \models \theta$ and the tree width of $\mathcal{A}_{(F,G)_k}$ is not constant.*

## 7 Conclusion

In this paper we applied Courcelle's Theorem [3] and the logspace version of Elberfeld et al. [5] to the most prominent decision problems in the nonmontonic default logic and autoepistemic logic. Thereby we showed that the extension existence problem for a given default theory $(W,D)$ is solvable in time $O(f(k) \cdot |(W,D)|)$ and space $O(\log |(W,D)|)$ if the tree width of the corresponding MSO structure is bounded by $k$; similarly for the expansion existence problem for a set of autoepistemic formulae, and as well for the implication problem for sets of formulae $F, G$.

Further we mention that one can achieve similar results for the credulous (resp. brave) and skeptical (resp. cautious) reasoning problems of the nonmontone logics from above by slight extensions of the constructed MSO-formulae.

Furthermore we introduced with *pseudo-cliques* a weaker notion of cliques: basically we have a clique where each edge is divided into two edges by a fresh node (or even a longer path). There we showed that for a graph the size of its largest sub-pseudo-clique gives a lower bound for its tree width. If we investigate default theories $(W, D)$ which contain an empty knowledge base $W$ and only defaults which are composed of propositions or the constant $\bot$ only, then for constant parameterizations we show collapses of P and NP (resp. L and NL) if the corresponding parameterized problem is in $\mathrm{XP}_{\mathrm{nu}}$ (resp. $\mathrm{XL}_{\mathrm{nu}}$). Thus through the concept of pseudo-cliques we construct a family of default theories whose tree width of its MSO-structures is unbounded. Therefore this parameter cannot be used to prove such complexity class collapses. Analogue claims can be made for the expansion existence problem in autoepistemic logic and the implication problem for sets of formulae.

For subsequent research it would be very interesting to find a parameterization that is non-trivial in the sense of Theorem 6 but uses many different values. Also insights on new types of parameterizations, in particular in the context of the new space parameterized complexity classes, would be very engaging.

*Acknowledgement.* For helpful hints and discussions we are grateful to Nadia Creignou (Marseille) and Thomas Schneider (Bremen). We also thank the anonymous referees for helpful comments.

## References

1. Beyersdorff, O., Meier, A., Thomas, M., Vollmer, H.: The Complexity of Propositional Implication. Information Processing Letters 109(18), 1071–1077 (2009)
2. Beyersdorff, O., Meier, A., Thomas, M., Vollmer, H.: The complexity of reasoning for fragments of default logic. J. Log. Comput. (2011), doi:10.1093/logcom/exq061
3. Courcelle, B.: Graph rewriting: An algebraic and logic approach. In: Handb. Theor. Comp. Science. Formal Models and Semantics, pp. 193–242. Elsevier (1990)
4. Creignou, N., Meier, A., Thomas, M., Vollmer, H.: The complexity of reasoning for fragments of autoepistemic logic. ACM Transaction on Computational Logic 13(2) (2010), http://tocl.acm.org/accepted/457meier.pdf
5. Elberfeld, M., Jakoby, A., Tantau, T.: Logspace versions of the theorems of Bodlaender and Courcelle. In: Proc. 51th FOCS. IEEE Computer Society (2010)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
7. Gottlob, G.: Complexity results for nonmonotonic logics. J. Log. Comput. 2(3), 397–425 (1992)
8. Gottlob, G., Pichler, R., Wei, F.: Bounded treewidth as a key to tractability of knowledge representation and reasoning. Artif. Intell. 174(1), 105–132 (2010)
9. Moore, R.C.: Semantical considerations on modal logic. Art. Int. 25, 75–94 (1985)
10. Niemelä, I.: Towards Automatic Autoepistemic Reasoning. In: van Eijck, J. (ed.) JELIA 1990. LNCS, vol. 478, pp. 428–443. Springer, Heidelberg (1991)
11. Reiter, R.: A logic for default reasoning. Artificial Intelligence 13, 81–132 (1980)
12. Stockhusen, C.: Anwendungen monadischer Logik zweiter Stufe auf Probleme beschränkter Baumweite und deren Platzkomplexität. Diplomarbeit (2011)
13. Thomas, M., Vollmer, H.: Complexity of non-monotonic logic. Bulletin of the EATCS 102, 53–82 (2010)

# Cayley Graph Automatic Groups Are Not Necessarily Cayley Graph Biautomatic

Alexei Miasnikov[1] and Zoran Šunić[2]

[1] Department of Mathematical Sciences, Stevens Institute, Castle Point on Hudson
Hoboken, NJ 07030-5991, USA
amiasnikov@gmail.edu

[2] Department of Mathematics, Texas A&M University, MS-3368
College Station, TX 77843-3368, USA
sunic@math.tamu.edu

**Abstract.** We show that there are Cayley graph automatic groups that are not Cayley graph biautomatic. In addition, we show that there are Cayley graph automatic groups with undecidable Conjugacy Problem and that the Isomorphism Problem is undecidable in the class of Cayley graph automatic groups.

**Keywords:** Automatic Structure, Cayley Graph Automatic Group, Cayley Graph Biautomatic Group, Conjugacy Problem, Isomorphism Problem.

## 1 Introduction

The notion of automatic groups, based on ideas of Thurston, Cannon, Gilman, Epstein and Holt, was introduced in [5]. The initial motivation was to understand the fundamental groups of compact 3-manifolds and make them tractable for computing. It was quickly realized that automatic group come short in dealing with manifolds of Nil and Sol types. This immediately triggered a search for suitable generalizations. In [4] Bridson and Gilman came up with a sufficiently powerful notion of automaticity (asynchronously automatic groups where regular languages are replaced with indexed languages) that covers all fundamental groups of compact 3-manifolds, but at the cost of losing the good algorithmic properties.

Since 1990's a lot of groups were proved to be automatic (see the survey in [7]), but a certain frustration still lingers there. It turns out that many basic questions on automatic groups remain wide open despite a considerable effort by the group theoretic community. Three such basic problems ask if automatic groups are biautomatic, if they have a decidable Conjugacy Problem, and if the Isomorphism Problem is decidable within the class. The Cayley graph automatic groups, introduced in [7] (building on the earlier work of Khoussainov and Nerode [8] and the more recent work of Blumensath and Grädel [1]) retain the basic algorithmic properties of the standard automatic groups (decidability of the Word Problem in quadratic time and decidability of the Conjugacy Problem in the biautomatic case) but form a much wider class of groups, which, in particular, contains many nilpotent and solvable groups, which are not automatic under the standard definition. From the algorithmic view-point this indicates that the new class gives a legitimate notion of automaticity. Another confirmation that Cayley graph automatic groups provide a robust generalization of the standard automatic groups is given by the fact that the basic problems mentioned above can be tamed in this case. Namely, we show that all three have a negative solution in the class of Cayley graph automatic groups.

**Theorem 1.** *There are Cayley graph automatic groups that are not Cayley graph biautomatic.*

This answers a question raised implicitly at the end of the introductory section in [7]. The theorem below answers a question raised implicitly at the end of Section 8 in [7].

**Theorem 2.** *There are Cayley graph automatic groups with undecidable Conjugacy Problem.*

Our last result concerns the Isomoprhism Problem.

**Theorem 3.** *The Isomorphism Problem is not decidable in the class of Cayley graph automatic groups.*

Our results follow from several results of Kharlampovich, Khoussainov, and Miasnikov [7], Bogopolski, Martino, and Ventura [2], and Levitt [9].

Bogopolski, Martino, and Ventura proved that certain group extensions have decidable Conjugacy Problem (here and thereafter $F_n$ denotes the free group of rank $n$).

**Theorem 4 (Bogopolski, Martino, Ventura [2] Corollary 7.6).** *There exists a group of the form $\mathbb{Z}^d \rtimes_\tau F_n$ with undecidable Conjugacy Problem.*

The homomorphism $\tau : F_n \to \mathsf{GL}_d(\mathbb{Z})$ constructed in the proof of Theorem 4 in [2] is not injective. In fact, the image $\tau(F_n)$ is not finitely presented and the question of existence of a group of the form $\mathbb{Z}^d \rtimes_\tau F_n$ with undecidable Conjugacy Problem such that $\tau(F_n)$ is finitely presented was left open. A modification of the construction from [2] that was provided in [10] resolved this question.

**Theorem 5 (Šunić, Ventura [10]).** *There exists a group of the form $\mathbb{Z}^d \rtimes_\tau F_n$ with undecidable Conjugacy Problem such that $\tau$ is injective.*

The primary goal of [10] was to prove that the Conjugacy Problem is not decidable in the class of automaton groups (these are self-similar groups of rooted regular tree automorphisms generated by finite, invertible, synchronous transducers; see [6]). The class of automaton groups should not be confused with the class of automatic groups, as defined in [5], nor with its generalization, the Cayley graph automatic groups, as defined in [7]. At present, the relation between the class of automaton groups and the class of Cayley graph automatic groups is not clear and only the latter is the subject of consideration in this work.

The final ingredient in the proof of Theorem 2 is Theorem 6, which is not stated in [7] in the form in which we quote it here, but it is a corollary of the other results presented there. Theorem 2 directly follows from Theorem 4 and Theorem 6.

**Theorem 6 (Kharlampovich, Khoussainov, Miasnikov [7]).** *All groups of the form $\mathbb{Z}^d \rtimes F_n$ are Cayley graph automatic.*

As a direct corollary of Theorem 2 and the following result, we obtain Theorem 1.

**Theorem 7 (Kharlampovich, Khoussainov, Miasnikov [7] Thm. 8.5).** *Cayley graph biautomatic groups have decidable Conjugacy Problem.*

As a direct corollary of Theorem 6 and the following result of Levitt, we obtain Theorem 3.

**Theorem 8 (Levitt [9]).** *The Isomorphism Problem is not decidable in the class of groups of the form $\mathbb{Z}^d \rtimes F_n$.*

Note that there is an earlier work of Zimmermann [11] showing that the isomorphism problem is not decidable in (free-abelian)-by-surface groups, and that his argument could be applied for (free-abelian)-by-free groups.

It is important to observe that our examples of Cayley graph automatic groups that are not biautomatic and have undecidable Conjugacy Problem are not automatic in the standard sense. Indeed, one can prove the following.

**Theorem 9.** *If a group of the form $\mathbb{Z}^d \rtimes F_n$ has subexponential Dehn function, then it has decidable Conjugacy Problem.*

In the remaining sections we provide the necessary definitions and other details.

## 2 Cayley Graph Automatic and Cayley Graph Biautomatic Groups

Let $\Sigma$ be a finite alphabet. We will sometimes extend this alphabet by a special symbol $\diamond$ that is not in $\Sigma$, and we denote $\Sigma_\diamond = \Sigma \cup \{\diamond\}$.

The *convolution* $\otimes(w_1, \ldots, w_n)$ of an $n$-tuple of words $(w_1, \ldots, w_n)$ over $\Sigma$ is the word of length $\max\{|w_1|, \ldots, |w_n|\}$ over $(\Sigma_\diamond)^n$ in which the $j$-th symbol is $(\sigma_1, \ldots, \sigma_n)$, where

$$\sigma_i = \begin{cases} \text{the } j\text{-th symbol of } w_i, & \text{if } j \leq |w_i| \\ \diamond, & \text{otherwise} \end{cases}.$$

For instance,

$$\otimes(aaa, babaa, \emptyset) = \begin{pmatrix} a \\ b \\ \diamond \end{pmatrix} \begin{pmatrix} a \\ a \\ \diamond \end{pmatrix} \begin{pmatrix} a \\ b \\ \diamond \end{pmatrix} \begin{pmatrix} \diamond \\ a \\ \diamond \end{pmatrix} \begin{pmatrix} \diamond \\ a \\ \diamond \end{pmatrix},$$

where $\emptyset$ denotes the empty word and the symbols in $(\Sigma_\diamond)^n$ are written, for convenience, as columns.

Let $R$ be an $n$-ary relation on $\Sigma^*$. The *convolution* $\otimes R$ of $R$ is the language over $(\Sigma_\diamond)^n$ defined by

$$\otimes R = \{ \otimes(w_1, \ldots, w_n) \mid (w_1, \ldots, w_n) \in R \}.$$

A relation $R$ is *regular* over $\Sigma$ if its convolution $\otimes R$ is a regular language over $(\Sigma_\diamond)^n$, i.e., $\otimes R$ is recognizable by a finite automaton over the alphabet $(\Sigma_\diamond)^n$ (let us note that, in this work, the automata always read words from left to right). A subset $R$ of $\Sigma^*$ may be considered as a unary relation and, since in this case, $\otimes R = R$, there is no difference (or confusion) between the regularity of $R$ over $\Sigma$ as a subset of $\Sigma^*$ (as is usually defined) or as a relation on $\Sigma^*$ (in the sense defined above, using convolutions).

Let $G$ be a finitely generated group with finite generating set $S$. The right Cayley graph of $G$ with respect to $S$ is the graph $\Gamma(G, S)$ with $G$ as the set of vertices and, for each $g$ in $G$ and $s$ in $S$, an edge from $g$ to $gs$. The Cayley graph can be interpreted as a system of $|S|$ binary relations $E_s$ on $G$, for $s$ in $S$, where

$$E_s = \{ (g, gs) \mid g \in G \}.$$

A map $^- : G \to \Sigma^*$ induces $|S|$ binary relations on $\Sigma^*$ given by

$$\overline{E}_s = \{ (\overline{g}, \overline{gs}) \mid g \in G \}.$$

**Definition 1.** *A finitely generated group $G$ with finite generating set $S$ is* Cayley graph automatic *if there exists a finite alphabet $\Sigma$ and an injective map $^- : G \to \Sigma^*$ such that*

*$\overline{G}$ is regular (over $\Sigma$) and*

*$\overline{E}_s$ is regular (over $\Sigma$), for every $s$ in $S$.*

*In such a case the tuple $(\overline{G}, \overline{E}_{s_1}, \ldots, \overline{E}_{s_k})$ is called an* automatic structure *of the Cayley graph $\Gamma(G, S)$ or Cayley graph automatic structure of $G$ (with respect to $S = \{s_1, \ldots, s_k\}$).*

In addition to the right Cayley graph one may consider the left Cayley graph $\Gamma^\ell(G, S)$ as well. The vertex set is $G$ and, for each $g$ in $G$ and $s$ in $S$, an edge from $g$ to $sg$. The left Cayley graph can be interpreted as a system of $|S|$ binary relations $E^\ell_s$ on $G$, for $s$ in $S$, where

$$E^\ell_s = \{ (g, sg) \mid g \in G \}.$$

**Definition 2.** *A finitely generated group $G$ with finite generating set $S$ is* Cayley graph biautomatic *if there exists a finite alphabet $\Sigma$ and an injective map $^- : G \to \Sigma^*$ such that*

*$\overline{G}$ is regular (over $\Sigma$),*

*$\overline{E}_s$ is regular (over $\Sigma$), for every $s$ in $S$, and*

*$\overline{E}^\ell_s$ is regular (over $\Sigma$), for every $s$ in $S$.*

*In such a case the tuple $(\overline{G}, \overline{E}_{s_1}, \ldots, \overline{E}_{s_k}, \overline{E}^\ell_{s_1}, \ldots, \overline{E}^\ell_{s_k})$ is called a* biautomatic structure *of the pair of Cayley graphs $\Gamma(G, S)$ and $\Gamma^\ell(G, S)$ or Cayley graph biautomatic structure of $G$ (with respect to $S = \{s_1, \ldots, s_k\}$).*

It is important to observe that being Cayley graph automatic is a property of the group and does not depend on the chosen finite generating set $S$, i.e., $G$ is Cayley graph automatic with respect to a finite generating $S$ if and only if it is Cayley graph automatic with respect to any of its other finite generating sets (Theorem 6.9. [7]).

All (bi)automatic groups, as defined in [5] are Cayley (bi)automatic (Proposition 7.3. and Proposition 8.4. [7]). The class of Cayley graph automatic groups is much wider than the class of automatic groups. For instance, it includes the Heisenberg group $H = \langle a, b \mid [a, [a, b]] = [b, [a, b]] = 1 \rangle$ and many other nilpotent groups that are not automatic (Example 6.6 [7]). Nevertheless, the class of Cayley (bi)automatic groups retains many algorithmic properties of (bi)automatic groups. For instance, every Cayley graph automatic group has Word Problem decidable in quadratic time and every Cayley graph biautomatic group has decidable Conjugacy Problem (Theorem 8.2. and Theorem 8.5. [7]).

The class of Cayley graph automatic groups has good closure properties. The following is, in particular, relevant for our purposes.

**Theorem 10 (Kharlampovich, Khoussainov, Miasnikov [7] Thm. 10.3).** *Let $A$ and $B$ be Cayley graph automatic groups with finite generating sets $X$ and $Y$, respectively. Let $\tau : B \to \mathsf{Aut}(A)$ be a homomorphism such that the automorphism $\tau(y)$ is automatic, for every $y$ in $Y$. Then the semidirect product $G = A \rtimes_\tau B$ is Cayley graph automatic.*

Let us briefly explain what is meant by an automatic automorphism $\alpha$ of a Cayley graph automatic group $A$. Let $A$ be automatic over $\Sigma$ and $^- : A \to \Sigma^*$ be the injective map used in the automatic structure of $A$. The automorphism $\alpha$ induces a binary relation $\{(a, a^\alpha) \mid a \in A\}$ on $A$, which, in turn, induces a binary relation $\overline{\alpha} = \{(\overline{a}, \overline{a^\alpha}) \mid a \in A\}$ on $\Sigma^*$. The automorphism $\alpha$ is automatic if the relation $\overline{\alpha}$ is a regular relation over $\Sigma$.

The semidirect product $A \rtimes_\tau B$ is the set of all pairs $(b, a)$, for $b \in B$, $a \in A$, with product defined by $(b_1, a_1)(b_2, a_2) = (b_1 b_2, a_1^{\tau(b_2)} a_2)$.

It is known that the free abelian group $A = \mathbb{Z}^d$ and the free group $B = F_n$ of finite ranks are automatic, and hence they are Cayley graph automatic. The argument in the proof of Proposition 10.5 in [7], showing that every automorphism of $\mathbb{Z}^2$ is automatic, can be extended to show that every automorphism of $\mathbb{Z}^d$ is automatic. In other words, multiplication of $d$-tuples of integers by any fixed $d \times d$ matrix in $\mathsf{GL}_d(\mathbb{Z})$ is automatic. These observations, together with Theorem 10 immediately imply Theorem 6.

## 3 (Free-abelian)-by-free Groups with Undecidable Conjugacy Problem

For the duration of this section, let $A = \mathbb{Z}^d$ and $B = F_n$ (this agreement is not crucial for all statements, but this is the setting we are aiming for and there is no need to go into more general considerations).

Let $C$ be a subgroup of $\mathsf{Aut}(A) = \mathsf{GL}_d(\mathbb{Z})$. We say that $C$ has undecidable Orbit Problem if there is no algorithm that decides, on input consisting of arbitrary pair of vectors $u$ and $v$ in $A$, if there exists a matrix $c$ in $C$ such that $u^c = v$ (we use the right action of matrices on vectors; this is just the multiplication of vectors by the matrix $c$ on the right). Let $\tau : B \to \mathsf{GL}_d(\mathbb{Z})$ be a homomorphism such that $\tau(B) = C$. If $C$ has undecidable Orbit Problem then the semidirect product $G = A \rtimes_\tau B$ has undecidable Conjugacy Problem. Indeed, as observed in [2], two vectors $u$ and $v$ in $A$ are conjugate in $G$ if and only if they are in the same orbit under the action of $C = \tau(B)$, and since the latter problem is undecidable, so is the Conjugacy Problem in $G$.

A good way to construct orbit undecidable subgroups of $\mathsf{GL}_d(\mathbb{Z})$ is provided in [2] (Section 7; in particular, Proposition 7.5. and Corollary 7.6., the latter of which is listed in our introduction as Theorem 4). Let $d \geq 4$ and let $H$ be a finitely presented group with undecidable Word Problem. Use the Mikhailova construction to obtain the corresponding finitely generated subgroup $H'$ of $F_2 \times F_2$ with undecidable Membership Problem and then consider $F_2 \times F_2$ as a subgroup of $\mathsf{GL}_d(\mathbb{Z})$ through a specific embedding ($F_2 \times F_2$ embeds in $\mathsf{GL}_d(\mathbb{Z})$, for $d \geq 4$) that turns the undecidability of the Word Problem in $H$ into undecidability of the Orbit Problem for $H' = C \leq \mathsf{GL}_d(\mathbb{Z})$ (a specific embedding of $F_2 \times F_2$ with this property is spelled out precisely in [2]). The group $G = A \rtimes_\tau B$, where $\tau : B \to \mathsf{GL}_d(\mathbb{Z})$ is any homomorphism with $\tau(B) = C$, has undecidable Conjugacy Problem.

The group $C$, as defined above, is finitely generated and not finitely presented. Thus, $\tau$ is not injective for any group of the form $G = A \rtimes_\tau B$ with $C = \tau(B)$ as in the above construction.

The following modification, introduced in [10], provides groups of the form $G = A \rtimes_\tau B$ with undecidable Conjugacy Problem and injective $\tau$. Let $C = \langle g_1, \ldots, g_n \rangle$ be an orbit undecidable subgroup of $\mathsf{GL}_d(\mathbb{Z})$, let $B = F(f_1, \ldots, f_n)$ be free of rank $n$, and let $C' = \langle g_1', \ldots, g_n' \rangle$ be any free subgroup of rank $n$ of $\mathsf{GL}_2(\mathbb{Z})$ (for every $n$, such subgroups exist). Define $\tau : B \to \mathsf{GL}_{d+2}(\mathbb{Z})$ by

$$\tau(f_i) = \begin{bmatrix} g_i & 0_{d\times 2} \\ 0_{2\times d} & g_i' \end{bmatrix},$$

for $i = 1, \ldots, n$, where $0_{d\times 2}$ and $0_{2\times d}$ are the zero matrices of appropriate sizes. In other words, the action of $\tau(f_i)$ on the first $d$ coordinates of $\mathbb{Z}^{d+2}$ is the same as the action of the matrix $g_i$, and on the last two coordinates as the action of the matrix $g_i'$. It is clear that $\tau$ is injective (since it is injective "on the last two coordinates"). Moreover, the undecidability of the Orbit Problem for $C$ in $\mathsf{GL}_d(\mathbb{Z})$ induces the undecidability of the Orbit Problem for the free subgroup $C' = \tau(B)$ in $\mathsf{GL}_{d+2}(\mathbb{Z})$ (see Proposition 1 in [10], which is listed as Theorem 5 in our introduction).

At the end, we show that our examples of Cayley graph automatic groups that are not Cayley graph biautomatic and have undecidable Conjugacy Problem are not automatic under the standard definition. In fact, Theorem 9 implies that our examples cannot even have subexponential Dehn functions (recall that the groups that are automatic in the standard sense have quadratic Dehn functions).

*Proof (Proof of Theorem 9).* Let $G = \mathbb{Z}^d \rtimes_\tau F_n$ be a group with subexponential Dehn function. Bridson showed that the Dehn function of $G$ can be either polynomial or exponential and the former is possible only when $F_n$ has a subgroup $F'$ of finite index such that $\tau(F')$ is unipotent [3]. This implies that $\tau(F_n)$ is virtually solvable. Since virtually solvable subgroups of $\mathsf{GL}_d(\mathbb{Z})$ have decidable Orbit Problem, it follows that $G$ has decidable Conjugacy Problem (see Proposition 6.9 and Corollary 6.10 in [2]).

## References

1. Blumensath, A., Grädel, E.: Automatic structures. In: 15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, CA, pp. 51–62. IEEE Comput. Soc. Press, Los Alamitos (2000)
2. Bogopolski, O., Martino, A., Ventura, E.: Orbit decidability and the conjugacy problem for some extensions of groups. Trans. Amer. Math. Soc. 362(4), 2003–2036 (2010)
3. Bridson, M.R.: Optimal isoperimetric inequalities for abelian-by-free groups. Topology 34(3), 547–564 (1995)
4. Bridson, M.R., Gilman, R.H.: Formal language theory and the geometry of 3-manifolds. Comment. Math. Helv. 71(4), 525–555 (1996)
5. Epstein, D.B.A., Cannon, J.W., Holt, D.F., Levy, S.V.F., Paterson, M.S., Thurston, W.P.: Word processing in groups. Jones and Bartlett Publishers, Boston (1992)
6. Grigorchuk, R.I., Nekrashevich, V.V., Sushchanskiĭ, V.I.: Automata, dynamical systems, and groups. Tr. Mat. Inst. Steklova 231(Din. Sist., Avtom. i Beskon. Gruppy), 134–214 (2000)
7. Kharlampovich, O., Khoussainov, B., Miasnikov, A.: From automatic structures to automatic groups (2011), arXiv:1107.3645v2
8. Khoussainov, B., Nerode, A.: Automatic Presentations of Structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995)
9. Levitt, G.: Unsolvability of the isomorphism problem for [free abelian]-by-free groups (2008), arXiv:0810.0935v2
10. Šunić, Z., Ventura, E.: The conjugacy problem in automaton groups is not solvable (2010), arXiv:1010.1993v2
11. Zimmermann, B.: Zur Klassifikation höherdimensionaler Seifertscher Faserräume. In: Low-Dimensional Topology (Chelwood Gate, 1982). London Math. Soc. Lecture Note Ser., vol. 95, pp. 214–255. Cambridge Univ. Press, Cambridge (1985)

# On Model Checking for Visibly Pushdown Automata

Tang Van Nguyen and Hitoshi Ohsaki

Research Team for Verification and Specification
National Institute of Advanced Industrial Science and Technology, Japan
{t.nguyen,ohsaki}@ni.aist.go.jp

**Abstract.** In this paper we improve our previous work by introducing *optimized on-the-fly* algorithms to test universality and inclusion problems of visibly pushdown automata. We implement the proposed algorithms in a prototype tool. We conduct experiments on randomly generated VPA. The experimental results show that the proposed method outperforms the standard one by several orders of magnitude.

## 1 Introduction

Visibly pushdown automata [1] are pushdown automata whose stack behavior (*i.e.,* whether to execute push, pop, or no stack operation) is completely determined by the input symbol according to a fixed partition of the input alphabet. As shown in [1], this class of visibly pushdown automata enjoys many good properties similar to those of the class of finite automata. The main reason for this being is that, each nondeterministic VPA can be transformed into an equivalent deterministic one. Therefore, checking context-free properties of pushdown models is decidable as long as the calls and returns are made visible. As a result, visibly pushdown automata have turned out to be useful in some context, *e.g.,* as automaton model for processing XML streams [6,3], and as AOP protocols for component-based systems [4]. To check universality for a nondeterministic VPA $M$ over its alphabet $\Sigma$ (that is, to check if $L(M) = \Sigma^*$), the standard method is first to make it complete, determinize it, complement it, and then check it for emptiness. To check the inclusion problem $L(M) \subseteq L(N)$, the standard method computes the complement of $N$, takes its intersection with $M$ and then, checks for emptiness. This is costly as computing the complement necessitates a full determinization. This explosion is in some senses inevitable, because determinization for VPA requires exponential time blowup [1]. This raises a natural question: Are there methods to efficiently implement decision procedures like universality (or inclusion) checking for VPA.

A *pushdown system* (*e.g.,* see [2]) is a pushdown automaton that is regardless of input symbols. Bouajjani *et al.* [2] have introduced a method to compute reachable configurations of a pushdown system. The key of their technique is to use a finite automaton so-called $\mathcal{P}$*-automaton* to encode a set of infinite configurations of a pushdown system. In our previous paper [5], we proposed on-the-fly algorithms to check universality and inclusion of VPA. The key idea is based on doing determinization and generating $\mathcal{P}$-automata simultaneously.

In this paper we improve the algorithms presented in [5] by optimizing the determinized VPA and generated $\mathcal{P}$-automaton as follows. First, we propose an on-the-fly method to test universality of VPA $M$. In particular, in order to check universality of a nondeterministic VPA, we simultaneously determinize this VPA and apply the $\mathcal{P}$-automata technique to compute a set of reachable configurations of the target determinized VPA. When a rejecting configuration is found, the checking process stops and reports that the original VPA is not universal. Otherwise, if all configurations are accepting, the original VPA is universal. Furthermore, to strengthen the algorithm, we define a partial ordering over transitions of $\mathcal{P}$-automaton, and only *minimal* transitions are used to incrementally generate the $\mathcal{P}$-automaton. The purpose of this process is to keep the determinization step implicitly for generating reachable configurations as small as possible. This improvement helps to reduce not only the size of the $\mathcal{P}$-automaton but also the complexity of the determinization phase. The intuitive idea behind this process is to find whether there exists a word $w$ such that $w \notin L(M)$ as early as possible. Second, an algorithmic solution to inclusion checking for VPA using on-the-fly manner will be presented. Again, no explicit determinization is performed. To solve the language-inclusion problem for nondeterministic VPA, $L(M) \subseteq L(N)$, the main idea is to find at least one word $w$ accepted by $M$ but not accepted by $N$, *i.e.*, $w \in L(M) \setminus L(N)$. Finally, we implement all algorithms in a prototype tool, named VPAChecker, and tested them in a series of experiments. Our preliminary experiments on randomly generated VPA show a significant improvement of on-the-fly methods compared to the standard ones.

The remainder of this paper is organized as follows. In Section 2 we recall basic notions and properties of VPA, and then we give an improvement on determinization of VPA. Section 3 propose optimized on-the-fly algorithms for checking universality and inclusion of VPA. Also, the correctness proof of the proposed algorithm is presented in this section. Implementation as well as experimental results are presented and analyzed in Section 4. Finally, we conclude the paper in Section 5.

## 2 Visibly Pushdown Automata

### 2.1 Definitions

Let $\Sigma$ be the finite input alphabet, and let $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$ be a partition of $\Sigma$. The intuition behind the partition is: $\Sigma_c$ is the finite set of *call* (push) symbols, $\Sigma_r$ is the set of *return* (pop) symbols, and $\Sigma_i$ is the finite set of *internal* symbols. Visibly pushdown automata are formally defined as follows:

**Definition 1 (Visibly Pushdown Automata [1]).** *A* visibly pushdown automaton *(VPA) $M$ over $\Sigma$ is a tuple $M = (Q, \Gamma, Q_0, \Delta, F)$ where $Q$ is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states, $\Gamma$ is a* *finite stack alphabet with a special symbol* $\perp$ *(representing the* bottom-of-stack*), and* $\Delta = \Delta_c \cup \Delta_r \cup \Delta_i$ *is the transition relation, where* $\Delta_c \subseteq Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\perp\})$*,* $\Delta_r \subseteq Q \times \Sigma_r \times \Gamma \times Q$*, and* $\Delta_i \subseteq Q \times \Sigma_i \times Q$*.*

- If $(q, c, q', \gamma) \in \Delta_c$, where $c \in \Sigma_c$ and $\gamma \neq \perp$, there is a *push-transition* from $q$ on input $c$ where when reading $c$, $\gamma$ is pushed onto the stack and the control changes from state $q$ to $q'$; we denote such a transition by $q \xrightarrow{c/+\gamma} q'$.
- Similarly, if $(q, r, \gamma, q') \in \Delta_r$, there is a *pop-transition* from $q$ on input $r$ where $\gamma$ is read from the top of the stack and popped (if the top of the stack is $\perp$, then it is read but not popped), and the control state changes from $q$ to $q'$; we denote such a transition $q \xrightarrow{r/-\gamma} q'$.
- If $(q, i, q') \in \Delta_i$, there is an *internal-transition* from $q$ on input $i$ where when reading $i$, the state changes from $q$ to $q'$; we denote such a transition by $q \xrightarrow{i} q'$. Note that there are no stack operations on internal transitions.

Let $St = \{w\perp \mid w \in (\Gamma \setminus \{\perp\})^*\}$ be the set of *stack contents*. A *configuration* is a pair $(q, \sigma)$ of $q \in Q$ and $\sigma \in St$. The transition relation of a VPA can be used to define how the configuration of the machine changes in a single step: we say $(q, \sigma) \xrightarrow{a} (q', \sigma')$ if one of the following conditions holds:

- If $a \in \Sigma_c$ then there exists $\gamma \in \Gamma$ such that $q \xrightarrow{a/+\gamma} q'$ and $\sigma' = \gamma \cdot \sigma$
- If $a \in \Sigma_r$, then there exists $\gamma \in \Gamma$ such that $q \xrightarrow{a/-\gamma} q'$ and either $\sigma = \gamma \cdot \sigma'$, or $\gamma = \perp$ and $\sigma = \sigma' = \perp$
- If $a \in \Sigma_i$, then $q \xrightarrow{a} q'$ and $\sigma = \sigma'$.

A $(q_0, w_0)$-*run* on a word $u = a_1 \cdots a_n$ is a sequence of configurations $(q_0, w_0) \xrightarrow{a_1} (q_1, w_1) \cdots \xrightarrow{a_n} (q_n, w_n)$, and is denoted by $(q_0, w_0) \xrightarrow{u} (q_n, w_n)$. A word $u$ is accepted by $M$ if there is a run $(q_0, w_0) \xrightarrow{u} (q_n, w_n)$ with $q_0 \in Q_0$, $w_0 = \perp$, and $q_n \in F$. The language $L(M)$ is the set of words accepted by $M$. The language $L \subseteq \Sigma^*$ is a *visibly pushdown language* (VPL) if there exists a VPA $M$ with $L = L(M)$.

**Definition 2 (Deterministic VPA [1]).** *A VPA M is* deterministic *if* $|Q_0| = 1$ *and for every configuration* $(q, \sigma)$ *and* $a \in \Sigma$*, there are at most one transition from* $(q, \sigma)$ *by* $a$*. For deterministic VPA (DVPAs) we denote the transition relation by* $\delta$ *instead of* $\Delta$*, and write:* $\delta(q, a) = (q', \gamma)$ *instead of* $(q, a, q', \gamma) \in \Delta_c$ *if* $a \in \Sigma_c$*,* $\delta(q, a, \gamma) = q'$ *instead of* $(q, a, \gamma, q') \in \Delta_r$ *if* $a \in \Sigma_r$*, and* $\delta(q, a) = q'$ *instead of* $(q, a, q') \in \Delta_i$ *if* $a \in \Sigma_i$*.*

As shown in [1], any nondeterministic VPA can be transformed into an equivalent deterministic one. The construction has two components: a set of *summary edges* $S$, that keeps track of what state transitions are possible from a push transition to the corresponding pop transition, and a set of *path edges* $R$, that keeps track of all possible states reached from initial states. Reader(s) are referred to [1] for more details. During implementation of VPA's operations, we found that the set of summaries $S$ may contain redundant pairs in the sense that these pairs

```
Algorithm 1. Optimized determinization for VPA
  Data: A nondeterministic VPA $M = (Q, \Gamma, Q_0, \Delta, F)$
  Result: A determinized VPA $M^{od} = (Q', \Gamma', Q'_0, \Delta', F')$
1 begin
2     $Q' = 2^{Q \times Q}$, $\Gamma' = Q' \times \Sigma_c$,
3     $Q'_0 = \{Id_{Q_0}\}$, $F' = \{S \in Q' \mid \Pi_2(S) \cap F \neq \varnothing\}$,
4     and the transition relation $\Delta' = \Delta'_i \cup \Delta'_c \cup \Delta'_r$ is given by:
        – Internal: For every $a \in \Sigma_i$, $S \xrightarrow{a} S' \in \Delta'_i$ where
          $S' = \{(q, q') \mid \exists q'' \in Q : (q, q'') \in S, q'' \xrightarrow{a} q' \in \Delta_i\}$.
        – Push: For every $a \in \Sigma_c$, $S \xrightarrow{a/+(S,a)} Id_{R'} \in \Delta'_c$ where
          $R' = \{q' \mid \exists q \in \Pi_2(S) : q \xrightarrow{a/+\gamma} q' \in \Delta_c\}$.
        – Pop: For every $a \in \Sigma_r$,
          - if the stack is empty : $S \xrightarrow{a/-\bot} S' \in \Delta'_r$ where
          $S' = \{(q, q') \mid \exists q'' \in Q : (q, q'') \in S, q'' \xrightarrow{a/-\bot} q' \in \Delta_r\}$.
          - otherwise:
          $S \xrightarrow{a/-(S',a')} S'' \in \Delta'_r$, where
          $\begin{cases} S'' = \{(q,q') \mid \exists q_3 \in Q : (q,q_3) \in S', (q_3,q') \in Update\} \\ Update = \left\{ (q,q') \,\middle|\, \begin{array}{l} \exists q_1, q_2 \in Q : (q_1,q_2) \in S, \\ q \xrightarrow{a'/+\gamma} q_1 \in \Delta_c, q_2 \xrightarrow{a/-\gamma} q' \in \Delta_r \end{array} \right\} \end{cases}$
```

do not keep information of reachable states. In other words, Alur-Madhusudan's algorithm defines each state of the output deterministic VPA as a pair $(S, R)$. However, by a little modification of the algorithm, we can make every pair $(S, R)$ satisfy $\Pi_2(S) = R$ without disturbing the correctness of the algorithm (where, $\Pi_2(S) = \{s \mid (s, s') \in S\}$ is the projection on the second component.) After this modification, the component $R$ is no longer needed. In the following, we formally present an optimization for determinization by keeping the set of summaries as small as possible. For a finite set $X$, let denote $Id_X = \{\ (q, q) \mid q \in X\}$. Let $M = (Q, \Gamma, Q_0, \Delta, F)$ be a nondeterministic VPA. We construct an equivalent deterministic VPA $M^{od} = (Q', \Gamma', Q'_0, \Delta', F')$ ( *od* stands for *optimized determinization*) as presented in Algorithm 1.

**Theorem 3 (Optimized Determinization [5]).** *For a given nondeterministic VPA $M$ of $n$ states, one can construct a deterministic VPA $M^{od}$ such that $L(M^{od}) = L(M)$. Moreover, the number of states and stack symbols of $M^{od}$ in the worst case are $2^{n^2}$ and $|\Sigma_c| \cdot 2^{n^2}$, respectively.*

## 3 Universality and Inclusion Checking

### 3.1 Emptiness Checking

A *pushdown system* (*e.g.,* see [2]) is a pushdown automaton that is regardless of input symbols. Bouajjani *et al.* [2] have introduced a method to compute reachable configurations of a pushdown system. The key of their technique is to use a finite automaton "so-called $\mathcal{P}$-*automaton*" to encode a set of infinite configurations of a pushdown system. In the following, we apply $\mathcal{P}$-automata technique to check emptiness of visibly pushdown automata. Our definition, though in essence does not differ from the one in [2], has been tailored so that concepts discussed in this paper are easily related to the definition. Given a VPA $\mathcal{P} = (Q, \Gamma, Q_0, \Delta, F)$, a $\mathcal{P}$-automaton is used in order to represent sets of configurations $C$ of $\mathcal{P}$. A $\mathcal{P}$-automaton uses $\Gamma$ as the input alphabet, and $Q$ as set of initial states. Formally,

**Definition 4 ($\mathcal{P}$-automata [2]).**

1. *A $\mathcal{P}$-automaton of a VPA $\mathcal{P}$ is a finite automaton $A = (P, \Gamma, \delta, Q, F_A)$ where $P$ is the finite set of states, $\delta \subseteq P \times \Gamma \times P$ is the set of transitions, $Q \subseteq P$ is the set of* initial states *and $F_A \subseteq P$ is the set of* final states*.*
2. *A $\mathcal{P}$-automaton* accepts *or* recognizes *a configuration $(p, w)$ if $p \xrightarrow{w} q$, for some $p \in Q$, $q \in F_A$. The set of configurations recognized by $\mathcal{P}$-automaton $A$ is denoted by $Conf(\mathcal{P})$.*

For a VPA $\mathcal{P} = (Q, \Gamma, Q_0, \Delta, F)$ and a set of configurations $C$, let $A$ be a $\mathcal{P}$-automaton representing $C$. The $\mathcal{P}$-automaton $A_{Post^*(C)}$ representing the set of configurations reachable from $C$ (denoted by $Post^*(C)$) is constructed as follows: We compute $Post^*(C)$ as a language accepted by a $\mathcal{P}$-automaton $A_{post^*(C)}$ with $\epsilon$-moves. We denote the relation $q(\xrightarrow{\epsilon})^* \cdot \xrightarrow{\gamma} \cdot (\xrightarrow{\epsilon})^* p$ by $q \Longrightarrow^{\gamma} p$. Formally, $A_{post^*(C)}$ is obtained from $A$ by the procedure given in Algorithm 2:

### 3.2 Universality Checking

**Standard Method.** The standard algorithm for universality of VPA is to first determinize the automaton, and then check for the reachability of a non-accepting state. Reachable configurations of a determinized VPA can be computed by using $\mathcal{P}$-automata technique. A configuration $c = (q, w)$ is *rejecting* if $q$ is not a final state. When a rejecting configuration is found, we stop and report that the original VPA is not universal. Otherwise, if all reachable configurations of determinized VPA are accepting, the original VPA is universal. Let $\mathsf{ReachableConf}(M^{od})$ and $\mathsf{RejectingConf}(M^{od})$ denote the sets of reachable and rejecting configurations of $M^{od}$, respectively. With the above observation, we obtain the following lemma:

**Lemma 5.** *Let $M$ be a nondeterministic VPA according to Algorithm 1. The automaton $M$ is not universal iff there exists a rejecting reachable configuration of $M^{od}$,* i.e., $\mathsf{ReachableConf}(M^{od}) \cap \mathsf{RejectingConf}(M^{od}) \neq \varnothing$.

**Algorithm 2.** The algorithm to construct $\mathcal{P}$-Automaton

```
Data: A VPA $\mathcal{P} = (Q, \Gamma, Q_0, \Delta, F)$ and a finite automaton $A$ representing set
      of configurations $C$.
Result: A finite automaton $A_{post^*(C)}$ representing $post^*(C)$
1 begin
2     for (each pair $(q', \gamma')$ such that $\mathcal{P}$ contains at least one rule of the form
      $q \xrightarrow{a/+\gamma'} q' \in \Delta_c$) do
3         Add a new state $p_{(q',\gamma')}$ to $A$. Here $(q', \gamma')$ is used as an index to
          distinguish $p_{(q',\gamma')}$ with others newly added states, and thus, we can
          minimize the number of necessary added states.
4     Add new transitions to $A$ according to the following saturation rules:
        1. Internal: if ($q \xrightarrow{a} q' \in \Delta_i$ and $q \Longrightarrow^{\gamma} p$ in the current automaton) then
            Add a transition $(q', \gamma, p)$.
        2. Push: if ($q \xrightarrow{a/+\gamma'} q' \in \Delta_c$ and $q \Longrightarrow^{\gamma} p$ in the current automaton) then
            first add $(q', \gamma', p_{(q',\gamma')})$, and then add $(p_{(q',\gamma')}, \gamma, p)$.
        3. Pop: if ($q \xrightarrow{a/-\gamma} q' \in \Delta_r$ and $q \Longrightarrow^{\gamma} p$ in the current automaton) then
            Add a transition $(q', \epsilon, p)$.
```

**Optimized On-the-fly Method.** To improve efficiency of the checking process, we perform simultaneously on-the-fly determinization and $\mathcal{P}$-automata construction. There are two interleaving phases in this approach. First, we determinize VPA $M$ step by step (iterations). After each step of determinization, we update the $\mathcal{P}$-automaton. Second, using the $\mathcal{P}$-automaton, we perform determinization again, and so on. It is crucial to note that this procedure terminates. This is because the size of the $M^{od}$ is finite, and the $\mathcal{P}$-automaton construction terminates. Lemma 5 means that checking universality of $M$ amounts to finding a rejecting configuration of $M^{od}$. In the following we present an on-the-fly way to explore such rejecting configurations (if there are any) efficiently. We begin with the following observations that play an important role in establishing theoretical background for correctness of our algorithms. For a given nondeterministic VPA $M$, let $M^{od}$ be the determinized VPA. Recall that a state $S$ of $M^{od}$ belongs to $2^{Q \times Q}$. We now define an ordering over states and stack symbols of $M^{od}$ as follows:

**Definition 6 (Partial ordering over states and stack symbols).**

- *Let $S_1$ and $S_2$ be states of $M^{od}$. We say $S_1 \leq S_2$ if, $S_1 \subseteq S_2$.*
- *Let $\gamma'_1 = (S_1, a)$ and $\gamma'_2 = (S_2, a)$ be stack symbols of $M^{od}$. We say $\gamma'_1 \leq \gamma'_2$ if $S_1 \subseteq S_2$.*

**Lemma 7.** *Let $S_1 \xrightarrow{a} S'_1$ and $S_2 \xrightarrow{a} S'_2$ be internal transitions of the determinized VPA $M^{od}$. We have $S'_1 \leq S'_2$ if $S_1 \leq S_2$.*

*Proof.* By the determinization procedure, we have:

– $S_1 \xrightarrow{a} S'_1 \in \Delta'_i$ where $S'_1 = \{(q,q') \mid \exists q'' \in Q : (q,q'') \in S_1, q'' \xrightarrow{a} q' \in \Delta_i\}$.
– $S_2 \xrightarrow{a} S'_2 \in \Delta'_i$ where $S'_2 = \{(q,q') \mid \exists q'' \in Q : (q,q'') \in S_2, q'' \xrightarrow{a} q' \in \Delta_i\}$.

Thus, it is easy to verify that $S'_1 \leq S'_2$ if $S_1 \leq S_2$. □

Similarly, for the cases of push and pop transitions, the next two lemmas hold:

**Lemma 8.** *Let $S_1 \xrightarrow{a/+(S_1,a)} Id_{R'_1}$ and $S_2 \xrightarrow{a/+(S_2,a)} Id_{R'_2}$ be push transitions of the determinized VPA $M^{od}$. If $S_1 \leq S_2$, we have $Id_{R'_1} \leq Id_{R'_2}$.*

**Lemma 9.** *Let $S_1 \xrightarrow{a/-(S'_1,a')} S''_1$ and $S_2 \xrightarrow{a/-(S'_2,a')} S''_2$ be pop transitions of the determinized VPA $M^{od}$. Assume that $(S'_1,a') \leq (S'_2,a')$ and $S_1 \leq S_2$. Then, we have $S''_1 \leq S''_2$.*

Now, we are in a position to extend the ordering in Definition 6 to an ordering over configurations of the determinized VPA $M^{od}$.

**Definition 10.** *Let $c_1 = (S_1, \gamma_n \cdots \gamma_1 \bot)$ and $c_2 = (S_2, \gamma'_n \cdots \gamma'_1 \bot)$ be two configurations of $M^{od}$. We say $c_1 \leq c_2$ iff the following conditions hold: $S_1 \leq S_2$, and $\gamma_i \leq \gamma'_i$ for all $1 \leq i \leq n$.*

**Lemma 11.** *Let $c_1 = (S_1, \gamma_n \cdots \gamma_1 \bot)$ and $c_2 = (S_2, \gamma'_n \cdots \gamma'_1 \bot)$ be configuration of $M^{od}$ such that $c_1 \leq c_2$. For any word $w = a_1 \cdots a_k \in \Sigma^*$, if $(S_1, \gamma_n \cdots \gamma_1 \bot) \xrightarrow{w} (\bar{S}_1, \bar{\gamma}_m \cdots \bar{\gamma}_1 \bot)$ and $(S_2, \gamma'_n \cdots \gamma'_1 \bot) \xrightarrow{w} (\bar{S}_2, \bar{\gamma}'_m \cdots \bar{\gamma}'_1 \bot)$, then $(\bar{S}_1, \bar{\gamma}_m \cdots \bar{\gamma}_1 \bot) \leq (\bar{S}_2, \bar{\gamma}'_m \cdots \bar{\gamma}'_1 \bot)$*

*Proof.* We prove this lemma by induction on the length $|w|$ of $w$. If $|w| = 1$, it means that $w = a$. The proof immediately follows from Lemma 7, Lemma 8, or 9 wrt. the type of input symbol $a$. Now, assume that the lemma holds for the case $|w| = i$. Again, using induction hypothesis and Lemmas 7, 8 or 9, it is easy to verify that this lemma also holds for the case $|w| = i + 1$. The lemma is proved. □

**Algorithm 3.** Extract minimal transitions of P-automata at each incremental step

```
Data: A set of transitions $T(A)$ of $\mathcal{P}$-automaton $A$
Result: A set of "minimal" transitions T(A) of $A$
1 begin
2     for each state $S$ of $A$ do
3         if $(S_1,\gamma_1,S) \in T(A) \wedge (S_2,\gamma_2,S) \in T(A)$ such that: $S_1 \leq S_2$ and
          $\gamma_1 \leq \gamma_2$ then
4             $T(A) \longleftarrow T(A) \backslash \{(S_2,\gamma_2,S)\}$.
5     return T(A);
```

It is crucial to note that, $L(M) \neq \Sigma^*$, iff there exists a *rejecting* reachable configuration of $M^{od}$. Recall that a configuration $(S, \sigma)$ is rejecting if $\Pi_2(S) \cap F = \varnothing$. Note that if $(S, \sigma) \leq (S', \sigma')$ and $\Pi_2(S') \cap F = \varnothing$, then $\Pi_2(S) \cap F = \varnothing$. Based on this observation and Lemma 11, it is sufficient to compute only minimal reachable configurations and check for the existence of a *rejecting* configuration. Formally, we define the set of *minimal reachable* configurations of a determinized VPA $N$ as follows:

**Definition 12.** $\mathsf{MinimalReachableConf}(N) = \{(s, \sigma) \in ReachableConf(N) \mid \neg\exists(s', \sigma') \in ReachableConf(N) \cdot (s', \sigma') \leq (s, \sigma)\}$.

Let $C_0 = \{(Id_{Q_0}, \bot)\}$ be the set of initial configurations of $M^{od}$. Let $A_{post^*(C_0)}$ be the $\mathcal{P}$-automaton for presenting the set of $\mathsf{ReachableConf}(M^{od})$. Let $A$ be the $\mathcal{P}$-automaton that is obtained from $A_{post^*(C_0)}$ at each incremental expansion step as follows: for two configurations $(S_1, \gamma_1\sigma)$ and $(S_2, \gamma_2\sigma)$, we only need to compare the states (*i.e.,* $S_1$ and $S_2$) and top-of-stack symbols (*i.e.,* $\gamma_1$ and $\gamma_2$). Assume that $S_1 \leq S_2$ and $\gamma_1 \leq \gamma_2$, then $(S_1, \gamma_1\sigma) \leq (S_2, \gamma_2\sigma)$ . So, we only need to keep the "smaller" configuration $(S_1, \gamma_1\sigma)$. We formalize this observation in Algorithm 3.

**Algorithm 4.** The optimized on-the-fly algorithm for university checking of VPA

```
Data: A nondeterministic VPA $M = (Q, \Gamma, Q_0, \Delta, F)$
Result: Universality of $M$
1  begin
2      Create the initial state of the minimal determinized VPA $Md$ using
       Algorithm 1;
3      Initiate $\mathcal{P}$-automaton $A$ to represent the initial configuration of $Md$;
4      Create transitions of $Md$ departing from the initial state using Algorithm
       1;
5      while (the set of new transitions of $Md$ is not empty) do
6          Apply Algorithm 2 on the current $Md$ to create new transitions and
           states of $A$;
7          Then apply Algorithm 3 on the current $A$ to obtain the newly
           optimized $\mathcal{P}$-automaton $A$;
8          if a rejecting state is added to $A$ then
9              return False;
10         Apply Algorithm 1 on new states of $A$ to create new transitions of $Md$;
11     return True;
```

**Theorem 13 (Correctness of Algorithm 4).** *Let $M$ be a nondeterministic VPA. Let $M_i$ and $A_i$ be deteminized VPA and the P-automaton obtained in the $i^{th}$ repetition of the while-loop in Algorithm 4. The VPA M is not universal if and only if $L(A_i) \cap \mathsf{RejectingConf}(M^i) \neq \emptyset$ for some $i$.*

*Proof.* If $L(A_i) \cap \mathsf{RejectingConf}(M_i) \neq \varnothing$ for some $i$ then of course $M$ is not universal. Conversely, if $M$ is not universal, there exists at least one rejecting reachable configuration $q = (S, \sigma)$ (i.e., $\Pi_2(S) \cap F = \emptyset$). If $q \in L(A_i)$ for some $i$, then the theorem holds. If $q \notin L(A_i)$ for any $i$ (within a bound), it means that $q$ was out of $L(A_i)$ by a removing action at a certain step of the incremental expansion using Algorithm 3. Thus, there is at least one reachable configuration $q' = (S', \sigma')$, such that $q' \leq q$ and $q' \in L(A_i)$. Because $q' \leq q$, $S' \leq S$ and $\Pi_2(S') \cap F = \emptyset$, and thus $q'$ is a rejecting configuration. As a result, there exists an $i$ such that $q' \in L(A_i) \cap \mathsf{RejectingConf}(M_i)$, the theorem holds. □

**Complexity:** In the worst case (i.e., the automaton is universal), the complexity of our proposed algorithm is the same as of the standard one, $O(2^{3n^2})$ where $n$ are numbers of states of $M$ (this is because checking emptiness of VPA is cubic time [2]). However, in the case of not universal, our methods outperforms the standard one as the experiments will show in the next section.

### 3.3 Inclusion Checking

To check whether $L(A) \subseteq L(B)$, the standard method is to check whether $L(A \times \overline{B}) = \emptyset$, where $\overline{B}$ is the complement of $B$. For inclusion checking of VPA, we first determinize $B$, take its complement $\overline{B}$) and make product with A incrementally step by step ( denoted by $A \times \overline{B}$). Concurrently, we check for reachability of this product automaton using on-the-fly manner. If there is a reachable state $(q, s)$ such that $q \in F_A$ and $s \cap F_B = \emptyset$ ($s$ is an accepting state of $\overline{B}$). In this case, there exists a word $w$ such that $w \in L(A)$ and $w \notin L(B)$. In this case we stop and report that $L(A) \nsubseteq L(B)$. Although our formalization is different from the antichain of finite automata [7], the intuition behind our method is the same as theirs. In other words, the on-the-fly approach tries to find if there exists at least a word $w \in L(A) \setminus L(B)$. If such a word $w$ was found, we can conclude that $L(A) \nsubseteq L(B)$. Otherwise, $L(A)$ is a subset of $L(B)$. In the worst case, the complexity is $O(m^3 \cdot 2^{3n^2})$ for VPA inclusion checking, where $m$ and $n$ are numbers of states of $A$ and $B$.

## 4 Implementation and Experiments

We have implemented the above approaches, on the top of VPAlib [1], for testing universality and inclusion of VPA in a prototype tool named VPAchecker. The package is implemented in Java 1.5.0 on Windows XP. To compare the optimized algorithm with the standard algorithm, we run our implementations on randomly generated VPA. All tests are performed on a PC equipped with 1.50 GHz Intel® Core$^{\text{TM}}$ Duo Processor L2300 and 1.5 GB of memory. During experiments, we fix the size of the input alphabet to $|\Sigma_c| = |\Sigma_r| = |\Sigma_i| = 2$, and the size of the stack alphabet to $|\Gamma| = 3$. The *density of final states* $f = \frac{|F|}{|Q|}$ is the ratio of number of

[1] http://www.emn.fr/x-info/hnguyen/vpa/

**Table 1.** Universality checking for VPA generated by r-random (50 automata for each sample)

| | *number of states* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ON-THE-FLY-OPT | 5 | 10 | 20 | 40 | 60 | 80 | 100 | 150 |
| no. of success | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 47 |
| total time (s) | 19 | 35 | 43 | 87 | 147 | 222 | 496 | 336 |
| no. of timeout (*60* s) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |
| | *number of states* | | | | | | | |
| ON-THE-FLY | 5 | 10 | 20 | 40 | 60 | 80 | 100 | 150 |
| no. of success | 50 | 50 | 50 | 50 | 50 | 50 | 46 | 47 |
| total time | 23 | 46 | 52 | 110 | 210 | 247 | 686 | 372 |
| no. of timeout (*60* s) | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 3 |
| | *number of states* | | | | | | | |
| STANDARD | 5 | 10 | 20 | 40 | 60 | 80 | 100 | 150 |
| no. of success | 21 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| total time (s) | 456 | 31 | 0 | 0 | 0 | 0 | 0 | 0 |
| no. of timeout (*60* s) | 29 | 49 | 50 | 50 | 50 | 50 | 50 | 50 |

**Table 2.** Universality checking for VPA generated by r-regular random (50 automata for each sample)

| | *number of states* | | | | | | |
|---|---|---|---|---|---|---|---|
| ON-THE-FLY-OPT | 5 | 10 | 15 | 20 | 30 | 40 | 50 |
| no. of success | 50 | 45 | 19 | 2 | 0 | 0 | 0 |
| total time (s) | 52 | 833 | 892 | 86 | 0 | 0 | 0 |
| no. of timeout (*180* s) | 0 | 5 | 31 | 48 | 50 | 50 | 50 |
| | *number of states* | | | | | | |
| ON-THE-FLY | 5 | 10 | 15 | 20 | 30 | 40 | 50 |
| no. of success | 50 | 43 | 13 | 0 | 0 | 0 | 0 |
| total time | 68 | 1425 | 754 | 0 | 0 | 0 | 0 |
| no. of timeout (*180* s) | 0 | 7 | 37 | 50 | 50 | 50 | 50 |
| | *number of states* | | | | | | |
| STANDARD | 5 | 10 | 15 | 20 | 30 | 40 | 50 |
| no. of success | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| total time (s) | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| no. of timeout (*180* s) | 50 | 50 | 50 | 50 | 50 | 50 | 50 |

**Table 3.** Checking inclusion with $r(q, a) = 2$, $f = 1$

| | *number states of A and B* | | | | |
|---|---|---|---|---|---|
| ON-THE-FLY-OPT | (10,5) | (100,5) | (500,5) | (1000,5) | (3000,5) |
| success | 20 | 20 | 5 | 5 | 2 |
| time(s) | 27 | 910 | 336 | 1257 | 357 |
| timeout (*300* s) | 0 | 0 | 15 | 15 | 18 |

final states over the number of states of a VPA. We first set parameters of the tests as follows:

**Definition 14 (r-random).** *The* density of final states $f = \frac{|F|}{|Q|} = 1$ *and the* density of transitions $r = \frac{k_a}{|Q|} = 2$*, where* $k_a$ *is the number of transitions for each input symbol* $a$*,* $F$ *and* $Q$ *are set of final states and states of VPA, respectively.*

We ran our tests on randomly VPA generated by the parameter r-random. We have tried VPA sizes from 10 to 100. We generated 50 VPA for each sample point, and setting timeout to 60 seconds. The experimental results are given in Table 1. We found that all successfully checked VPA are not universal, and thus we omit the row for universal results in the table. The experiments shows that STANDARD can solve for generated VPA instances with 5 states only. It becomes stuck when the number of states greater than or equal to 10. Meanwhile, ON-THE-FLY is significantly more efficient than STANDARD, it can check for almost all VPA. We also applied optimization of $\mathcal{P}$-automaton and implemented it as ON-THE-FLY-OPT. Based on experimental results, ON-THE-FLY-OPT is a little bit better than ON-THE-FLY. The parameter r-random does not guarantee the completeness of VPA. Therefore, the probability of being universal is very low. In order to increase the probability of being universal, we define a new parameter as below:

**Definition 15 (r-regular random).** *The* density of final states$f = \frac{|F|}{|Q|} = 1$ *and the* density of transitions $r : Q \times \Sigma \rightarrow N$*;* $r(q, a)$ *depends on not only the input symbol* $a$ *but also on the state* $q$*. In particular, we select:* $r(q, a) = 2$ *for all* $q \in Q$ *and* $a \in \Sigma_c$*,* $r(q, b) = 6$ *for all* $q \in Q$ *and* $b \in \Sigma_r$*, and* $r(q, c) = 2$ *for all* $q \in Q$ *and* $c \in \Sigma_i$*.*

With r-regular random, a VPA with 10 states has 200 transitions. We again test for various sizes of VPA from 5 to 50. We ran with 50 samples for each point, setting timeout to 180 seconds. The results are reported in Table 2. For this parameter, results of STANDARD are all timeout even with only 5 states. ON-THE-FLY behaves in significantly better ways than those of STANDARD. Since VPA generated by r-regular random parameter are universal, all three algorithms work more slowly. Especially, if the number of states is greater than 30, results of ON-THE-FLY are all timeout. Similarly, ON-THE-FLY-OPT is a little bit better than ON-THE-FLY for this parameter.

We also performed experiments for inclusion checking $L(A) \subseteq L(B)$. For this, we selected parameter r-regular random for generating the most difficult instances of inclusion checking. This is because using r-regular random both $A$ and $B$ are universal, and thus $L(A)$ is included in $L(B)$. We generated various sizes of $A$ (10, 100, 200, 500, 1000, and 3000 states) and $B$ (5 and 10 states). We ran with 20 samples for each point, setting timeout to 300 seconds. The experimental results are summarized in Table 3. There we only list the number of tests that finish within timeout. The detailed results are reported in Table 3. We see that STANDARD does not work well, it get all timeout for the smallest size $(10, 5)$. The results show that ON-THE-FLY-OPT outperforms STANDARD.

## 5 Conclusion

In this paper we presented the optimized on-the-fly algorithms for testing universality and inclusion of nondeterministic VPA. In summary, to check universality of a nondeterministic VPA $M$, the intuition idea behind on-the-fly manner is try to find whether there exists a word $w$ such that $w \notin L(M)$. Similarly, to check inclusion $L(M) \subseteq L(N)$, the intuition idea is to find whether there exists at least a word $w$ such that $w \in L(M) \backslash L(N)$. All algorithms have been implemented in a prototype tool. Although the ideas of our methods are simple, the experimental results showed that the proposed algorithms are considerably faster than the standard ones.

We should emphasize that much work needs to be done in the future, which includes, *e.g.,* (1) consider how to apply the tool to case studies in practice, for example, checking correctness requirements for XML streams. At the moment, the data structures for VPA are rather naive. That is reason why the running time of our tool is not fast. It would be interesting to explore a more compact data structure; for this, (2) we plan to manipulate VPA using BDD-based representation. Currently, we have been working on aspects (1) and (2). The preliminary results related to these mentioned future work will be reported in another opportunity.

**Acknowledgements.** We would like to thank Prof. Mizuhito Ogawa, Dr. Nao Hirokawa of JAIST and anonymous reviewers for helpful comments to improve this paper.

## References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Babai, L. (ed.) STOC, pp. 202–211. ACM (2004)
2. 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)
3. Kumar, V., Madhusudan, P., Viswanathan, M.: Visibly pushdown automata for streaming xml. In: WWW, pp. 1053–1062 (2007)
4. Nguyen, D.H., Südholt, M.: Vpa-based aspects: better support for aop over protocols. In: SEFM, pp. 167–176. IEEE Computer Society (2006)
5. Nguyen, T.V.: A tighter bound for determinization of visibly pushdown automata. In: INFINITY, vol. 10, pp. 62–76 (2009)
6. Pitcher, C.: Visibly pushdown expression effects for xml stream processing. In: PLAN-X, pp. 5–19 (2005)
7. De Wulf, M., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Antichains: A New Algorithm for Checking Universality of Finite Automata. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 17–30. Springer, Heidelberg (2006)

# Automaton-Based Array Initialization Analysis

Đurica Nikolić and Fausto Spoto

Dipartimento di Informatica, Università di Verona, Italy
`{durica.nikolic,fausto.spoto}@univr.it`

**Abstract.** We define an automaton-based abstract interpretation of a trace semantics which identifies loops that definitely initialize all the elements of an array, a useful piece of information for the static analysis of imperative languages. This results in a fully automatic and fast analysis, that does not use manual code annotations. Its implementation inside the Julia analyzer is efficient and precise.

## 1 Introduction

This work was born from a problem faced during the static analysis of Java and Android programs. Fig. 1 shows an example: fields `mOriginal` and `mRotated` hold arrays, initialized by `readModel()` and then read and dereferenced by other methods. In this case the `null`-pointer analysis of Julia [12] issued spurious warnings since it could not prove that the elements of these arrays were initialized.

We wanted to prove, automatically, that *all elements of fields `mOriginal` and `mRotated` are initialized at point **. Complete initialization of arrays to some value is undecidable [2,9], but static analysis can often prove it. Typically, theorem proving or predicate abstraction are used [7,8]. However, not all techniques are automatic; some require a previous manual annotation of the program with invariants; others must be instantiated with different abstract domains (or predicate abstractions), depending on the specific program at hand; others have an overwhelming cost. An impressive evaluation of those techniques has been done in [6]: its authors implemented and compared them, with the result that very few are completely automatic and none efficient for real large software. Moreover, they present a new technique based on abstract interpretation [3,4]: a fixpoint over an abstraction of arrays into *segments* with strict or non-strict bounds. The abstraction of the elements of a single segment is given over an abstract domain, left parametric.

Our contributions are:

1. a new automaton-based abstract interpretation of execution traces, proving that all elements of an array are definitely initialized at a program point;
2. a proof of correctness for the previous analysis;
3. experiments showing that the analysis is efficient (1 second on average for large software) precise and scales to the analysis of more than $100,000$ lines.

Our static analysis is intraprocedural, but aware of interprocedural side-effects. It is a whole-program analysis, hence it does not apply to classes in isolation nor to libraries. We give definitions and proofs for an automaton spotting the full

```
private void readModel(String prefix) {
  ....
  String[] p = ....
  int numpoints = p.length;
  this.mOriginal = new ThreeDPoint[numpoints];
  this.mRotated = new ThreeDPoint[numpoints];

  for (int i = 0; i < numpoints; i++) {
    this.mOriginal[i] = new ThreeDPoint();
    this.mRotated[i] = new ThreeDPoint();
    String[] coord = p[i].split("␣");
    this.mOriginal[i].x=Float.valueOf(coord[0]);
    this.mOriginal[i].y=Float.valueOf(coord[1]);
    this.mOriginal[i].z=Float.valueOf(coord[2]);
  }
  [point *]
  ....
}
```

**Fig. 1.** A snippet of code from the `CubeWallpaper` Android program by Google

initialization of monodimensional arrays, with a standard pattern of initialization from element 0 upwards. This is, of course, a restriction, but covers the most frequent situations, as confirmed by our experiments in Section 6. We have also implemented some automata for full initialization of bidimensional arrays and for other orders of initialization, but they are not presented here due to space limitations, and we do not consider them a contribution of this paper.

Our abstract domain is rather simple: it contains only 5 elements. This permits us to provide a fully detailed soundness proof of low complexity. The key to this simplicity is abstract interpretation, that relates the semantics of the program with the pattern recognition ability of the automaton. The analysis in [6] is more general and precise than ours (for instance, it deals with out-of-order array initializations, while we are not able to do it), but this comes at the cost of a higher theoretical complexity. We could not perform an experimental comparison with them, since we analyze Java bytecode while the Clousot analyzer of [6] works for .NET. Also, the great precision of [6] largely depends on its specific instantiation and available supporting aliasing analysis.

One can apply our analysis also to position-based collection classes such as `java.util.ArrayList`. However, those Java classes have been devised in such a way that elements can be read only if they have already been set, or an exception is thrown. Hence, in Java, they are always fully initialized, potentially to `null`.

Proving full array initialization at a program point does not mean that the array is initialized *to some kind of values* (e.g. to non-`null` values) and that this remains true later, when the array is accessed, possibly in different methods from the one which initializes it. The local initialization at a program point must be lifted to a global property at all program points where the array is read. An extended version of this paper [10] presents our solutions to these problems.

## 2 A Simple Imperative Language and Its Semantics

We present here a simplified imperative language, inspired by [5]. It exposes only features relevant to our work. Namely, it has a minimal set of types, does

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\mathsf{E}$ | $::=$ | $n \mid x$ | Integer / Variable | $\mathsf{B}$ | $::=$ | $\mathtt{true} \mid \mathtt{false}$ | Truth/Falsity |
| | $\mid$ | $x.length$ | Array length | | $\mid$ | $\neg\mathsf{B}_1$ | Negation |
| | $\mid$ | $x[\mathsf{E}]$ | Array element | | $\mid$ | $\mathsf{E}_1 \textcircled{<} \mathsf{E}_2$ | $\textcircled{<} \in \{<, \leq, =\}$ |
| | $\mid$ | $\mathsf{E}_1 \oplus \mathsf{E}_2$ | $\oplus \in \{+, -, *, \div, \%\}$ | | $\mid$ | $\mathsf{B}_1 \textcircled{\vee} \mathsf{B}_2$ | $\textcircled{\vee} \in \{\wedge, \vee\}$ |
| $\mathsf{A}$ | $::=$ | $\mathsf{B}$ | Test | $\mathsf{C}$ | $::=$ | $\mathsf{L}_1 : \mathsf{A} \to \mathsf{L}_2;$ | Command |
| | $\mid$ | $x := \mathsf{E}$ | Variable assignment | | | | |
| | $\mid$ | $x := \mathtt{new}\ t[\mathsf{E}]$ | Creation of an array | $n \in \mathbb{Z},\ x \in \mathsf{Var},\ \mathsf{E}, \mathsf{E}_1, \mathsf{E}_2 \in \mathbb{E},$ | | | |
| | $\mid$ | $x[\mathsf{E}] := \mathsf{E}$ | Array element assign | $\mathsf{B}, \mathsf{B}_1, \mathsf{B}_2 \in \mathbb{B},\ \mathsf{A} \in \mathbb{A},\ \mathsf{C} \in \mathbb{C},\ t \in \mathsf{Type}$ | | | |

**Fig. 2.** Abstract syntax of programs

not include classes, structures, procedures (our analysis is intraprocedural) nor exceptions. The actual implementation of our analysis includes all features of monothreaded Java bytecode such as classes, method calls and exceptions.

In our language, commands are labeled *actions*. These actions are executed when the interpreter of the language is at a given, *initial* label and lead to another, *successor* label. More actions can share the same initial label and hence our language is, in general, non-deterministic. The exact nature of labels is irrelevant: we can assume, for instance, that they are integers.

**Definition 1 (Syntax of Programs).** *A* program *is a finite set of* commands, *with a distinguished* initial command $\mathsf{C}_{init}$. $\mathbb{C}$ *is the set of commands* $\mathsf{C}$ *of the form* $\mathsf{L}_1 : \mathsf{A} \to \mathsf{L}_2;$, *where* $\mathsf{L}_1$ *and* $\mathsf{L}_2$ *are called* initial *and* successor *labels of* $\mathsf{C}$, *and* $\mathsf{A}$ *is the* action *executed by* $\mathsf{C}$. *We define selectors* $\mathsf{ini}(\mathsf{C}) = \mathsf{L}_1$, $\mathsf{suc}(\mathsf{C}) = \mathsf{L}_2$ *and* $\mathsf{act}(\mathsf{C}) = \mathsf{A}$. *Actions can be Boolean expressions in* $\mathbb{B}$ *(whose definition uses arithmetic expressions in* $\mathbb{E}$*), creation of arrays and assignments to local variables in* $\mathsf{Var}$ *or to array elements, and they are defined by the grammar in Fig. 2. The set of* types, $\mathsf{Type}$, *is the minimal set containing* $\mathsf{int}$ *and* $\mathsf{array\ of}\ t$ *for every* $t \in \mathsf{Type}$. *We assume that every* $v \in \mathsf{Var}$ *has a static type* $\mathsf{t}(v)$.

We assume programs well-typed. For instance, given an action $x[y[3]] := z$, then $\mathsf{t}(y) = \mathsf{array\ of\ int}$ and $\mathsf{t}(x) = \mathsf{array\ of}\ \mathsf{t}(z)$. We let $\mathsf{vars}(D)$ stand for the variables occurring in an expression or action $D$, and $\mathsf{mod}(\mathsf{A}) \subseteq \mathsf{vars}(\mathsf{A})$ for the variables modified (i.e., assigned) by an action $\mathsf{A}$. Namely, $\mathsf{mod}(\mathsf{B}) = \varnothing$, $\mathsf{mod}(x := \mathsf{E}) = \{x\}$, $\mathsf{mod}(x := \mathtt{new}\ t[\mathsf{E}]) = \{x\}$ and $\mathsf{mod}(x[\mathsf{E}_1] := \mathsf{E}_2) = \varnothing$.

| | |
|---|---|
| 1. i = 0; | $\mathsf{C}_0$ 1: $i := 0 \to 2;$ |
| 2.**while** (i < a.length) { | $\mathsf{C}_1$ 2: $i < a.length \to 3;$ |
| 3. **if**(i % 3 == 0) { | $\mathsf{C}_2$ 2: $\neg(i < a.length) \to 9;$ |
| 4. a[i]=...; | $\mathsf{C}_3$ 3: $i\%3 = 0 \to 4;$ |
| } **else** { | $\mathsf{C}_4$ 3: $\neg(i\%3 = 0) \to 5;$ |
| 5. a[i]=...; | $\mathsf{C}_5$ 4: $a[i] := \ldots \to 8;$ |
| 6. i++; | $\mathsf{C}_6$ 5: $a[i] := \ldots \to 6;$ |
| 7. a[i]=...; | $\mathsf{C}_7$ 6: $i := i + 1 \to 7;$ |
| } | $\mathsf{C}_8$ 7: $a[i] := \ldots \to 8;$ |
| 8. i++; | $\mathsf{C}_9$ 8: $i := i + 1 \to 2;$ |
| } | $\mathsf{C}_{10}$ 9: $\cdots$ |
| 9. . . . | |

*Example 1.* The figure on the left shows a Java loop (left) initializing an array $a$ and its corresponding transition system (right). This code fragment is well-typed with $\mathsf{t}(i) = \mathsf{int}$ and $\mathsf{t}(a) = \mathsf{array\ of\ int}$.

At run-time, variables hold values which, in a programming language such as Java, may be primitive or non-primitive; the latter include objects and arrays. Def. 2 simplifies the picture by only considering integers as primitive values and arrays as non-primitive values. That simplification does not limit the results of this paper, that is only concerned about arrays and integer counters.

**Definition 2 (Values).** Values *are elements of* $\mathsf{Val} = \mathbb{Z} \cup \mathbb{L} \cup \{\boldsymbol{null}\}$, *where* $\mathbb{L}$ *is a finite set of* memory locations. $\mathsf{Arr}$ *is the set of arrays* $a = \langle n, [v_0, \ldots, v_{n-1}]\rangle$, *where* $n \in \mathbb{N}$ *is the length of* $a$ *and* $v_i \in \mathsf{Val}$ *are its elements, for* $i \in [0..n) \subseteq \mathbb{N}$. *We define* $a.\mathsf{length} = n$ *and* $a[i] = v_i$, *for* $i \in [0..n)$. *We also define the* update *of* $a$ *at* $i$ *as* $a[i \mapsto v] = \langle n, [v_0, \ldots, v_{i-1}, v, v_{i+1}, \ldots, v_{n-1}]\rangle \in \mathsf{Arr}$, *which is undefined when* $i$ *is outside the range of* $a$. *A* memory *is a partial map* $\mu : \mathbb{L} \to \mathsf{Arr}$.

An environment represents the state of an interpreter of the language. It provides a value for each variable and specifies the memory of the system.

**Definition 3 (Environment).** *An* environment *is a pair* $e = \langle \rho, \mu\rangle$ *of a total map* $\rho : \mathsf{Var} \to \mathsf{Val}$ *and a memory* $\mu$. *As in Java, we ban dangling pointers, i.e.,* $\forall v \in \mathsf{Var}$, *if* $\rho(v) \in \mathbb{L}$, *then* $\mu(\rho(v))$ *is defined, and for every* $a \in \mathsf{Arr}$ *and* $\forall i \in [0..a.\mathsf{length})$, *such that* $a[i] \in \mathbb{L}$, *then* $\mu(a[i])$ *is defined. We require static types respected i.e.,* $\forall v \in \mathsf{Var}$ *we have* $\rho(v) \approx_\mu \mathsf{t}(v)$, *where (i)* $x \approx_\mu \mathsf{int}$ *iff* $x \in \mathbb{Z}$; *and (ii)* $x \approx_\mu \mathsf{array\ of}\ t$ *iff* $x = \boldsymbol{null}$ *or (*$x \in \mathbb{L}$ *and* $\forall i \in [0..\mu(x).\mathsf{length})$, $\mu(x)[i] \approx_\mu t$*). We let* $\mathcal{E}$ *be the set of all environments.*

**Definition 4 (Value of Expressions).** *The evaluations* $\mathcal{A}[\![\mathsf{E}]\!] : \mathcal{E} \to \mathsf{Val}$ *and* $\mathcal{B}[\![\mathsf{B}]\!] : \mathcal{E} \to \{\mathsf{true}, \mathsf{false}\}$ *of expressions are partial maps defined as*

$$\begin{aligned}
\mathcal{A}[\![n]\!]\langle\rho,\mu\rangle &= n & \mathcal{B}[\![\mathsf{true}]\!]e &= \mathsf{true}\\
\mathcal{A}[\![x]\!]\langle\rho,\mu\rangle &= \rho(x) & \mathcal{B}[\![\mathsf{false}]\!]e &= \mathsf{false}\\
\mathcal{A}[\![x.\mathit{length}]\!]\langle\rho,\mu\rangle &= \mu(\rho(x)).\mathsf{length} & \mathcal{B}[\![\neg\mathsf{B}]\!]e &= \neg\mathcal{B}[\![\mathsf{B}]\!]e\\
\mathcal{A}[\![x[\mathsf{E}]]\!]\langle\rho,\mu\rangle &= \mu(\rho(x))[\mathcal{A}[\![\mathsf{E}]\!]\langle\rho,\mu\rangle] & \mathcal{B}[\![\mathsf{E}_1 \otimes \mathsf{E}_2]\!]e &= \mathcal{A}[\![\mathsf{E}_1]\!]e \otimes \mathcal{A}[\![\mathsf{E}_2]\!]e\\
\mathcal{A}[\![\mathsf{E}_1 \oplus \mathsf{E}_2]\!]\langle\rho,\mu\rangle &= \mathcal{A}[\![\mathsf{E}_1]\!]\langle\rho,\mu\rangle \oplus \mathcal{A}[\![\mathsf{E}_2]\!]\langle\rho,\mu\rangle & \mathcal{B}[\![\mathsf{B}_1 \circledvee \mathsf{B}_2]\!]e &= \mathcal{B}[\![\mathsf{B}_1]\!]e \circledvee \mathcal{B}[\![\mathsf{B}_2]\!]e.
\end{aligned}$$

*Both maps are undefined when their defining expression is undefined or when any of of their arguments is undefined.*

The execution of an action maps an initial environment into one of its successors.

**Definition 5 (Semantics of Actions).** *The semantics of an action* $\mathsf{A}$ *is a partial map* $\mathcal{S}[\![\mathsf{A}]\!] : \mathcal{E} \to \mathcal{E}$ *defined as*

$$\mathcal{S}[\![\mathsf{B}]\!]e = \begin{cases} e & \text{if } \mathcal{B}[\![\mathsf{B}]\!]e = \mathsf{true}\\ \text{undefined} & \text{otherwise}\end{cases}$$

$$\mathcal{S}[\![x := \mathsf{E}]\!]\langle\rho,\mu\rangle = \langle\rho[x \mapsto \mathcal{A}[\![\mathsf{E}]\!]\langle\rho,\mu\rangle], \mu\rangle$$

$$\mathcal{S}[\![x[\mathsf{E}_1] := \mathsf{E}_2]\!]\langle\rho,\mu\rangle = \begin{cases} \langle\rho, \mu[l \mapsto \mu(l)[i \mapsto \mathcal{A}[\![\mathsf{E}_2]\!]\langle\rho,\mu\rangle]]\rangle\\ \qquad \text{if } l = \rho(x),\ l \neq \boldsymbol{null}\\ \qquad i = \mathcal{A}[\![\mathsf{E}_1]\!]\langle\rho,\mu\rangle \text{ and } 0 \leqslant i < \mu(l).\mathsf{length}\\ \text{undefined otherwise}\end{cases}$$

$$\mathcal{S}[\![x := \boldsymbol{new}\ t[\mathsf{E}]]\!]\langle\rho,\mu\rangle = \langle\rho[x \mapsto l_f], \mu[l_f \mapsto \langle\mathcal{A}[\![\mathsf{E}]\!]\langle\rho,\mu\rangle, [\mathsf{def}, \ldots, \mathsf{def}]\rangle]\rangle,$$

*where* $l_f$ *is a fresh location i.e.,* $l_f \notin \mathsf{dom}(\mu)$ *and* $\mathsf{def}$ *is the default value for* $t$. *This map is undefined when any of its arguments is undefined.*

Our operational semantics works over execution traces of states. A state is an environment enriched with a component recording the next command to be executed, similar to the program counter in an actual interpreter of the language.

**Definition 6 (State).** *A* state *is a pair $\sigma = \langle e, \mathsf{C}\rangle \in \mathcal{E} \times \mathbb{C}$. The set of states is denoted by $\Sigma$. We define the selectors* $\mathsf{env}(\sigma) = e$ *and* $\mathsf{cmd}(\sigma) = \mathsf{C}$*.*

A trace is a sequence of states that reflects an actual execution of the program.

**Definition 7 (Trace).** *A* finite partial trace *$\tau$ of states is a finite sequence of states $\langle \sigma_1, \ldots, \sigma_n\rangle$. For every $1 \leqslant i < n$, if $\sigma_i = \langle e, \mathsf{C}\rangle$, we require that $\mathcal{S}[\![\mathsf{act}(\mathsf{C})]\!]e$ is defined and that $\sigma_{i+1} = \langle \mathcal{S}[\![\mathsf{act}(\mathsf{C})]\!]e, \mathsf{C}'\rangle$ with $\mathsf{suc}(\mathsf{C}) = \mathsf{ini}(\mathsf{C}')$. When $n = 0$, the trace is empty and denoted by $\epsilon$. Otherwise, we define $\mathsf{first}(\tau) = \sigma_1$ and $\mathsf{last}(\tau) = \sigma_n$. The set of traces is denoted by $\mathcal{T}$. The* concatenation $\circ$ *of two traces is defined as $\tau_1 \circ \epsilon = \tau_1$, $\epsilon \circ \tau_2 = \tau_2$ and $\langle \sigma^1_1, \ldots, \sigma^1_{n_1}\rangle \circ \langle \sigma^2_1, \ldots, \sigma^2_{n_2}\rangle = \langle \sigma^1_1, \ldots, \sigma^1_{n_1}, \sigma^2_1, \ldots, \sigma^2_{n_2}\rangle$ if the latter is a trace; it is undefined otherwise.*

We define the operational semantics of our language as a transformer of sets of traces: it expands every trace $\tau$ with a state whose next command to be executed is a given command $\mathsf{C}$ that can be attached to $\tau$ according to Def. 7.

**Definition 8 (Operational Semantics).** *Let $\mathsf{C} \in \mathbb{C}$ and $\Rightarrow^{\mathsf{C}} : \wp(\mathcal{T}) \to \wp(\mathcal{T})$ be defined as $T \Rightarrow^{\mathsf{C}} \{\tau \circ \langle e, \mathsf{C}\rangle \mid \tau \in T \wedge e \in \mathcal{E} \wedge \tau \circ \langle e, \mathsf{C}\rangle$ is defined$\}$. The* operational semantics *at $\mathsf{C}$ is the set $@\mathsf{C}$ of all possible traces that lead to $\mathsf{C}$ and start with the execution of the distinguished command $\mathsf{C}_{init}$, that is, $@\mathsf{C} = \{\tau \in T_n \mid \exists T_1, \ldots, T_n \subseteq \mathcal{T}. \{\epsilon\} \Rightarrow^{\mathsf{C}_1} T_1 \Rightarrow^{\mathsf{C}_2} T_2 \cdots \Rightarrow^{\mathsf{C}_n} T_n \wedge \mathsf{C}_1 = \mathsf{C}_{init} \wedge \mathsf{C}_n = \mathsf{C}\}$.*

## 3 Regular Trace Approximation

We define here an approximation of the execution traces of a program through a finite deterministic automaton. Its states are sets of traces and represent elements of an abstract domain. This is defined through regular expressions specifying sequences of commands that can be executed to construct the traces. We prove that the transition relation of the automaton is a correct approximation of the $\Rightarrow^{\mathsf{C}}$ operational relation. The automaton (Fig. 3) is designed for a pair of program variables $a$, of type array, and $i$, of integer type. It is defined over the *alphabet* $\Lambda = \{\mathbf{0}, +, =, \geqslant, \mathrm{I}, \mathrm{R}\}$ and has *states* $S = \{\mathsf{INIT}, \mathsf{START}, \mathsf{WRITTEN}, \mathsf{ACCEPT}\}$. Its *transition relation* is a function $\delta : S \times \Lambda \to S$: given states $\mathsf{p}, \mathsf{q} \in S$ and $\lambda \in \Lambda$, if the automaton has a transition from $\mathsf{p}$ to $\mathsf{q}$ labelled by $\lambda$, then $\delta(\mathsf{p}, \lambda) = \mathsf{q}$.

The alphabet is an abstraction of the commands of the program.

**Definition 9 (Abstraction of Commands).** *Consider an array $a$ and an index integer variable $i$. The* abstraction of commands, *$s : \mathbb{C} \to \Lambda$, is defined as: $s(\langle \mathsf{L}_1 : \mathsf{A} \to \mathsf{L}_2;\rangle) = \lambda$, where the abstract value $\lambda$ can be $\mathbf{0}$, $+$, $=$ or $\geqslant$, if action $\mathsf{A}$ is $i := 0$, $i := i + 1$, $a[i] := \mathsf{E}$ or $\neg(i < a.length)$ respectively. Otherwise, if $\mathsf{A}$ is such that $\mathsf{mod}(\mathsf{A}) \cap \{a, i\} = \varnothing$, i.e., if the local variables assigned by $\mathsf{A}$ are not $a$ nor $i$, then $\lambda = \mathrm{I}$. In all other cases, $\lambda = \mathrm{R}$.*

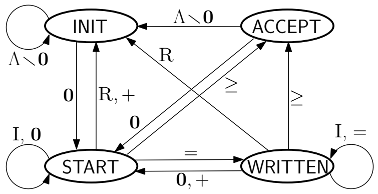


**Fig. 3.** Automaton detecting fully initialized arrays

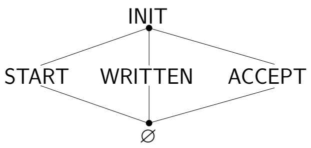


**Fig. 4.** The abstract domain $\mathbb{A}$

Note that assignments to $a[i]$ are abstracted into = but any assignment to any other element of the array (such as to $a[i+1]$) is considered *irrelevant* (I). This abstraction is syntactical: a command with action $i := 0$ is abstracted into $\mathbf{0}$; another with action $i := 1 - 1$ into R. This does not affect correctness (Theorem 1) but one might simplify and normalize the actions, making the abstraction more semantical and precise. We have not implemented this improvement.

**Definition 10 (Abstraction of Traces).** *The* abstraction of traces *is given by* $\beta : \mathcal{T} \rightarrow \Lambda^*$ *and defined as* $\beta(\langle\sigma_1, \ldots, \sigma_{n-1}, \sigma_n\rangle) = \beta(\langle\sigma_1, \ldots, \sigma_{n-1}\rangle)s(\mathsf{cmd}(\sigma_n)) = s(\mathsf{cmd}(\sigma_1)) \ldots s(\mathsf{cmd}(\sigma_n))$*, for non-empty traces, with* $\beta(\epsilon) = \epsilon$*.*

Since $\Lambda$ contains abstractions of commands, the meaning of the states of the automaton, $S$, becomes clearer. As we formalize below (Def. 11), INIT means that nothing is known about the last executed commands; START means that an assignment $i := 0$ is executed, and potentially followed by an alternation of assignments to $a[i]$ and unitary increments of $i$; WRITTEN means that an assignment to $a[i]$ has just been executed and the automaton is waiting to match it with a corresponding unitary increment of $i$; ACCEPT means that the complete initialization of the array can be asserted. An arbitrary number of irrelevant actions can always be executed between relevant actions.

**Definition 11 (Abstract Domain $\mathbb{A}$).** *The states of the automaton in Fig. 3 correspond to the following sets of traces defined by regular expressions over* $\Lambda$*:* $\mathsf{INIT} = \{\tau \in \mathcal{T} \mid \beta(\tau) \in \Lambda^*\} = \mathcal{T}$*,* $\mathsf{START} = \{\tau \in \mathcal{T} \mid \beta(\tau) \in \Lambda^*\mathbf{0}\mathrm{I}^*((=\mathrm{I}^*)^+ + \mathrm{I}^*)^*\}$*,* $\mathsf{WRITTEN} = \{\tau \in \mathcal{T} \mid \beta(\tau) \in \Lambda^*\mathbf{0}\mathrm{I}^*((=\mathrm{I}^*)^+ + \mathrm{I}^*)^*(=\mathrm{I}^*)^+\}$*,* $\mathsf{ACCEPT} = \{\tau \in \mathcal{T} \mid \beta(\tau) \in \Lambda^*\mathbf{0}\mathrm{I}^*((=\mathrm{I}^*)^+ + \mathrm{I}^*)^*(=\mathrm{I}^*)^* \geqslant\}$*, where* $*$ $(^+)$ *means zero or more (at least one) repetitions. We define the set* $\mathbb{A} = \{\mathsf{INIT}, \mathsf{START}, \mathsf{WRITTEN}, \mathsf{ACCEPT}, \varnothing\}$*.*

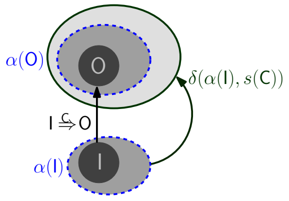
**Proposition 1.** $\mathbb{A}$ *is a Moore family of* $\wp(\mathcal{T})$ *i.e., it is an abstract domain ordered by set inclusion (Fig. 4). As standard for Moore families, the induced abstraction map* $\alpha : \wp(\mathcal{T}) \rightarrow \mathbb{A}$ *is* $\alpha(T) = \bigcap_{A \in \mathbb{A},\, T \subseteq A} A$*, for every* $T \subseteq \mathcal{T}$*.*

*Proof.* The last relevant instruction of any $\tau \in \mathcal{T}$ is $\mathbf{0}$ or +, if $\tau \in \mathsf{START}$; is =, if $\tau \in \mathsf{WRITTEN}$; is $\geqslant$, if $\tau \in \mathsf{ACCEPT}$. The intersection of these elements is $\varnothing \in \mathbb{A}$. Since $\mathsf{INIT} = \mathcal{T}$, we have that $\forall a \in \mathbb{A}$, $a \cap \mathsf{INIT} = a$. Hence $\mathbb{A}$ is a Moore family. □

Lemma 1 states a consistency or correctness relation [4] between the operational semantics and the transitions of the automaton in Fig. 3.

**Lemma 1.** *Let* $\mathsf{C} \in \mathbb{C}$ *and* $\mathsf{I}, \mathsf{O} \subseteq \mathcal{T}$*. If* $\mathsf{I} \Rightarrow^{\mathsf{C}} \mathsf{O}$ *then* $\alpha(\mathsf{O}) \subseteq \delta(\alpha(\mathsf{I}), s(\mathsf{C}))$*.*

*Proof.* Let $\tau \in \mathsf{O}$. By Def. 8, there exist $\tau' \in \mathsf{I}$ and $e \in \mathcal{E}$ s.t. $\tau = \tau' \circ \langle e, \mathsf{C} \rangle$, and therefore $\beta(\tau) = \beta(\tau')s(\mathsf{C})$. We proceed by case analysis. If $\alpha(\mathsf{I}) = \mathsf{START}$ and $s(\mathsf{C}) = \mathbf{0}$, then $\tau' \in \mathsf{I} \subseteq \alpha(\mathsf{I}) = \mathsf{START}$ and therefore $\beta(\tau) = \beta(\tau')s(\mathsf{C}) \in \Lambda^*\mathbf{0}\mathrm{I}^*((=\mathrm{I}^*)^+ + \mathrm{I}^*)^* s(\mathsf{C}) = \Lambda^*\mathbf{0}\mathrm{I}^*((=\mathrm{I}^*)^+ + \mathrm{I}^*)^*\mathbf{0} \subseteq \Lambda^*\mathbf{0} \subseteq \Lambda^*\mathbf{0}\mathrm{I}^*((=\mathrm{I}^*)^+ + \mathrm{I}^*)^*$. Hence, $\tau \in \mathsf{START}$. Since $\tau$ is arbitrary, $\mathsf{O} \subseteq \mathsf{START}$ and hence $\alpha(\mathsf{O}) \subseteq \alpha(\mathsf{START}) = \mathsf{START} = \delta(\mathsf{START}, \mathbf{0}) = \delta(\alpha(\mathsf{I}), s(\mathsf{C}))$. All other cases are proved similarly, see [10] for the full proof. □



$\alpha(\mathsf{O})$
O
$\delta(\alpha(\mathsf{I}), s(\mathsf{C}))$
$\mathsf{I} \overset{\mathsf{C}}{\Rightarrow} \mathsf{O}$
$\alpha(\mathsf{I})$
I

The figure on the left illustrates this result: inner circles (with no borders) are I and O. Shapes with dashed borders are their abstractions through $\alpha$. The shape with a solid border is the abstract state obtained by executing $\delta$ from $\alpha(\mathsf{I})$ and is, in general, an approximation of $\alpha(\mathsf{O})$.

## 4 The Static Analysis Algorithm

```
1:  for all $\mathsf{C} \in \mathbb{C}$ do
2:      $\varphi(\mathsf{C}) := \varnothing$;
3:  end for
4:  ws := $[\langle \mathsf{C}_{init}, \mathsf{INIT} \rangle]$;
5:  $\varphi(\mathsf{C}_{init}) := \{\mathsf{INIT}\}$;
6:  while (!ws.isEmpty()) do
7:      $\langle \mathsf{C}, \sigma^\sharp \rangle$ := ws.pop();
8:      for all $\mathsf{C}_1$ such that suc(C) = ini($\mathsf{C}_1$) do
9:          $\sigma_1^\sharp := \delta(\sigma^\sharp, s(\mathsf{C}))$;
10:         if ($\sigma_1^\sharp \notin \varphi(\mathsf{C}_1)$) then
11:             ws.push($\langle \mathsf{C}_1, \sigma_1^\sharp \rangle$);
12:             $\varphi(\mathsf{C}_1) := \varphi(\mathsf{C}_1) \cup \{\sigma_1^\sharp\}$;
13:         end if
14:     end for
15: end while
```

**Fig. 5.** The ArrayInit algorithm

We describe here a static analysis that determines a subset of those commands that are exactly at the end of a loop performing a complete initialization of an array. This subset is in general strict, since identification of completely initialized arrays is undecidable. The analysis, intraprocedural but aware of interprocedural side-effects, is designed for a specific pair $\langle a, i \rangle$ of variables. Its result lets us compute an under-approximation of the points where $a$ has been initialized through a loop with index variable $i$. We repeat the analysis for each pair $\langle a, i \rangle$, but in practice, a pair $\langle a, i \rangle$ is significant only when $a$ and $i$ occur in actions $a[i] := \mathsf{E}$, which drastically reduces the number of pairs to consider.

Our analysis is formalized by the working set-based fixpoint algorithm in Fig. 5. When the working set (ws) is empty (line 6), a fixpoint is reached. The algorithm starts by applying the automaton in Fig. 3 from $\mathsf{C}_{init}$ and the INIT state (line 4). It reads and *executes* commands in any order allowed by the labels of the program. Consequently, the state of the automaton evolves and the algorithm records it just before executing a command. We use a map $\varphi$ to that purpose, initially empty (line 2) and updated at each command (line 12).

*Example 2.* We show the application of ArrayInit over the Java loop and its corresponding transition system given in Example 1. We assume the working set implemented as a stack. We write I, S, W and A for INIT, START, WRITTEN and

| it. | ws | $C_0$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ | $C_8$ | $C_9$ | $C_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $\langle C_0, I\rangle$ | I | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| 1 | $\langle C_1, S\rangle, \langle C_2, S\rangle$ | I | S | S | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| 2 | $\langle C_2, S\rangle, \langle C_3, S\rangle, \langle C_4, S\rangle$ | I | S | S | S | S | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| 3 | $\langle C_3, S\rangle, \langle C_4, S\rangle, \langle C_{10}, A\rangle$ | I | S | S | S | S | ∅ | ∅ | ∅ | ∅ | ∅ | A |
| 4 | $\langle C_4, S\rangle, \langle C_{10}, A\rangle, \langle C_5, S\rangle$ | I | S | S | S | S | S | ∅ | ∅ | ∅ | ∅ | A |
| 5 | $\langle C_{10}, A\rangle, \langle C_5, S\rangle, \langle C_6, S\rangle$ | I | S | S | S | S | S | S | ∅ | ∅ | ∅ | A |
| 6 | $\langle C_5, S\rangle, \langle C_6, S\rangle$ | I | S | S | S | S | S | S | ∅ | ∅ | ∅ | A |
| 7 | $\langle C_6, S\rangle, \langle C_9, W\rangle$ | I | S | S | S | S | S | S | ∅ | ∅ | W | A |
| 8 | $\langle C_9, W\rangle, \langle C_7, W\rangle$ | I | S | S | S | S | S | S | W | ∅ | W | A |
| 9 | $\langle C_7, W\rangle$ | I | S | S | S | S | S | S | W | ∅ | W | A |
| 10 | $\langle C_8, S\rangle$ | I | S | S | S | S | S | S | W | S | W | A |

**Fig. 6.** Application of ArrayInit on an array initialization loop

ACCEPT. Fig. 6 shows the evolution of ws and $\varphi$ during the iterations. Column $C_i$ stands for the content of $\varphi(C_i)$. Initially, ws $= [\langle C_0, I\rangle]$ with $C_0 = C_{init}$, $\varphi(C_0) = \{I\}$ and $\varphi$ holds the empty set elsewhere. Then we pop $\langle C_0, I\rangle$ from ws and compute $\delta(I, s(C_0)) = \delta(I, \mathbf{0}) = S$. Since $\mathsf{suc}(C_0) = \mathsf{ini}(C_1) = \mathsf{ini}(C_2)$, control passes to $C_1$ and $C_2$. Since $S \notin \varnothing = \varphi(C_1) = \varphi(C_2)$, we push $\langle C_1, S\rangle$ and $\langle C_2, S\rangle$ into ws and update $\varphi$ at $C_1$ and $C_2$. Since ws is not empty, the algorithm continues by popping $\langle C_1, S\rangle$ from ws and computes $\delta(S, s(C_1)) = \delta(S, \mathrm{I}) = S$. Since $\mathsf{suc}(C_1) = \mathsf{ini}(C_3)$ and $S \notin \varnothing = \varphi(C_3)$, we push $\langle C_3, S\rangle$ into ws and update $\varphi$ at $C_3$. The algorithm continues similarly until the working set is empty. $\square$

*Example 3.* Fig. 7 shows another Java fragment, its transition system and iterations of our algorithm. ArrayInit can identify even this unusual and non-trivial array initialization as a complete initialization (continues in Example 4). $\square$

**Proposition 2 (Soundness).** *Let* $C \in \mathbb{C}$. ArrayInit *ends with* $@C \subseteq \cup\varphi(C)$.

*Proof.* We prove a stronger property that entails the thesis: at the end of ArrayInit, for every sequence of application of $\Rightarrow$ of the form $\{\epsilon\} \Rightarrow^{C_1} T_1 \cdots \Rightarrow^{C_n} T_n$,

```
...
1.  i = 0;
2.  while(i < a.length/(i+1)){
3.    a[i] = 7;
4.    if(i > a[i]){
5.      a[i] = i;
      }
6.    i++;
    }
7.  while(i < a.length){
8.    a[i] = 10;
9.    i++;
    }
10. ...
```

| | |
|---|---|
| $C_0$ | $1: i := 0 \to 2;$ |
| $C_1$ | $2: i < a.length \div 2 \to 3;$ |
| $C_2$ | $2: \neg(i < a.length \div 2) \to 7;$ |
| $C_3$ | $3: a[i] := 7 \to 4;$ |
| $C_4$ | $4: (i > a[i]) \to 5;$ |
| $C_5$ | $4: \neg(i > a[i]) \to 6;$ |
| $C_6$ | $5: a[i] := i \to 6;$ |
| $C_7$ | $6: i := i + 1 \to 2;$ |
| $C_8$ | $7: (i < a.length) \to 8;$ |
| $C_9$ | $7: \neg(i < a.length) \to 10;$ |
| $C_{10}$ | $8: a[i] := 10 \to 9;$ |
| $C_{11}$ | $9: i := i + 1 \to 7;$ |
| $C_{12}$ | $10: \cdots$ |

| it. | ws | $C_0$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ | $C_8$ | $C_9$ | $C_{10}$ | $C_{11}$ | $C_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $\langle C_0, I\rangle$ | I | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| 1 | $\langle C_1, S\rangle, \langle C_2, S\rangle$ | I | S | S | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| 2 | $\langle C_2, S\rangle, \langle C_3, S\rangle$ | I | S | S | S | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| 3 | $\langle C_3, S\rangle, \langle C_8, S\rangle, \langle C_9, S\rangle$ | I | S | S | S | ∅ | ∅ | ∅ | ∅ | S | S | ∅ | ∅ | ∅ |
| 4 | $\langle C_8, S\rangle, \langle C_9, S\rangle, \langle C_4, W\rangle, \langle C_5, W\rangle$ | I | S | S | S | S | S | ∅ | ∅ | S | S | ∅ | ∅ | ∅ |
| 5 | $\langle C_9, S\rangle, \langle C_4, W\rangle, \langle C_5, W\rangle, \langle C_{10}, S\rangle$ | I | S | S | S | S | S | ∅ | ∅ | S | S | S | ∅ | ∅ |
| 6 | $\langle C_4, W\rangle, \langle C_5, W\rangle, \langle C_{10}, S\rangle, \langle C_{12}, A\rangle$ | I | S | S | S | S | S | ∅ | ∅ | S | S | S | ∅ | A |
| 7 | $\langle C_5, W\rangle, \langle C_{10}, S\rangle, \langle C_{12}, A\rangle, \langle C_6, W\rangle$ | I | S | S | S | S | S | W | ∅ | S | S | S | ∅ | A |
| 8 | $\langle C_{10}, S\rangle, \langle C_{12}, A\rangle, \langle C_6, W\rangle, \langle C_7, W\rangle$ | I | S | S | S | S | S | W | W | S | S | S | ∅ | A |
| 9 | $\langle C_{12}, A\rangle, \langle C_6, W\rangle, \langle C_7, W\rangle, \langle C_{11}, W\rangle$ | I | S | S | S | S | S | W | W | S | S | S | W | A |
| 10 | $\langle C_6, W\rangle, \langle C_7, W\rangle, \langle C_{11}, W\rangle$ | I | S | S | S | S | S | W | W | S | S | S | W | A |
| 11 | $\langle C_7, W\rangle, \langle C_{11}, W\rangle$ | I | S | S | S | S | S | W | W | S | S | S | W | A |
| 12 | $\langle C_{11}, W\rangle$ | I | S | S | S | S | S | W | W | S | S | S | W | A |

**Fig. 7.** A pair of loops fully initializing an array and their analysis with ArrayInit

with $T_n \neq \varnothing$ and $\mathsf{C}_1 = \mathsf{C}_{init}$, we have $T_n \subseteq \cup\varphi(\mathsf{C}_n)$ and, during the execution of ArrayInit, there is a step where a pair $\langle \mathsf{C}_n, \sigma^\sharp \rangle$, with $\alpha(T_n) \subseteq \sigma^\sharp$, is pushed on the working set $\mathsf{ws}$. The thesis follows from the definition of $@\mathsf{C}$. We prove this property by induction on the length $n \geqslant 1$ of the sequence of applications of $\Rightarrow$.
**Base case:** In this case $n = 1$, hence we have a sequence $\{\epsilon\} \Rightarrow^{\mathsf{C}_1} T_1$ with $\mathsf{C}_1 = \mathsf{C}_{init}$. Line 5 of the algorithm and the fact that it never removes states from the range of $\varphi$ guarantee that at the end of the algorithm $T_1 \subseteq \mathsf{INIT} = \cup\varphi(\mathsf{C}_{init})$. Moreover, $\langle \mathsf{C}_{init}, \mathsf{INIT} \rangle$ is pushed into $\mathsf{ws}$ at line 4, and $\alpha(T_1) \subseteq \mathsf{INIT}$.
**Induction:** Assume that the result holds for some $n \geqslant 1$. We prove it for $n+1$. A sequence of applications of $\Rightarrow$ of length $n+1$ has form $\{\epsilon\} \Rightarrow^{\mathsf{C}_1} T_1 \cdots \Rightarrow^{\mathsf{C}_{n+1}} T_{n+1}$, with $T_{n+1} \neq \varnothing$ and $\mathsf{C}_1 = \mathsf{C}_{init}$. Since $T_{n+1} \neq \varnothing$, we also have $T_n \neq \varnothing$, by definition of $\Rightarrow$. In a sequence $\{\epsilon\} \Rightarrow^{\mathsf{C}_1} T_1 \cdots \Rightarrow^{\mathsf{C}_n} T_n$, by inductive hypothesis we have that, at the end of ArrayInit, $T_n \subseteq \cup\varphi(\mathsf{C}_n)$ and, during the execution of ArrayInit, there is a step at which $\langle \mathsf{C}_n, \sigma^\sharp \rangle$ with $\alpha(T_n) \subseteq \sigma^\sharp$ is pushed into $\mathsf{ws}$. The algorithm terminates only when $\mathsf{ws}$ is empty (line 6), so that pair must have been removed from $\mathsf{ws}$ at some moment, at line 7. $T_{n+1} \neq \varnothing$, so $\mathsf{suc}(\mathsf{C}_n) = \mathsf{ini}(\mathsf{C}_{n+1})$. Hence $\mathsf{C}_{n+1}$ was considered in the loop at line 8, state $\sigma_1^\sharp = \delta(\sigma^\sharp, s(\mathsf{C}_n))$ was computed at line 9 and compared against $\varphi(\mathsf{C}_{n+1})$ at line 10. This might have had two outcomes: **1**) if $\sigma_1^\sharp \notin \varphi(\mathsf{C}_{n+1})$, line 12 adds $\sigma_1^\sharp$ to $\varphi(\mathsf{C}_{n+1})$, where it remains until the end of ArrayInit. No state is ever removed from the range of $\varphi$, so $\sigma_1^\sharp \subseteq \cup\varphi(\mathsf{C}_{n+1})$. By extensivity of $\alpha$ [4], Lemma 1 and monotonicity[1] of $\delta$, we have $T_{n+1} \subseteq \alpha(T_{n+1}) \subseteq \delta(\alpha(T_n), s(\mathsf{C}_n)) \subseteq \delta(\sigma^\sharp, s(\mathsf{C}_n)) = \sigma_1^\sharp \subseteq \cup\varphi(\mathsf{C}_{n+1})$. Line 11 pushed $\langle \mathsf{C}_{n+1}, \sigma_1^\sharp \rangle$ into $\mathsf{ws}$ and from $T_{n+1} \subseteq \sigma_1^\sharp$ (shown above) we have $\alpha(T_{n+1}) \subseteq \alpha(\sigma_1^\sharp) = \sigma_1^\sharp$ since $\sigma_1^\sharp \in \mathbb{A}$. **2**) if $\sigma_1^\sharp \in \varphi(\mathsf{C}_{n+1})$, then it is still there at the end of ArrayInit. As above, we can prove that $T_{n+1} \subseteq \cup\varphi(\mathsf{C}_{n+1})$ and $\alpha(T_{n+1}) \subseteq \sigma_1^\sharp$. □

Hence our algorithm supports a correct array initialization analysis.

**Theorem 1.** *Consider a program $P$, variables $a$ (array) and $i$ (index) and the automaton in Fig. 3 for $a$ and $i$. At the end of the* ArrayInit *algorithm, for every $\mathsf{C} \in \mathbb{C}$ such that $\varphi(\mathsf{C}) = \{\mathsf{ACCEPT}\}$ we have that $\mathsf{ini}(\mathsf{C})$ is a point of $P$ where all elements of $a$ have been initialized by a loop with index $i$.*

*Proof.* By Proposition 2, $@\mathsf{C} \subseteq \cup\varphi(\mathsf{C}) = \mathsf{ACCEPT}$ i.e., every trace $\tau$ leading to $\mathsf{C}$ is in the language of $\Lambda^*\mathbf{0}\mathrm{I}^*((=\mathrm{I}^*)^+ + \mathrm{I}^*)^*(=\mathrm{I}^*)^*\geqslant$. Hence $\tau$ ends with an assignment of 0 to $i$ ($\mathbf{0}$) followed by a repetition of at least an assignment to $a[i]$ ($=$) and then a single increment of $i$ ($+$). At the end of $\tau$, $i$ holds the length of $a$. Since only irrelevant actions are allowed in $\tau$ between those actions, $a$ is definitely completely initialized at the end of the execution represented by $\tau$. □

*Example 4.* From Fig. 6 we know that $\varphi(\mathsf{C}_{10}) = \{\mathsf{ACCEPT}\}$ at the end of the algorithm. By Theorem 1, the array has been completely initialized when program point $9 = \mathsf{ini}(\mathsf{C}_{10})$ is reached. The same holds for program point 10 in Fig. 7. □

[1] It can be proven easily (see [10]) that $\varnothing \subset \mathsf{p} \subseteq \mathsf{q} \Rightarrow \delta(\mathsf{p}, \lambda) \subseteq \delta(\mathsf{q}, \lambda)$.

## 5 Dealing with Implicit Upper Bounds and Side-Effects

Although the analysis of Sec. 3 and 4 determines where an array held in a *local variable* is fully initialized, it has strong limitations. It identifies the comparison between a loop index variable $i$ and the size of an array $a$ in a syntactical, explicit way (Def. 9): it must have the form $\neg(i < a.length)$. This is not the case in Fig. 1 and our analysis would fail there. Moreover, it works only for arrays held in local variables. Again, this is not the case in Fig. 1, where they are stored into *instance variables* i.e., fields. The problem with fields goes beyond the extension of our language of expressions (Fig. 2) with a new expression $x.field$: it actually requires careful attention to side-effects. For instance, method `foo` in Fig. 8 recreates the array at each iteration. At the end of the loop, none of its elements is initialized. A naive extension of our analysis to arrays held in fields might easily turn out to be unsound. We discuss below how we overcome these two limitations.

```
int i = 0; x.f = this;
while (i<this.a.length){
  this.a[i++] = 2;
  foo(x);
}
void foo(x) {
  y = x.f;
  y.a = new T[...];
}
```

**Fig. 8.** Side-effects hinder the full initialization of `this.a`

**Implicit Upper Bounds.** We consider some frequent implicit ways of expressing the upper bound of an array. Often, a variable is used, as `numpoints` in Fig. 1. To prove that it holds the array length, we use the *definite expression aliasing* analysis available in Julia, a traditional available expression analysis [1] for bytecode: bindings from variables to expressions are *generated* by assignments, that also *kill* other bindings referring to the old value of the variables. In Fig. 1, the binding `numpoints = this.mOriginal.length` is generated by the first `new` and never killed later, since `this`, `numpoints` and `this.mOriginal` are not updated. Similarly for the binding `numpoints = this.mRotated.length`. Def. 9 is improved: when $\mathsf{A}$ is $\neg(i < var)$, its abstraction is $\geqslant$ if, there, we have the binding $var = a.length$ ($a$ can be a local variable or a field).

In other cases, the upper bound is a numerical constant. This includes the case when a final static integer field is used (i.e., a symbolic constant) since compilers usually replace it with its numerical value and Julia analyzes the bytecode. Here, we must be sure that the *same* constant is used for the array length *wherever* it is created, also outside the method where the initialization loop occurs. Since there might be more creation points for the objects stored inside the array variable, that condition must hold for all of them. Here, we exploit the *creation point analysis* available in Julia: for each variable at a given program point or field, it over-approximates the set of program points where its content might be created. This is a concretization of class analysis [11]. Def. 9 is improved: when $\mathsf{A}$ is $\neg(i < con)$ and $con$ a numerical constant, its abstraction is $\geqslant$ if all creation points for $a$ (the variable being initialized) have the form `new T[`$con$`]`.

**Side-Effects.** When the array is stored in a field, as in $x.field$, we must strengthen the notion of irrelevant action $\mathsf{A}$ (case I of Def 9): we must require that $\mathsf{mod}(\mathsf{A}) \cap \{x, i\} = \varnothing$ *and* that $\mathsf{A}$ does not modify *field*. The only actions that might modify

| NAME | VENDOR | LOC | TOTAL LOC | ARRAY INITIALIZATION TOTAL | DETECTED | TIME | TOTAL TIME |
|---|---|---|---|---|---|---|---|
| AbdTest | Android Distribution | 489 | 56334 | 1 | 1 | 2.36 | 121.73 |
| AccelerometerPlay | Android Distribution | 306 | 46854 | 1 | 1 | 0.35 | 71.99 |
| CubeWallpaper | Android Distribution | 370 | 25654 | 3 | 3 | 0.12 | 28.51 |
| HoneycombGallery | Android Distribution | 948 | 71501 | 1 | 0 | 1.06 | 157.85 |
| TicTacToe | Android Distribution | 607 | 59040 | 3 | 3 | 0.70 | 102.65 |
| Snake | Android Distribution | 420 | 57075 | 1 | 0 | 0.36 | 117.49 |
| Real3D | Android Distribution | 1228 | 74384 | 2 | 2 | 1.06 | 177.95 |
| ChimeTimer | Moonblink | 4095 | 95781 | 9 | 7 | 0.80 | 383.45 |
| Dazzle | Moonblink | 4376 | 100271 | 4 | 1 | 1.02 | 394.44 |
| OnWatch | Moonblink | 9746 | 113368 | 10 | 6 | 2.91 | 525.15 |
| Tricorder | Moonblink | 10410 | 106100 | 17 | 11 | 1.01 | 467.58 |
| TestAppv2 | Typoweather | 377 | 58365 | 1 | 1 | 0.38 | 102.34 |
| TxWthr | Typoweather | 2024 | 74441 | 7 | 1 | 0.42 | 179.78 |
| JFlex | | 7681 | 40872 | 7 | 6 | 1.35 | 72.46 |
| nti | | 2372 | 13098 | 4 | 4 | 0.09 | 13.55 |
| plume | | 8587 | 43302 | 24 | 21 | 1.19 | 113.07 |

**Fig. 9.** Experiments with our array initialization analysis. LOC is the number of non-blank, non-comment program lines reached and hence analyzed by Julia; TOTAL LOC is the total number of analyzed lines, including `java.*` and `android.*` libraries; TOTAL is the number of reachable loops in those programs (not in libraries) that fully initialize an array, computed by manual check; DETECTED is the number of them that our analysis successfully spot as complete initializations of arrays; in principle, for the most precise static analysis we have DETECTED=TOTAL; TIME is the time in seconds of our array initialization analysis; it is a small fraction of the TOTAL TIME (in seconds) of the nullness analysis of Julia: the latter includes parsing of the class files, preprocessing, aliasing, sharing, creation points, expression aliasing and side-effects analyses.

*field* are explicit assignments to *y.field*, for any *y*, and calls to non-pure methods (neither of them is in Fig. 2, but naturally a real language includes both). Here, we use the *side-effects analysis* provided by Julia: for each method call, it over-approximates the set of fields modified during the execution of the callee(s) (and of the methods that the callees invoke, recursively).

## 6 Experiments

We implemented our analysis in the Julia tool: `http://www.juliasoft.com`. Experiments in Figure 9 were performed on a quad-core Intel Xeon 64 bits machine at 2.66GHz, with 8GB of RAM, Linux 2.6.27 and Sun jdk 1.6. We analyzed the lexical analyzers generator `JFlex` (`http://jflex.de`), the `plume` library by Michael D. Ernst (`http://code.google.com/p/plume-lib`), the non-termination analyzer `nti` by Étienne Payet, programs from Google's Android 3.0 distribution and Android applications from public repositories (`http://code.google.com/p/moonblink` and `http://code.google.com/p/typoweather`).

| NAME | NULLNESS ARRINIT | ~~ARRINIT~~ |
|---|---|---|
| AccelerometerPlay | 3 | 6 |
| ChimeTimer | 33 | 36 |
| CubeWallpaper | 0 | 3 |
| TicTacToe | 0 | 2 |
| JFlex | 57 | 65 |
| OnWatch | 82 | 85 |
| Real3D | 19 | 19 |
| Tricorder | 107 | 121 |
| TxWthr | 48 | 49 |
| nti | 15 | 15 |
| plume | 57 | 59 |

While Figure 9 shows that our array initialization analysis is fast and precise, the table on the left shows that it is useful to a client analysis. In particular, it considers those programs from Fig. 9 that contain at least a loop initializing an array of reference type, since otherwise the array initialization analysis would be irrelevant for nullness analysis. It reports the number of null-pointer warnings with (ArrInit) and without (~~ArrInit~~) our array initialization analysis. In the former case, the precision of the nullness analysis is improved by 8.48% on average on these programs and its cost is only 0.47% higher (compare TIME and TOTAL TIME in Fig. 9).

When Julia fails to spot complete array initialization, the problem is related to weaknesses in the supporting analyses rather than to our array initialization analysis. For instance, there is a complete array initialization in `HoneycombGallery` that Julia fails to spot (Fig. 9). Here it is:

```
String[] items = new String[cat.getEntryCount()];
for (int i = 0; i < cat.getEntryCount(); i++)
  items[i] = cat.getEntry(i).getName();
```

The loop upper bound is `cat.getEntryCount()`, which does not fall in the cases considered in Sec. 5. The use of a method call as loop upper bound is problematic since the definite expression aliasing analysis must be able to prove that the value of `cat.getEntryCount()` is constant between the creation of the array and the check of the loop upper bound, also when the loop body has side-effects, as here.

## 7 Conclusion

Our new automaton-based abstract interpretation detects fully initialized arrays held in local variables or fields. The implementation is efficient, precise and effective to support a client nullness analysis. We observe that our array initialization analysis is not tailored to nullness, but can support any other client analysis.

## References

1. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques and Tools. Addison-Wesley (1986)
2. Bradley, A.R., Manna, Z., Sipma, H.B.: What's Decidable About Arrays? In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 427–442. Springer, Heidelberg (2005)
3. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of the 4th Symp. on Principles of Programming Languages, pp. 238–252. ACM (1977)
4. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proc. of the 6th Symposium on Principles of Programming Languages (POPL 1979), pp. 269–282. ACM (1979)
5. Cousot, P., Cousot, R.: Systematic design of program transformation frameworks by abstract interpretation. In: Proc. of the 29th Symposium on Principles of Programming Languages (POPL 2002), pp. 178–190. ACM (2002)

6. Cousot, P., Cousot, R., Logozzo, F.: A parametric segmentation functor for fully automatic and scalable array content analysis. In: Proc. of the 38th Symposium on Principles of Programming Languages, New York, USA, pp. 105–118 (2011)
7. Flanagan, C., Qadeer, S.: Predicate abstraction for software verification. In: Proc. of the 29th Symposium on Principles of Programming Languages (POPL 2002), pp. 191–202. ACM (2002)
8. Graf, S., Saïdi, H.: Construction of Abstract State Graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
9. Habermehl, P., Iosif, R., Vojnar, T.: What Else Is Decidable about Integer Arrays? In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 474–489. Springer, Heidelberg (2008)
10. Nikolić, D., Spoto, F.: Automaton-based array initialization analysis, http://profs.sci.univr.it/~nikolic/download/LATA2012/LATA2012Ext.pdf
11. Palsberg, J., Schwartzbach, M.I.: Object-oriented type inference. In: Proceedings of Object-Oriented Programming, Systems, Languages & Applications (OOPSLA 1991). ACM SIGPLAN Notices, vol. 26(11), pp. 146–161. ACM (1991)
12. Spoto, F.: Precise null-pointer analysis. Software and Systems Modeling 10(2), 219–252 (2011)

# Dynamics of Circuits and Intersecting Circuits

Mathilde Noual[1,2]

[1] Université de Lyon, ÉNS-Lyon, LIP, CNRS UMR5668, Lyon, France
[2] IXXI, Institut Rhône-alpin des systèmes complexes, Lyon, France

**Abstract.** This paper presents a combinatorial study to characterise the dynamics of intersecting Boolean automata circuits and more specifically that of *double Boolean automata circuits*. Explicit formulae are given to count the number of periodic configurations and attractors of these networks and a conjecture proposes a comparison between the number of attractors of isolated circuits and that of double circuits. The aim of this study is to give intuition on the way circuits interact and how a circuits intersection modifies the "degrees of freedom" of the overall network.

**Keywords:** Positive and Negative Circuits, Boolean Automata Network, Regulation Network, Dynamical Behaviour, Attractor.

## Introduction

Since McCulloch and Pitts [9] proposed *threshold Boolean automata networks* to represent formally neural networks and, later, Kauffman [8] and Thomas [17] introduced the first Boolean models of genetic regulation networks, automata networks have been widely studied (*e.g.* [3,5,6,7,12,14]). One of the main motivations in this context has been to better understand the emergent dynamical behaviours that networks of interacting elements display that cannot be explained or predicted by a simple analysis of the local interactions they involve. In particular, Thomas highlighted the importance of specific patterns in the interaction structure of a network and focused especially on circuits (*i.e.* oriented cycles of consecutive interactions) [18].

Circuits are instances of the most basic structural patterns that have a significant impact on the dynamics of a network. They are responsible for the diversity in its possible limit behaviours. And indeed, in a network whose structure is acyclic, the "information" runs linearly from the source automata that have constant states towards the sink automata whose states influence that of no other. In [4], we described the dynamics of isolated Boolean automata circuits, *i.e.* , Boolean networks structured as simple circuits. Now, using this as a basis, we aim at to going further, towards the understanding of the behaviour of arbitrary networks that possibly involve several interacting circuits. To do so, we propose to start by studying the dynamics of intersecting circuits.

Of course, after having studied exhaustively the dynamics of isolated circuits, the approach that consists in doing the same for two intersecting circuits, then three, then ... is certainly limited very early by a complexity problem so that it cannot hope to reach the stage where it provides a complete description of the dynamics of networks whose interaction structures are arbitrary strongly connected components. However, because the first step turned out to provide some significant intuition on the way circuits interact through their intersections, we present it here.

In Section 1, we start by giving some definitions and preliminary results. Next, in Section 2, we give and prove our main result on the dynamics of circuits (summing up results from [4]) and double-circuits. Finally, in Section 3, we propose to compare, in terms of combinatorics, the dynamics of these two types of networks.

## 1 Definitions and Preliminary Results

### Boolean Automata Networks

A *Boolean automata network* is a couple $\mathcal{N} = (G, \mathcal{F})$ where $G = (V, A)$ is a digraph called the *interaction structure* of $\mathcal{N}$ and $\mathcal{F}$ is its set of *local transition functions*: $\mathcal{F} = \{f_i : \{0,1\}^n \to \{0,1\} \mid i \in V\}$. Nodes of $G$ are considered as the automata of the network. They are supposed to be numbered from 0 to $n-1$ where $|V| = n$ is the network size. Boolean vectors $x \in \{0,1\}^n$ are called *configurations* of $\mathcal{N}$. Their coefficients $x_i \in \{0,1\}$ represent automata states. Informally, the local transition function $f_i$ of automaton $i$ outputs the new state that $i$ will take if it is updated, given as input the current state of the network. It satisfies the following:

$$(j,i) \in A \Leftrightarrow \exists x \in \{0,1\}^n,\ f_i(x) \neq f_i(\overline{x}^j) \tag{1}$$

where $\overline{x}^j$ is the configuration defined by $\forall k \neq j,\ \overline{x}^j_k = x_k$ and $\overline{x}^j_j = \neg x_j$. Here, we add the requirement that local transition functions be locally monotone so that each arc of the interaction structure can be *signed*, *i.e.* , $\forall (j,i) \in A$, only two cases are possible:

$$\begin{aligned} sign(j,i) = + &\ \Leftrightarrow\ \forall x \in \{0,1\}^n,\ x_j = 0,\ f_i(x) \leq f_i(\overline{x}^j), \\ sign(j,i) = - &\ \Leftrightarrow\ \forall x \in \{0,1\}^n,\ x_j = 0,\ f_i(x) \geq f_i(\overline{x}^j). \end{aligned}$$

For the sake of simplicity, in the sequel, the arity of local transition functions is restricted so that they become functions of the following form: $f_i : \{0,1\}^{deg^-(i)} \to \{0,1\}$ (where $deg^-(i)$ denotes the in-degree of $i$ in $G$). Moreover, we consider only the parallel updating mode so we define the *global transition function* of the network as follows:

$$F : \begin{cases} \{0,1\}^n \to \{0,1\}^n \\ \quad x \quad\ \mapsto (f_0(x), f_1(x), \ldots, f_{n-1}(x)) \end{cases}.$$

For any network configuration $x \in \{0,1\}^n$, we prefer the notation:

$$x = x(0) \text{ and } x(t) = F^t(x)$$

where $F^t$ is the $t^{\text{th}}$ iterate of $F$ so that if $x$ represents the initial network configuration, then, $x(t)$ represents its configuration at time step $t$.

### Circuits and Double-circuits

A *circuit* of size $n$ is a digraph denoted by $C_n = (V, A)$ whose set of nodes is considered to be $V = \mathbb{N}/n\mathbb{N}$ so that $i + j$ denotes node $i + j$ mod $n$, and whose set of arcs equals $A = \{(i, i+1) \mid i \in V\}$. A *double circuit* $D_{\ell r}$ (see Fig. 1 left) is a digraph of order $n = \ell + r - 1$ consisting of two sub-graphs called the *side-circuits* of $D_{\ell r}$. The *left-* (resp. *right-*) *circuit* is a sub-graph isomorphic to $C_\ell$ (resp. to $C_r$). Its size $\ell$ (resp. $r$) is called the *left-* (resp. *right-*) *size* of $D_{\ell r}$. Nodes of the left-circuit are numbered from 0 to $\ell - 1$. The right-circuit contains those that are numbered from $\ell$ to $\ell + r - 1$ as well as node 0 which belongs to both side-circuits and is the only node with in- and out-degree 2 (rather than 1).

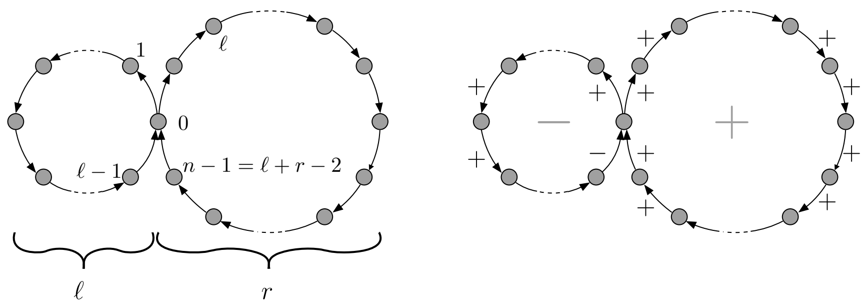


**Fig. 1.** Left: A double circuit $D_{\ell r}$. Right: The signed interaction structure of a *canonical* negative-positive DBAC $\mathcal{D}^{-+}_{\ell r}$ . All arcs are positive except for the arc $(\ell-1, 0)$, *i.e.* $f_0^L = neg$, $f_0^R = id$ and $\forall i \neq 0, f_i = id$.

## Boolean Automata Circuits and Double Boolean Automata Circuits

A *Boolean automata circuit* (or BAC) is a network $(C_n, \mathcal{F})$ whose interaction structure is a circuit. By (1), in such a network where all local transition functions have arity 1, it holds that $\forall i \in V,\ f_i \in \{id : a \mapsto a, neg : a \mapsto \neg a\}$. BACs with an even (resp. odd) number of negative arcs are called *positive* (resp. *negative*) *circuits* and are denoted by $\mathcal{C}_n^+$ (resp. $\mathcal{C}_n^-$). A *double Boolean automata circuit*, or DBAC for short, is a network $(D_{\ell r}, \mathcal{F})$ whose interaction structure is a double circuit $D_{\ell r}$. By (1) and by the local monotony of local transition functions, in a DBAC, $f_0$ can be written:

$$f_0(x_{\ell-1}, x_{n-1}) = f_0^L(x_{\ell-1}) \diamond f_0^R(x_{n-1}) \quad \text{where} \quad \diamond \in \{\wedge, \vee\} \tag{2a}$$

and the following is true:

$$\forall i \in V, i \neq 0, \quad f_i, f_0^L, f_0^R \in \{id, neg\}. \tag{2b}$$

A side-circuit with an even (resp. an odd) number of negative arcs is called a *positive* (resp. a *negative*) *(side-)circuit.* We use $\mathcal{D}^{ss'}_{\ell r}$ to denote a DBAC with left-size $\ell$, right-size $r$, left-sign $s$ and right-sign $s'$ ($s, s' \in \{-, +\}$). Also, for any network configuration $x = (x_0, \ldots, x_{n-1}) \in \{0,1\}^n$, we write $x^L = (x_0, \ldots, x_{\ell-1})$ and $x^R = (x_0, x_\ell, \ldots, x_{n-1})$ to denote, respectively, the restrictions of $x$ to the configurations of the left- and right-circuits.

## Network Dynamics

The dynamics of a network $\mathcal{N} = (G, \mathcal{F})$ are described by a *transition graph* $\mathcal{T} = (\{0,1\}^n, T)$ which equals the graph of the global transition function of $\mathcal{N}$:

$$T = \{\ (x(t), x(t+1))\ \} \subseteq \{0,1\}^n \times \{0,1\}^n.$$

Configurations that belong to circuits in $\mathcal{T}$ are called *periodic configurations*. For a periodic configuration $x = x(t)$, any integer $p$ satisfying $x(t) = x(t+p)$ is called a *period* of $x$. An *attractor* of period $p \in \mathbb{N}$, or *p-attractor*, is a set of $p$ configurations belonging to the orbit of a configuration that has *minimal* period $p$. Attractors of period 1 are called *fix points*.

## Dirichlet Convolutions

Let $1\!\!1$ be the function $1\!\!1 : n \in \mathbb{N} \mapsto 1$ and let $\star$ denote the *Dirichlet convolution* [2], that is, the binary operator such that for any two arithmetic functions $f$ and $g$:

$$f \star g : n \in \mathbb{N} \mapsto \textstyle\sum_{p|n} f(p) \cdot g\big(n/p\big)\,.$$

We recall that the set of arithmetic functions with point-wise addition and Dirichlet convolution is a commutative ring. The multiplicative identity of this ring is the function $\delta : \mathbb{N} \mapsto \mathbb{N}$ defined by $\delta(1) = 1$ and $\forall n > 1,\ \delta(n) = 0$. Let us also recall that the inverse of function $1\!\!1$ for the Dirichlet convolution is the Möbius function (see [2], for instance) $\mu : \mathbb{N} \mapsto \{-1, 0, 1\}$:

$$\mu : \begin{cases} n \mapsto 0 & \text{if } n \text{ is not square-free} \\ n \mapsto 1 & \text{if } n > 0 \text{ has an even number of prime factors} \\ n \mapsto -1 & \text{if } n > 0 \text{ has an odd number of prime factors.} \end{cases}$$

The importance of this function here lies in the *Möbius inversion formula* that it satisfies (deriving from $1\!\!1 \star \mu = \delta$) for all arithmetic functions $f$ and $g$:

$$\begin{gathered} g = f \star 1 \ \Rightarrow\ f = g \star \mu \\ \textit{i.e. } \forall n \in \mathbb{N},\ g(n) = \textstyle\sum_{p|n} f(p) \ \Rightarrow\ f(n) = \textstyle\sum_{p|n} g(p) \cdot \mu\big(n/p\big). \end{gathered}$$

Another notable property is the resulting relation between the Möbius function and the Euler totient $\varphi$: since $\varphi$ satisfies $\forall n \in \mathbb{N},\ n = \varphi \star 1\!\!1(n)$, it holds that $\varphi = \mu \star id$, where $id : n \in \mathbb{N} \mapsto n$.

## Useful Quantities

We call *order* of a network $\mathcal{N}$ the least common multiple of all attractor periods of $\mathcal{N}$. We write $\omega(\mathcal{N}) = \mathrm{lcm}\{p \in \mathbb{N} \mid p \text{ is an attractor period of } \mathcal{N}\}$ and can show that $\omega(\mathcal{N}) = \min\{p \in \mathbb{N} \mid \forall \text{ periodic } x \in \{0,1\}^n,\ F^p(x) = x\}$. In combinatorial terms, the asymptotic dynamics of a network can then be described by the four quantities (see [10]) listed below for a divisor $p$ of $\omega(\mathcal{N})$:

- The number $\mathtt{X}(p)$ of configurations of period $p$ satisfying (see (3b)):

$$\mathtt{X} = \mathtt{Y} \star \mathbb{1}; \tag{3a}$$

- The number $\mathtt{Y}(p)$ of configurations of minimal period $p$ satisfying:

$$\mathtt{Y} = \mathtt{X} \star \mu; \tag{3b}$$

- The number $\mathtt{A}(p) = \frac{\mathtt{Y}(p)}{p}$ of $p$-attractors satisfying:

$$\mathtt{A} = inv \cdot (\mathtt{X} \star \mu); \tag{3c}$$

- The number $\mathtt{T}(p)$ of attractors of period a divisor of $p$, satisfying:

$$\mathtt{T} = \mathtt{A} \star \mathbb{1} = inv \cdot (\mathtt{X} \star \varphi). \tag{3d}$$

where $inv : n \in \mathbb{N} \mapsto 1/n$ and where the last equality above holds because completely multiplicative functions such as $inv$ distribute over $\star$. For an integer $p$ that does not divide $\omega(\mathcal{N})$, $\mathtt{X}(p) = \mathtt{Y}(p) = \mathtt{A}(p) = \mathtt{T}(p) = 0$.

*Remark 1.* As a result of the existence of the relations (3a) to (3d), to determine any of the three quantities $\mathtt{Y}$, $\mathtt{A}$ and $\mathtt{T}$ relative to a given network $\mathcal{N}$, it suffices to determine the quantity $\mathtt{X}$.

## Canonical Boolean Automata Circuits and DBACs

**Lemma 2 ([4]).** *For a* BAC $\mathcal{N} = \mathcal{C}_n^s$, *the four quantities* $\mathtt{X}$, $\mathtt{Y}$, $\mathtt{A}$, $\mathtt{T}$ *depend only on the size* $n \in \mathbb{N}$ *and sign* $s \in \{-,+\}$ *of the circuit and for a* DBAC $\mathcal{N} = \mathcal{D}_{\ell r}^{ss'}$, *they depend only on the sizes* $\ell, r \in \mathbb{N}$ *and signs* $s, s' \in \{-,+\}$ *of the side-circuits.*

By Lemma 2 which was proven in [4], $\mathtt{X}$, $\mathtt{Y}$, $\mathtt{A}$ and $\mathtt{T}$ thus do not depend on the number and position of the negative arcs in a (side-)circuit. We can therefore concentrate on *canonical* circuits and *canonical* DBACs with minimal numbers of negative arcs (0 or 1 for circuits, $0, 1$ or 2 for DBACs). By Lemma 2, $\mathtt{X}$, $\mathtt{Y}$, $\mathtt{A}$ and $\mathtt{T}$ do not depend either on whether $\diamond = \vee$ or $\diamond = \wedge$ in the definition of the local transition function $f_0$ (see (2a)). For canonical DBACs, arbitrarily, we choose $\diamond = \vee$. In addition, we set $\forall i \neq 0,\ f_i = id$ and $f_0^L = neg$ (resp. $f_0^R = neg$) if the left-circuit (resp. right-circuit) is meant to be negative, $f_0^L = id$ (resp. $f_0^R = id$) if it is meant to be positive. This way, the only possible negative arcs in canonical DBACs are arcs $(\ell - 1, 0)$ and $(n - 1, 0)$.

## Relationships between Automata States in a DBAC

Let us note that in a canonical DBAC, because $\forall i \neq 0,\ f_i = id$, the state of any node $i \neq 0$ at any time can be expressed simply as the state of node 0 at some previous time: $\forall x(t) \in \{0,1\}^n,\ \forall i \in V$,

$$\begin{cases} x_i(t+i) = x_0(t) & \text{if } i < \ell \text{ belongs to the left-circuit and} \\ x_i(t+i-\ell) = x_0(t) & \text{if } i \geq \ell \text{ belongs to the right-circuit.} \end{cases} \tag{4}$$

Also, if $x(t)$ has period $p$, then it can be shown by induction that:

$$x_i(t) = \begin{cases} x_{i \bmod p}(t) & \text{if } i < \ell \text{ and} \\ x_{(i-\ell) \bmod p}(t) & \text{if } i \geq \ell. \end{cases} \tag{5}$$

Further, if $x(t)$ has period $p$ and if $t \geq p$, $\ell = k_\ell \cdot p + d_\ell \equiv d_\ell \bmod p$ and $r = k_r \cdot p + d_r \equiv d_r \bmod p$, then, according to (4) and (5), the circular binary word $w$ defined by:

$$w \in \{0,1\}^p,\ \forall i \in \mathbb{N}/p\mathbb{N},\ w_i = x_0(t+i) \tag{6a}$$

is such that:

$$x(t)^L = w^{k_\ell} w[0,\dots,d_\ell - 1] \ \text{ and } \ x(t)^R = w^{k_r} w[0,\dots,d_r - 1] \tag{6b}$$

where $w[m,\dots,m'] = w_m w_{m+1} \dots w_{m'}$, $\forall m \leq m' < |w|$. As a consequence, to describe the asymptotic dynamics of a canonical DBAC, it suffices to describe the behaviour of the intersection node 0, concentrating on the word $w$ associated to each periodic configuration.

## 2 Main Result

The main result, Theorem 3 below, involves both the *Lucas sequence* $(\mathtt{L}(n))_{n\in\mathbb{N}}$ [11,13] (sequence A204 of the OEIS [16]) defined by $\mathtt{L}(1) = 1$, $\mathtt{L}(2) = 3$ and $\mathtt{L}(n) = \mathtt{L}(n-1)+\mathtt{L}(n-2)$, $\forall n > 2$, and the *Perrin sequence* $(\mathtt{P}(n))_{n\in\mathbb{N}}$ [1] (sequence A1608 of the OEIS [16]) defined by $\mathtt{P}(0) = 3$, $\mathtt{P}(1) = 0$, $\mathtt{P}(2) = 2$ and $\mathtt{P}(n) = \mathtt{P}(n-2) + \mathtt{P}(n-3)$, $\forall n > 2$.

**Theorem 3.** *Forany network $\mathcal{N}$ which is either a Boolean automata circuit (*BAC*) or a double Boolean automata circuit (*DBAC*), the order $\omega(\mathcal{N})$ of $\mathcal{N}$ and the number of its configurations of period a divisor $p$ of $\omega(\mathcal{N})$ are given in the table below:*

| *Network $\mathcal{N}$* | *Order $\omega(\mathcal{N})$* | *Number of configurations of period $p$, $p\|\omega(\mathcal{N})$* |
|---|---|---|
| *Positive* BAC $\mathcal{C}_n^+$ | $n$ | $\mathtt{X}^+(p) \ = 2^p$ |
| *Negative* BAC $\mathcal{C}_n^-$ | $2n$ | $\mathtt{X}^-(p) \ = 2^p$ |
| *Positive-positive* DBAC $\mathcal{D}_{\ell r}^{++}$ | $\gcd(\ell, r)$ | $\mathtt{X}^{++}(p) = 2^p$ |
| *Negative-positive* DBAC $\mathcal{D}_{\ell r}^{-+}$ | $\omega(\mathcal{D}_{\ell r}^{-+}) \mid r$ | $\mathtt{X}_\ell^{-+}(p) = \mathtt{L}(\frac{p}{\Delta_p})^{\Delta_p}$ |
| *Negative-negative* DBAC $\mathcal{D}_{\ell r}^{--}$ | $\omega(\mathcal{D}_{\ell r}^{--}) \mid \ell{+}r$ | $\mathtt{X}_\Delta^{--}(p) = \mathtt{P}(\frac{p}{\Delta_p})^{\Delta_p}$ |

*where $\Delta = \gcd(\ell, r)$, and $\Delta_p = \gcd(p, \ell)$ (resp. $\Delta_p = \gcd(p, \Delta)$) in the case of a negative-positive* DBAC *(resp. in that of a negative-negative* DBAC*).*

*For any network $\mathcal{N}$ which is either a* BAC *or a* DBAC*, this and Remark 1 on Page 437 directly yield explicit formulae for the number of configurations of minimal period $p$, the number of $p$-attractors and the total number of attractors of $\mathcal{N}$.*

Thus, for instance, a negative-positive DBAC $\mathcal{D}^{-+}_{3,6}$ has $1 = \mathtt{Q}^{-+}_3(1)$ ($\mathtt{Q} \in \{\mathtt{X}, \mathtt{Y}, \mathtt{A}, \mathtt{T}\}$) fix point. Also, it has $\mathtt{X}^{-+}_3(2) = 3$ (resp. $\mathtt{Y}^{-+}_3(2) = 2$) configurations of (resp. minimal) period 2, $\mathtt{X}^{-+}_3(3) = 1$ (resp. $\mathtt{Y}^{-+}_3(3) = 0$) of (resp. minimal) period 3 and $\mathtt{X}^{-+}_3(6) = 27$ (resp. $\mathtt{Y}^{-+}_3(6) = 24$) of (resp. minimal) period 6. Therefore, it has $\mathtt{A}_3(2) = 1$ attractor of period 2, $\mathtt{A}_3(6) = 4$ of period 6 and $\mathtt{T}_3(6) = 6$ attractors in total.

Results in Theorem 3 that concern isolated Boolean automata circuits $\mathcal{C}^s_n$, $s \in \{-,+\}$ and positive-positive DBACs $\mathcal{D}^{++}_{\ell r}$ as well as network orders are proven in [4]. As a consequence, here, we focus on DBACs that have at least one negative side-circuit. The following four points are remarks that derive from Theorem 3 and which are also proven in [4]. In Points 1 and 3, a DBAC $\mathcal{D}^{ss'}_{\ell r}$ is said to "behave" as a BAC $\mathcal{C}^{s''}_n$ with same order $\omega = \omega(\mathcal{D}^{ss'}_{\ell r}) = \omega(\mathcal{C}^{s''}_n)$. By this it is meant that the sub-transition graph of $\mathcal{D}^{ss'}_{\ell r}$ induced by its periodic configurations is isomorphic to the transition graph of $\mathcal{C}^{s''}_n$ and thus $\mathtt{Q}^{ss'}(p) = \mathtt{Q}^{s''}(p)$, $\forall p | \omega$ and $\forall \mathtt{Q} \in \{\mathtt{X}, \mathtt{Y}, \mathtt{A}, \mathtt{T}\}$.

1. A positive-positive DBAC $\mathcal{D}^{++}_{\ell r}$ behaves as a positive BAC $\mathcal{C}^{+}_{\Delta}$, $\Delta = \gcd(\ell, r)$.
2. Strictly positive attractor periods of a DBAC $\mathcal{D}^{ss'}_{\ell r}$ divide the sizes of the underlying positive circuits (including the size $\ell + r$ of the non-elementary encompassing positive circuit when $s = s'$) and do not divide the sizes of the underlying negative circuits, if there are any.
3. A DBAC $\mathcal{D}^{ss}_{nn}$ behaves as a Boolean automata circuit $\mathcal{C}^s_n$.
4. A DBAC has as many fix points as it has positive side-circuits.

## Negative-Positive DBACs

In this section, without loss of generality as justified earlier, we consider only canonical DBACs $\mathcal{D}^{-+}_{\ell r}$ (see Fig. 1 right and caption). According to Point 2 above, these networks have exactly one fix point and all their other attractors have periods that divide $r$ without dividing $\ell$.

**Lemma 4 (Characterisation of periodic configurations).** *Let $p, \ell, r, k, d$ be integers such that $p | r$ and $\ell = kp + d \equiv d$ mod $p$. A configuration $x(0)$ of $\mathcal{D}^{-+}_{\ell r}$ has period $p$ if and only if there exists a circular word $w \in \{0,1\}^p$ satisfying (6b) that does not contain a sub-sequence $0u0$, $u \in \{0,1\}^{d-1}$.*

*Thus, the number of configurations of period $p$ of $\mathcal{D}^{-+}_{\ell r}$ equals the number of binary circular words of length $p$ without a sub-sequence $0u0$, $u \in \{0,1\}^{d-1}$.*

*Proof.* The second part is a direct consequence of the first. Let $x(0)$ be a configuration of $\mathcal{D}^{-+}_{\ell r}$ with period $= r/qp$ and let $w \in \{0,1\}^p$ be the word associated to $x(0)$ defined in (6). Then:

$$\begin{aligned} x_0(t) &= x_0(t + k \cdot p) \\ &= \neg x_{\ell-1}(t + k \cdot p - 1) \vee x_{n-1}(t + k \cdot p - 1) \\ &= \neg x_{\ell - k \cdot p}(t) \vee x_0(t + (k-q) \cdot p) \\ &= \neg x_d(t) \vee x_0(t), \\ = w_t &= \neg w_{t-d \bmod p} \vee w_t \end{aligned}$$

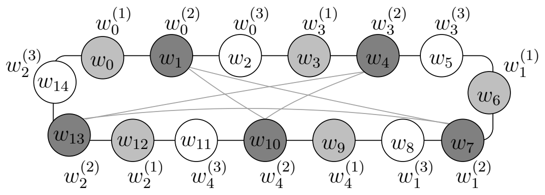


**Fig. 2.** The circular word $w = w_0 \dots w_{p-1} = x_0(t) \dots x_0(t+p-1)$ that characterises a configuration of period $p$ of a DBAC $\mathcal{D}^{-+}_{\ell r}$ (see (6)). Here, $p = 15$, $d = \ell \bmod p = 6$ so that $w$ is is an interleave of $\Delta_p = \gcd(d, p) = 3$ words $w^{(1)}$, $w^{(2)}$ and $w^{(3)}$ of size $p/\Delta_p = 5$, corresponding respectively to nodes in light grey, dark grey and white.

where $d$ and $k$ are as in Lemma 4. Thus, if $w$ contains a sub-sequence $0u0$, $u \in \{0,1\}^{d-1}$, then there is an integer $t$ such that $w_t = w_{t-d \bmod p}$ and $0 = \neg 0 \vee 0 = 1$ which is impossible. The converse of Lemma 4 can be proven easily. □

**Lemma 5.** *The number of configurations of period $p$ of $\mathcal{D}^{-+}_{\ell r}$, where $p|r$, equals $\mathtt{X}^{-+}_{\ell}(p) = \mathtt{L}(\frac{p}{\Delta_p})^{\Delta_p}$, $\Delta_p = \gcd(p, \ell)$, and as a consequence, $\mathtt{Q}^{-+}_{\ell}(p) = \mathtt{Q}^{-+}_{\ell \bmod r}(p)$ for any of the four quantities $\mathtt{Q} = \mathtt{X}, \mathtt{Y}, \mathtt{A}$ and $\mathtt{T}$, introduced in (3).*

*Proof.* The Lucas sequence counts the number of circular words of size $n$ that do not contain the sub-sequence 00. By Lemma 4, $\mathtt{X}^{-+}_{\ell}(p)$ equals the number of circular words $w \in \{0,1\}^p$ without a sub-sequence $0u0$, $u \in \{0,1\}^{d-1}$, where $d = \ell \bmod p$. Any one of these words $w$ can be written as an interleaving of a certain number $m$ of circular words $w^{(1)}, w^{(2)}, \dots, w^{(m)}$ of size $s = p/m$ without the sub-sequence 00 (see Fig. 2). The size $s$ of a word $w^{(j)}$ satisfies $s \times d = q \times p$ for a certain integer $q$ that is minimal, *i.e.* $sd = \operatorname{lcm}(d, p) = dp/\gcd(d, p)$ so that $s = p/\Delta_p$ and, consequently, $m = \Delta_p$ and $\mathtt{X}^{-+}_{\ell}(p) = \mathtt{L}(s)^m = \mathtt{L}(\frac{p}{\Delta_p})^{\Delta_p}$. From this and from (3) follows the rest of Lemma 5. □

## Negative-Negative DBACs

In a canonical negative-negative DBAC, all arcs are positive except the negative arcs $(\ell-1, 0)$ and $(n-1, 0)$, *i.e.* $\forall i \neq 0,\ f_i = id$ and $f^L_0 = f^R_0 = neg$. By Point 2 on Page 439, all attractors periods divide $N = \ell + r$ without dividing $\Delta = \gcd(\ell, r)$. We omit the proof of the following lemma which resembles that of Lemma 4.

**Lemma 6 (Characterisation of periodic configurations).** *Let $p, \ell, r, k_\ell, k_r, d_\ell, d_r$ be integers such that $p|\ell + r$, $\ell = k_\ell \cdot p + d_\ell \equiv d_\ell \bmod p$ and $r = k_r \cdot p + d_r \equiv d_r < d_\ell \bmod p$.*

*A configuration $x(0)$ of a DBAC $\mathcal{D}^{--}_{\ell r}$ has period $p$ if and only if there exists a word $w \in \{0,1\}^p$ satisfying (6b) that does not contain any sub-sequence $0u0$ nor $1u1u'1$, $u, u' \in \{0,1\}^{d_r-1}$.*

*Thus, the number of configurations of period $p$ of $\mathcal{D}^{--}_{\ell r}$ equals the number of binary circular words of length $p$ without a sub-sequence $0u0$ nor $1u1u'1$, $u, u' \in \{0,1\}^{d_r-1}$.*

**Lemma 7.** *For $n > 0$, $\mathtt{P}(n)$ counts the number of circular words of size $n$ without the sub-sequences* 00 *and* 111*.*

*Proof.* The statement of Lemma 7 is true for $1 \leq n \leq 3$. Let $E_n \subseteq \{0,1\}^n$ be the set of circular words of length $n$ without the factors 00 and 111. If $n \geq 3$, then any word $w \in E_n$ can be written $w = u0v$ where $u$ is a word with no 0, possibly the empty word $\varepsilon$. Further, for $n > 3$, only two disjoint cases are possible: either $v = 10v'$ or $v = 110v'$, for some binary word $v'$. From this follows that $E_n = \{u010v' \,|\, u \in \{\varepsilon, 1, 11\},\, u0v' \in E_{n-2}\} \uplus \{u0110v' \,|\, u \in \{\varepsilon, 1, 11\},\, u0v' \in E_{n-3}\}$ and then $|E_n| = |E_{n-2}| + |E_{n-3}|$. □

Lemma 7 explains why the Perrin sequence is involved in Theorem 3. Finally, combining Lemmas 6 and 7, we obtain Lemma 8 below as we did for Lemma 5. This ends the proof of the part of Theorem 3 that concerns negative-negative DBACs.

**Lemma 8.** *The number of configurations of period $p$ of a negative-negative* DBAC $\mathcal{D}^{--}_{\ell r}$*, where $p | \ell + r$, equals $\mathtt{X}^{--}_{\Delta}(p) = \mathtt{P}(\frac{p}{\Delta_p})^{\Delta_p}$, where $\Delta = \gcd(\ell, r)$ and $\Delta_p = \gcd(p, \Delta)$.*

## 3 Comparisons and Bounds

Since the aim of this paper is to gain intuition on the way that circuits interact, our next step is to compare the dynamics of isolated circuits with that of DBACs. In these lines, a first result is necessary to bound the number of periodic configurations of DBACs:

**Lemma 9.** *The numbers of configurations of period $p$ of $\mathcal{D}^{-+}_{\ell r}$ and of $\mathcal{D}^{--}_{\ell r}$ are respectively bounded as follows: $\mathtt{X}^{-+}_{\ell}(p) \leq 3^{\frac{p}{2}}$, if $p|r$ and $\mathtt{X}^{--}_{\Delta}(p) \leq 3/(2\sqrt{2}) \cdot 2^{p/2}$ ($\Delta = \gcd(\ell, r)$) if $p | \ell + r$.*

*Proof.* The first inequality is proven by exploiting the relation $\mathtt{L}(n) = \phi^n + (-1/\phi)^n$, $\forall n$, [13] where $\phi$ is the golden ratio. The second is derived from: $\forall m > 0$, $\mathtt{P}(m) < 3/(2\sqrt{2}) \cdot 2^{m/2}$ which can be proven by induction on $m$. □

From Lemma 9 and (3d), it holds that the *total* numbers of attractors of $\mathcal{D}^{-+}_{\ell r}$ and of $\mathcal{D}^{--}_{\ell r}$ are bounded respectively as follows:

$$\mathtt{T}^{-+}_{\ell}(r) \leq \tfrac{1}{r} \textstyle\sum_{p|r \,\wedge\, \neg\,(p|\ell)} \varphi(\tfrac{r}{p}) \cdot 3^{p/2} \leq \mathtt{T}^{+}(r) \tag{7a}$$

$$\mathtt{T}^{--}_{\Delta}(N) \leq \tfrac{3}{2\sqrt{2}} \cdot \tfrac{1}{N} \textstyle\sum_{p|N \,\wedge\, \neg\,(p|\Delta)} \varphi(\tfrac{N}{p}) \cdot 2^{p/2} \leq \mathtt{T}^{+}(N) \tag{7b}$$

where $\Delta = \gcd(\ell, r)$, $N = \ell + r$ and $\mathtt{T}^{+}(n)$ is the *total* number of attractors of $\mathcal{C}^{+}_{n}$. By Point 1 on Page 439 the total number of attractors of $\mathcal{D}^{++}_{\ell r}$ equals:

$$\mathtt{T}^{++}(\Delta) = \mathtt{T}^{+}(\Delta). \tag{7c}$$

Also, because it can be proven that the total number of attractors of a negative BAC equals $\mathtt{T}^{-}_{n}(2n) = (\varphi \star \mathtt{X}^{-}_{n})(2n)/2n = (1/2n) \cdot \sum_{odd\ d|n} \varphi(d) \cdot 2^{n/d}$ it holds that the number of attractors of $\mathcal{C}^{+}_{n}$ and of $\mathcal{C}^{-}_{n}$ are related as follows:

$$\mathtt{T}^{-}_{n}(2n) \leq \mathtt{T}^{+}(n)/2. \tag{7d}$$

Equations (7a) to (7d) provide combinatorial comparisons between isolated positive circuits and all other networks of the types considered in this paper.

Now, let us note that the total number of attractors of a network $\mathcal{N}$ is necessarily greater than what it would be if all periodic configurations had the greatest possible minimal period, that is, $\omega = \omega(\mathcal{N})$. Conversely, the total number of attractors is necessarily smaller than what it would be if all periodic configurations had the smallest possible minimal period, that is, if they were all fix points. Thus, whatever the network $\mathcal{N}$, the following inequalities hold: $\mathtt{X}/\omega \leq \mathtt{T} \leq \mathtt{X}$. As can be shown in the context of binary necklaces (see [15] for instance), in the case of isolated Boolean automata circuits, the upper bound can actually be made much smaller:

$$\forall n \in \mathbb{N}, \quad \begin{array}{rcl} \mathtt{X}^+(n)/n = 2^n/n \leq & T^+(n) & \leq 2^{n+1}/n = 2\mathtt{X}^+(n)/n \\ \mathtt{X}^-(2n)/2n = 2^{n-1}/n \leq & T^-(n) & \leq 2^n/n = 2\mathtt{X}^-(2n)/2n \end{array} \tag{8}$$

Here, we propose the following conjecture that extends (8) to all DBACs. Its central point is the upper bound in (9).

*Conjecture 10.* Let $\mathcal{N}$ be any network that is either a BAC or a DBAC and let $\omega = \omega(\mathcal{N})$. Then, the total number $\mathtt{T}(\omega)$ of attractors of $\mathcal{N}$ is related to its total number $\mathtt{X}(\omega)$ of periodic configurations as follows:

$$\mathtt{X}(\omega)/\omega \leq \mathtt{T}(\omega) \leq K \cdot \mathtt{X}(\omega)/\omega \tag{9}$$

for a certain constant $K \leq 2$. In other words, the expected value of attractor periods of $\mathcal{N}$ is very high: $\sum_{p|\omega} \mathtt{A}(p)/\mathtt{T}(\omega) \cdot p = \mathtt{X}(\omega)/\mathtt{T}(\omega) \geq \omega/K$.

Clearly, Conjecture 10 seems to be true in the cases that are considered because $\mathtt{X}$, and as a consequence $\mathtt{T}$, grow exponentially fast. Thus, almost all periodic configurations have the greatest possible minimal period. In the context of automata networks, this can perhaps be related to the instability of automata in a periodic configuration. Indeed, let us say that an automaton $i \in V$ is unstable in $x$ when $f_i(x) \neq x_i$. Then, having very large attractor periods allows attractors in which very little automata are unstable but where the rare instabilities need a lot of time to be gradually propagated all around the circuit or double-circuit.

Besides its meaning in terms of the expected values of periods, the importance of Conjecture 10 lies in that, combined with Lemma 9, it allows to derive more precise comparisons between the total number of attractors of positive Boolean automata circuits and that of the four other types of networks that have been considered in this paper (the two first relations below have already been proven):

$$T^-(\omega) \leq 1/2 \cdot T^+(\omega) \tag{10a}$$

$$T^{++}(\omega) = T^+(\omega) \tag{10b}$$

$$T^{-+}_{\Delta}(\omega) \leq K \cdot \left(\sqrt{3}/2\right)^{\omega} \cdot T^+(\omega) \tag{10c}$$

$$T^{--}_{\Delta}(\omega) \leq K' \cdot \left(\sqrt{2}/2\right)^{\omega} \cdot T^+(\omega). \tag{10d}$$

where $K$ and $K'$ are constants that are no greater than 2.

## Conclusion

We have given explicit formulae that describe exhaustively the dynamics of Boolean automata circuits and double-circuits (DBACs). As a result we have observed that circuits that intersect tend to hinder their respective degrees of freedom: their number of possible limit behaviours or attractors falls when they are made to interact. And, what is more, according to Conjecture 10 and to its consequences (relations in (10)), it seems to decrease by an exponential factor. In addition, the presence of underlying negative circuits also seems to cause a decrease in the number of attractors of a network.

There are two immediate limits to the scope of the study presented in this article. The first is structural: the circuits intersections that have been studied are of only one type, the simplest. However, it is easy to prove that any other type of intersection between two circuits is more restrictive because, precisely, it extends over more arcs and nodes. Consequently, DBACs necessarily have no less attractors than any other intersecting circuits.

This suggests that in an arbitrary Boolean automata network, the more there are intersecting circuits and the more there are negative circuits among them, the less freely can the network behave.

The second limit concerns the very special updating schedule that has been chosen, namely the parallel update schedule. As an concluding remark, let us note that this may provide an informal argument in favour of the existence of some dynamical properties of Boolean automata networks that are related more intimately to the interaction structure of the network than to its update schedule (although these properties may require a certain special update schedule to be revealed). Indeed, it appears that in the case studied here, the restriction in the networks possible limit behaviours is due more to the structural constraint (embodied by the intersection) that imposes two circuits to "be in phase" than to the choice of the parallel update schedule.

## References

1. Adams, W., Shanks, D.: Strong primality tests that are not sufficient. Mathematics of Computation (American Mathematical Society) 39(159), 255–300 (1982)
2. Apostol, T.M.: Introduction to analytic number theory. Springer, Heidelberg (1976)
3. Aracena, J., Ben Lamine, S., Mermet, O., Cohen, O., Demongeot, J.: Mathematical modeling in genetic networks: relationships between the genetic expression and both chromosomic breakage and positive circuits. IEEE Transactions on Systems, Man, and Cybernetics 33, 825–834 (2003)
4. Demongeot, J., Noual, M., Sené, S.: Combinatorics of boolean automata circuits dynamics (2011) (submitted)
5. Elena, A.: Algorithme pour la simulation dynamique des réseaux de régulation génétique. Master's thesis, University J. Fourier (2004)
6. Goles, E.: Comportement oscillatoire d'une famille d'automates cellulaires non uniformes. Ph.D. thesis, Université scientifique et médicale de Grenoble, France (1980), http://tel.archives-ouvertes.fr/tel-00293368/fr/

7. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences of the USA 79, 2554–2558 (1982)
8. Kauffman, S.A.: Metabolic stability and epigenesis in randomly constructed genetic nets. Journal of Theoretical Biology 22, 437–467 (1969)
9. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133 (1943)
10. Puri, Y., Ward, T.: Arithmetic and growth of periodic orbits. Journal of Integer Sequences 4(2) (2001)
11. Puri, Y., Ward, T.: A dynamical property unique to the Lucas sequence. The Fibonacci Quarterly. The Official Journal of the Fibonacci Association 39(5), 398–402 (2001)
12. Remy, É., Ruet, P., Thieffry, D.: Graphic requirements for multistability and attractive cycles in a Boolean dynamical framework. Advances in Applied Mathematics 41, 335–350 (2008)
13. Ribenboim, P.: The New Book of Prime Number Records. Springer, Heidelberg (1996)
14. Richard, A.: Positive circuits and maximal number of fixed points in discrete dynamical systems. Discrete Applied Mathematics 157, 3281–3288 (2009)
15. Riordan, J.: An Introduction to Combinatorial Analysis. Wiley, New York (1980)
16. Sloane, N.J.A.: The on-line encyclopedia of integer sequences, OEIS (2008)
17. Thomas, R.: Boolean formalisation of genetic control circuits. Journal of Theoretical Biology 42, 563–585 (1973)
18. Thomas, R.: On the relation between the logical structure of systems and their ability to generate multiple steady states or sustained oscillations. Springer Series in Synergetics, vol. 9, pp. 180–193 (1981)

# Canonizable Partial Order Generators

Mateus de Oliveira Oliveira

School of Computer Science and Communication,
KTH Royal Institute of Technology, 100-44 Stockholm, Sweden
`mdeoliv@kth.se`

**Abstract.** In a previous work we introduced slice graphs as a way to specify both infinite languages of directed acyclic graphs (DAGs) and infinite languages of partial orders. Therein we focused on the study of Hasse diagram generators, i.e., slice graphs that generate only transitive reduced DAGs. In the present work we show that any slice graph can be transitive reduced into a Hasse diagram generator representing the same set of partial orders. This result allow us to establish unknown connections between the true concurrent behavior of bounded $p/t$-nets and traditional approaches for representing infinite families of partial orders, such as Mazurkiewicz trace languages and Message Sequence Chart ($MSC$) languages. Going further, we identify the family of weakly saturated slice graphs. The class of partial order languages which can be represented by weakly saturated slice graphs is closed under union, intersection and a suitable notion of complementation (bounded cut-width complementation). The partial order languages in this class also admit canonical representatives in terms of Hasse diagram generators, and have decidable inclusion and emptiness of intersection. Our transitive reduction algorithm plays a fundamental role in these decidability results.

**Keywords:** Partial Orders, Automata, Canonization.

## 1 Introduction

It is widely recognized that both the true concurrency and the causality between the events of concurrent systems can be adequately captured through partial orders [14,16,34]. In order to represent the whole concurrent behavior of systems, several methods of specifying infinite families of partial orders have been proposed. Partial languages [17], series-parallel languages [26], concurrent automata [10], causal automata [29], approaches derived from trace theory [25,28], approaches derived from message sequence chart theory [19], and more recently, Hasse diagram generators [32].

Hasse diagram generators are defined with basis on slice graphs, which by their turn, may be regarded as a specialization (modulo some convenient notational adaptations) of graph grammars [9,12]. Indeed, slice graphs may be viewed as automata that concatenate atomic blocks called slices, to generate infinite families of directed acyclic graphs (DAGs) and to represent infinite sets of partial orders. A Hasse diagram generator $\mathcal{HG}$ is a slice graph that generates exclusively transitive reduced graphs. In other words, every DAG in the graph language generated by $\mathcal{HG}$ is the Hasse diagram of the partial order it represents. Such generators were introduced by us in [32] in the context of Petri net theory, and used to solve different open problems related to the partial order semantics of bounded *place/transition*-nets ($p/t$-nets). For instance, we showed that the set of partial order runs of any bounded $p/t$-net $N$ can be represented by an effectively constructible Hasse diagram generator $\mathcal{HG}_N$ [1]. Previously, approaches that mapped behavioral objects to $p/t$-nets were either not expressive enough to fully capture partial order behavior of bounded $p/t$-nets, or were not guaranteed to be finite and thus, not effective [13,18,21].

In [32] we also showed how to use Hasse diagram generators to verify the partial order behavior of concurrent systems modeled through bounded $p/t$-nets. More precisely, given a bounded $p/t$-net $N$ with partial order behavior $\mathcal{L}_{PO}(N)$ and a HDG $\mathcal{HG}$ representing a set $\mathcal{L}_{PO}(\mathcal{HG})$ of partial orders, we may effectively verify both whether $\mathcal{L}_{PO}(\mathcal{HG})$ is included into $\mathcal{L}_{PO}(N)$ and whether their intersection is empty. Previously an analogous verification result was only known for finite languages of partial orders [27]. As a meta-application of this verification result, we were able to test the inclusion of the partial order behavior of two bounded $p/t$-nets $N_1$ and $N_2$: Compute $\mathcal{HG}_{N_1}$ and test whether $\mathcal{L}_{PO}(\mathcal{HG}_{N_1}) \subseteq \mathcal{L}_{PO}(N_2)$. The possibility of performing such an inclusion test for bounded $p/t$-nets had been open for at least a decade. In the nineties, Jategaonkar-Jagadeesan and Meyer [23] proved that the inclusion of the causal behavior of 1-safe $p/t$-nets is decidable, and Montanari and Pistore [29] showed how to determine whether two bounded nets have bisimilar causal behaviors.

Finally, Hasse diagram generators may be used to address the synthesis of concurrent systems from behavioral specifications. The idea of the synthesis is appealing: Instead of constructing a system and verifying if it behaves as expected, we specify a priori which runs should be present on it, and then automatically construct a system satisfying the given specification [24]. In our setting the systems are modeled via $p/t$-nets and the specification is made in terms of Hasse diagram generators. In [32] we devised an algorithm that takes a Hasse diagram generator $\mathcal{HG}$ and a bound $b$ as input, and determines whether there is a $b$-bounded $p/t$-net whose partial order behavior includes $\mathcal{L}_{PO}(\mathcal{HG})$. If such a net exists, the algorithm returns the net $N$ whose behavior minimally includes $\mathcal{L}_{PO}(\mathcal{HG})$. More precisely for every other $b$-bounded $p/t$-net $N'$ satisfying $\mathcal{L}_{PO}(\mathcal{HG}) \subseteq \mathcal{L}_{PO}(N')$ it is guaranteed that $\mathcal{L}_{PO}(N) \subseteq \mathcal{L}_{PO}(N')$. This implies in particular, that if the set of runs specified by $\mathcal{HG}$ indeed matches the partial order behavior of a $b$-bounded $p/t$-net $N$, then this net will be returned. The synthesis of $p/t$-nets from finite sets of partial orders was accomplished in [4] and subsequently generalized in [5] to infinite languages specified by rational expressions over partial orders, which are nevertheless not expressive enough to represent the whole behavior of arbitrary bounded $p/t$-nets. For other results considering the synthesis of several types of Petri nets from several types of

[1] Indeed we can chose to derive from $N$ a HDG $\mathcal{HG}_N^{ex}$ representing the set of true concurrent runs of $N$ or a HDG $\mathcal{HG}_N^{cau}$ representing the set of causal runs of $N$.

automata and languages, specifying both sequential and step behaviors we point to [2,11,20].

## 2 Transitive Reduction of Slice Graphs and Its Consequences

Both the verification and the synthesis of $p/t$-nets described in the previous section are stated in function of Hasse diagram generators, and do not extend directly to general slice graphs. The main goal of this paper is to overcome this limitation, by proving that any slice graph can be transitive reduced into a Hasse diagram generator specifying the same partial order language.

**Theorem 1 (Transitive Reduction of General Slice Graphs).** *Any slice graph $\mathcal{SG}$ can be transitive reduced into a Hasse diagram generator $\mathcal{HG}$ representing the same partial order language, i.e., $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{HG})$.*

This result is interesting for two main reasons: First slice graphs are much more flexible than Hasse diagram generators from a specification point of view. Second it establishes interesting connections between $p/t$-nets and well known formalisms aimed to specify infinite families of partial orders, such as Mazurkie-wicz trace languages [28] and message sequence chart (MSC) languages [19,30]. More precisely, we prove that if a partial order language $\mathcal{L}_{PO}$ is specified through a pair $(\mathcal{A}, I)$ of finite automaton $\mathcal{A}$ over an alphabet of events $\Sigma$ and a Mazurkie-wicz independence relation $I \subseteq \Sigma \times \Sigma$, then there is a slice graph $\mathcal{SG}$ representing the same set of partial orders. A similar result holds if $\mathcal{L}_{PO}$ is specified by a high-level message sequence chart (HMSC), or equivalently, by a message sequence graph (MSG)[1,31]. We point out that in general, the slice graphs arising from these transformations may be far from being transitive reduced and that a direct translation of these approaches in terms of Hasse diagram generators is not evident. Nevertheless, Theorem 8 guarantees that these slice graphs can be indeed transitive reduced into Hasse diagram generators representing the same partial order language, allowing us in this way to apply both our verification and synthesis results:

**Theorem 2 (Verification).** *Let $N$ be a bounded $p/t$-net and $\mathcal{L}_{PO}$ a partial order language generated by pair $(\mathcal{A}, I)$ of finite automaton and independence relation, or by a message sequence graph $\mathcal{M}$. Then we may effectively verify whether $\mathcal{L}_{PO} \subseteq \mathcal{L}_{PO}(N)$ and whether $\mathcal{L}_{PO} \cap \mathcal{L}_{PO}(N) = \emptyset$.*

In Theorem 3 below we address the synthesis of (unlabeled) $p/t$-nets from partial order languages represented by traces or message sequence graphs. We point out that the synthesis of labeled $p/t$-nets (i.e., nets in which two transitions may be labeled by the same action) from Mazurkievicz trace languages and from local trace languages [21] was addressed respectively in [22] and in [25]. However there is a substantial difference between labeled and unlabeled $p/t$-nets when it comes to partial order behavior. For instance, if we allow the synthesized nets to be labeled, we are helped by the fact that labeled 1-safe $p/t$-nets are already as partial order expressive as their $b$-bounded counterparts [6]. Thus the synthesis of unlabeled nets tends to be harder.

**Theorem 3 (Synthesis).** *Let $b \in \mathbb{N}$ be a bound and $\mathcal{L}_{PO}$ be a partial order language represented by a pair $(\mathcal{A}, I)$ of automaton and independence relation, or by a message sequence graph $\mathcal{M}$. Then we may effectively compute a $b$-bounded p/t-net $N$ whose partial order behavior minimally includes $\mathcal{L}_{PO}$.*

Our transitive reduction algorithm is also a necessary step towards the canonization of slice graphs with respect to the partial order language they represent. We say that a function $\mathcal{C}$ canonizes slice graphs if for every slice graph $\mathcal{SG}$, $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{C}(\mathcal{SG}))$ and $\mathcal{C}(\mathcal{SG}) \sim \mathcal{C}(\mathcal{SG}')$ for all other slice graph $\mathcal{SG}'$ satisfying $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{SG}')$. In the same way that a Hasse diagram provides a minimal representation for its induced partial order, it is natural that Hasse diagram generators correspond to the canonical forms of slice graphs. However simply transitive reducing a slice graph is not sufficient to put it into a canonical form, and indeed canonization is in general uncomputable. Fortunately, there is a very natural and decidable[2] subclass of slice graphs (weakly saturated slice graphs) for which canonization is feasible. Besides admitting canonical representatives, partial order languages represented by weakly saturated slice graphs are closed under union, intersection and even under a special notion of complementation, which we call *bounded cut-width complementation.* Furthermore inclusion (and consequently, equality) and emptiness of intersection are decidable for this class of languages. Transitive reduction will play a **fundamental role** in these decidability results, for the reasons that we describe in the next paragraph, and in the definition of bounded cut-width complementation.

A slice graph $\mathcal{SG}$ is meant to represent three distinct languages: A slice language $\mathcal{L}(\mathcal{SG})$ which is a regular subset of the free monoid generated by a slice alphabet $\Sigma_{\mathbb{S}}$; a graph language $\mathcal{L}_G(\mathcal{SG})$ consisting of the DAGs which have a string representative in the slice language; and a partial order language $\mathcal{L}_{PO}(\mathcal{SG})$ obtained by taking the transitive closure of DAGs in the graph language. As we show in the full version of this paper, any weakly saturated slice graph can be efficiently transformed into a stronger form, wich we call *saturated slice graph*, representing the same partial order language. It turns out that questions about the graph language of saturated slice graphs can be translated into questions about their slice languages. Indeed this fact will follow by interpreting saturated slice languages as being closed under a certain commutation operation on the slice alphabet. If additionally, the slice graphs in consideration are Hasse diagram generators, then questions about their partial order languages can be further mapped to questions about their graph languages, paving in this way a path to decidability. The crucial point to be stressed is that this last observation fails badly if the slice graphs are not transitive reduced: There exist (**even saturated**) slice graphs $\mathcal{SG}$ and $\mathcal{SG}'$ for which $\mathcal{L}_G(\mathcal{SG}) \cap \mathcal{L}_G(\mathcal{SG}') = \emptyset$ but $\mathcal{L}_{PO}(\mathcal{SG}) \cap \mathcal{L}_{PO}(\mathcal{SG}) \neq \emptyset$, or for which $\mathcal{L}_G(\mathcal{SG}) \nsubseteq \mathcal{L}_G(\mathcal{SG}')$ but $\mathcal{L}_{PO}(\mathcal{SG}) \subseteq \mathcal{L}_{PO}(\mathcal{SG})$. Thus it is crucial that we transitive reduce slice graphs before performing operations with their partial

2 In [19] it is undecidable whether a MSC-language is linearization-regular. This is not in contradiction with the decidability of weak saturation. An analogous statement for us would be: It is undecidable whether a slice graph can be weakly saturated.

order languages. With regard to this observation, an important feature of our transitive reduction algorithm is that it preserves weak saturation.

A skeptic could wonder whether weak saturation is an excessively strong condition which could be only satisfied by uninteresting examples of slice graphs. We counter this skepticism by describing three natural situations in which weakly saturated slice graphs arise: The first two examples stem from the fact that our study of weakly saturated slice languages was inspired, and indeed generalizes, both the theory of recognizable trace languages [28] and the theory of linearization-regular[3] message sequence languages [19]. In particular, recognizable trace languages can be mapped to weakly saturated regular slice languages, while linearization-regular MSC languages which are representable by message sequence graphs, may be mapped to *loop connected* slice graphs, which can be efficiently weakly saturated. Our third and most important example comes from the theory of bounded $p/t$-nets. More precisely, we show that the Hasse diagram generators associated to bounded $p/t$-nets in [32] are saturated. This last observation has two important consequences: first, slice graphs are strictly more expressive than both Mazurkievicz trace languages, and MSC-languages, since there exist even 1-safe $p/t$-nets whose partial order behavior cannot be expressed through these formalisms; second, it implies that the behavior of bounded $p/t$-nets may be canonically represented by Hasse diagram generators. While in [32] we were able to associate a HDG $\mathcal{HG}_N$ to any bounded $p/t$-net $N$, we were not able to prove that if two nets $N$ and $N'$ have the same partial order behavior then they can be associated to same HDG[4]. By showing that the partial order language of bounded $p/t$-nets may be represented via saturated slice languages we are able to achieve precisely this goal:

**Corollary 4 (Canonical Hasse Diagram Generators and $p/t$-Nets).** *The partial order behavior of any bounded $p/t$-net $N$ can be canonically represented by a Hasse diagram generator $\mathcal{HG}_N$. In particular for any other bounded $p/t$-net $N'$ satisfying $\mathcal{L}_{PO}(N) = \mathcal{L}_{PO}(N')$ it holds that $\mathcal{HG}_N = \mathcal{HG}_{N'}$.*

Due to lack of space, in this extended abstract we only deal with our transitive reduction algorithm, which is in the core of all other results. We leave our development of weakly saturated slice languages, the reductions from Mazurkiewicz trace languages and MSC languages to slice graphs, as well as their relationship to theory of $p/t$-nets, to the full version of this paper which is available at the homepage of the author by the time this extended abstract is being submitted.

## 3 Slices

There are several automata theoretical approaches for the specification of infinite families of graphs: graph automata [33], automata over planar DAGs [7], graph

[3] In our work the term regular is used in the standard sense of finite automata theory. The notion of "regular" used in [19] is analogous to our notion of regular+saturated.

[4] In general, a partial order language can be represented by several distinct Hasse diagram generators.

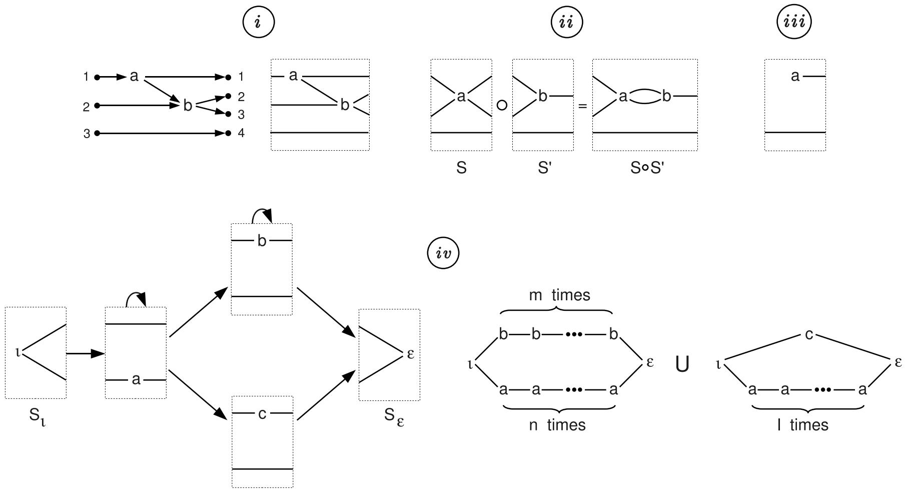


**Fig. 1.** *i*) A slice *ii*) Composition of slices. *iii*) A degenerate slice. *iv*) A slice graph labeled with unit slices, and an intuitive representation of its graph language. $\mathbf{S}_\iota$ is initial and $\mathbf{S}_\varepsilon$ i is final.

rewriting systems [3,9,12], and others [8,15]. In this section we will introduce an approach that is more suitable for our needs. Namely, the representation of infinite families of DAGs with bounded cut width. In particular, the slices defined in this section can be regarded as a specialized version of the multi-pointed graphs defined in [12], which are too general, and which are subject to a slightly different notion of concatenation.

A *slice* is a labeled DAG $\mathbf{S} = (V, E, l)$ whose vertex set $V$ is partitioned into three subsets: A non-empty center $C$ labeled by $l$ with the elements of an arbitrary set $T$ of events, and the in- and out-frontiers $I$ and $O$ respectively which are numbered by $l$ in such a way that $l(I) = \{1, \cdots, |I|\}$ and $l(O) = \{1, ..., |O|\}$. Furthermore a unique edge in $E$ touches each frontier vertex $v \in I \dot\cup O$, where $\dot\cup$ denotes the disjoint union of sets. This edge is outgoing if $v$ lies on the in-frontier $I$ and incoming if $v$ lies on the out-frontier $O$. In drawings, we surround slices by rectangles, and implicitly direct their edges from left to right. In-frontier and out-frontier vertices are determined respectively by the intersection of edges with the left and right sides of the rectangle. Frontier vertices are implicitly numbered from top to bottom. Center vertices are indicated by their labels (Fig. 1-*i*).

A slice $\mathbf{S}_1$ can be composed with a slice $\mathbf{S}_2$ whenever the out-frontier of $\mathbf{S}_1$ is of the same size as the in-frontier of $\mathbf{S}_2$. In this case, the resulting slice $\mathbf{S}_1 \circ \mathbf{S}_2$ is obtained by gluing the single edge touching the $j$-th out-frontier vertex of $\mathbf{S}_1$ to the corresponding edge touching the $j$-th in-frontier vertex of $\mathbf{S}_2$ (Fig. 1-*ii*). We note that as a result of the composition, multiple edges may arise, since the vertices on the glued frontiers disappear. A slice with a unique vertex in the center is called a *unit slice*. A sequence of unit slices $\mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n$ is a *unit decomposition* of a slice $\mathbf{S}$ if $\mathbf{S} = \mathbf{S}_1 \circ \mathbf{S}_2 \circ \cdots \circ \mathbf{S}_n$. The definition of unit decomposition extends to DAGs by regarding them as slices with empty in and out-frontiers. The slice-width of a slice is defined as the size of its greater frontier. The slice width of a unit decomposition $\mathbf{S} = \mathbf{S}_1 \circ \mathbf{S}_2 \circ \cdots \circ \mathbf{S}_n$ is the slice-width of its widest slice.

We say that a slice is *initial* if its in-frontier is empty and *final* if its out-frontier is empty. A unit slice is non-degenerate if its center vertex is connected to at least one in-frontier (out-frontier) vertex whenever the in-frontier (out)-frontier is not empty. In Fig. 1-*iii* we depict a degenerate unit slice. A *slice alphabet* is any finite set $\Sigma_{\mathbb{S}}$ of slices. The slice alphabet of width $c$ over a set of events $T$ is the set $\Sigma_{\mathbb{S}}^c$ of all unit slices of width at most $c$, whose center vertex is labeled with an event from $T$. A *slice language* over a slice alphabet $\Sigma_{\mathbb{S}}$ is a subset $\mathcal{L} \subseteq \Sigma_{\mathbb{S}}^*$ where for each string $\mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n \in \mathcal{L}$, $\mathbf{S}_1$ is initial, $\mathbf{S}_n$ is final and $\mathbf{S}_i$ can be composed with $\mathbf{S}_{i+1}$ for $1 \leq i < n$. From a slice language $\mathcal{L}$ we may derive a language $\mathcal{L}_G$ of DAGs by composing the slices in the strings of $\mathcal{L}$, and a language $\mathcal{L}_{PO}$ of partial orders, by taking the transitive closure of each DAG in $\mathcal{L}_G$:

$$\mathcal{L}_G = \{\mathbf{S}_1 \circ \mathbf{S}_2 \circ \cdots \circ \mathbf{S}_n | \mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n \in \mathcal{L}\} \quad \text{and} \quad \mathcal{L}_{PO} = \{H^* | H \in \mathcal{L}_G\} \quad (1)$$

In this paper we assume that all slices in a slice alphabet $\Sigma_{\mathbb{S}}$ are unit and non-degenerate, but this restriction is not crucial. With this assumption however, every DAG in the graph language derived from a slice language has a unique minimal and a unique maximal vertex.

A slice language is regular if it is generated by a finite automaton or regular expressions over slices[5]. We notice that a slice language is a subset of the free monoid generated by a slice alphabet $\Sigma_{\mathbb{S}}$ and thus we do not need to make a distinction between regular and rational slice languages. In particular every slice language generated by a regular expression can be also generated by a finite automaton. Equivalently, a slice language is regular if and only if it can be generated by the slice graphs defined below (see appendix of [32]):

**Definition 5 (Slice Graph).** *A* slice graph *over a slice alphabet $\Sigma_{\mathbb{S}}$ is a labeled directed graph $\mathcal{SG} = (\mathcal{V}, \mathcal{E}, \mathcal{S})$ possibly con-tai-ning loops but without multiple edges. The function $\mathcal{S} : \mathcal{V} \rightarrow \Sigma_{\mathbb{S}}$ satisfies the following condition: $(v_1, v_2) \in \mathcal{E}$ implies that $\mathcal{S}(v_1)$ can be composed with $\mathcal{S}(v_2)$. We say that a vertex on a slice graph is initial if it is labeled with an initial slice and final if it is labeled with a final slice. The slice language generated by $\mathcal{SG}$ is defined as follows:*

$$\mathcal{L}(\mathcal{SG}) = \{\mathcal{S}(v_1)\mathcal{S}(v_2) \cdots \mathcal{S}(v_n) : v_1v_2 \cdots v_n \textit{ is a walk on } \mathcal{SG} \textit{ from an initial to a final vertex}\}.$$

We write respectively $\mathcal{L}_G(\mathcal{SG})$ and $\mathcal{L}_{PO}(\mathcal{SG})$ for the graph and the partial order languages derived from $\mathcal{L}(\mathcal{SG})$. A slice language $\mathcal{L}$ is *transitive reduced* if all DAGs in $\mathcal{L}_G$ are simple and transitive reduced. In other words, each DAG in $\mathcal{L}_G$ is the Hasse diagram of a partial order in $\mathcal{L}_{PO}$. A slice graph is a *Hasse diagram generator* if its slice language is transitive reduced.

[5] The operation of the monoid is just the concatenation $\mathbf{S}_1\mathbf{S}_2$ of slice symbols $\mathbf{S}_1$ and $\mathbf{S}_2$ and should not be confused with the composition $\mathbf{S}_1 \circ \mathbf{S}_2$ of slices.

## 4 Sliced Transitive Reduction

In [32] we devised a method to filter out from the graph language of a slice graph $\mathcal{SG}$ all *DAGs* which are not transitive reduced. In this way we were able to obtain a Hasse diagram generator $\mathcal{HG}$ whose graph language consists precisely on the Hasse diagrams generated by $\mathcal{SG}$ (i.e. $\mathcal{L}_{PO}(\mathcal{HG}) \subseteq \mathcal{L}_{PO}(\mathcal{SG})$). The method we devised therein falls short of being a transitive reduction algorithm, since the partial order generated by the resulting slice graph $\mathcal{HG}$ could be significantly shrunk and indeed even reduced to the empty set. It was not even clear whether such a task could be accomplished at all, since we are dealing with the application of a non-trivial algorithm, i.e. the transitive reduction, to an infinite number of DAGs at the same time. Fortunately in this section we prove that such a transitive reduction is accomplishable, by developing an algorithm that takes a slice graph as input and returns a Hasse diagram generator $\mathcal{HG}$ satisfying $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{HG})$.

The difficulty in devising an algorithm to transitive reduce slice graphs stems from the fact that a slice that labels a vertex $\mathfrak{v}$ of a slice graph may be used to form both *DAGs* that are transitive reduced and *DAGs* that are not, depending on which path we are considering in the slice graph. This observation is illustrated in Figure 3.*ii* where the slice containing the event $a$ has this property. Thus in general the transitive reduction cannot be performed independently on each slice of the slice graph. Fortunately we will overcome this difficulty by introducing in Definition 6 and Lemma 7 a "sliced" characterization of superfluous edges of DAGs, i.e., edges that do not carry any useful transitivity information. By expanding each slice of the slice graph with a set of specially tagged copies satisfying the properties defined in Definition 6 and connecting them in a special way, we will be able to keep all paths which give rise to transitive reduced DAGs, and to create new paths which will give rise to transitive reduced versions of the non transitive-reduced DAGs generated by the original slice graph. In the proof of Theorem 8 we develop an algorithm that transitive reduces slice graphs which do not generate DAGs with multiple edges. Subsequently, in Theorem 8 we will eliminate the restriction on multiple edges and prove that slice graphs in general can be transitive reduced.

We say that an edge $e$ of a simple DAG $H$ is superfluous if the transitive closure of $H$ equals the transitive closure of $H \backslash \{e\}$. In this section we will develop a method to highlight the sliced parts of superfluous edges of a graph $H$ on any of its unit decompositions $\mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n$ (Fig. 2-*ii*). Deleting these highlighted edges from each slice of the decomposition, we are left with a unit decomposition $\mathbf{S}'_1\mathbf{S}'_2 \cdots \mathbf{S}'_n$ of the transitive reduction of $H$. It turns out that we may transpose this process to slice graphs. Thus given a slice graph $\mathcal{SG}$ we will be able to effectively compute a Hasse diagram generator $\mathcal{HG}$ that represents the same language of partial order as $\mathcal{SG}$, i.e. $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{HG})$.

A function $\mathcal{T} : E^2 \rightarrow \{0,1\}^2$ defined on the pairs of edges of a unit slice $\mathbf{S} = (\{v\}, E, l)$ is called a coloring of $\mathbf{S}$. A sequence of functions $\mathcal{T}_1\mathcal{T}_2 \cdots \mathcal{T}_n$ is a coloring of a unit decomposition $\mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n$ of a DAG $H$ if each $\mathcal{T}_i$ is a coloring of $\mathbf{S}_i$ and if the colors associated by $\mathcal{T}_i$ to pairs of edges touching the out-frontier

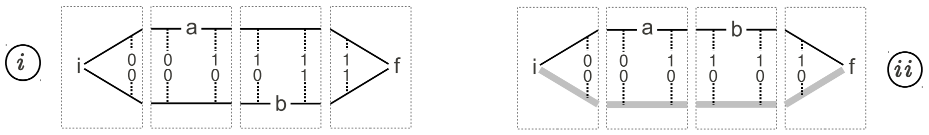


**Fig. 2.** The transitivity coloring of the unit decomposition of two DAGs. *i*) The DAG is transitive reduced. No edge is marked. *ii*) The DAG is not transitive reduced. The sliced parts of each superfluous edge are marked (in gray). Deleting the marked edges and composing the slices we are left with the transitive reduction of the original DAG.

of $\mathbf{S}_i$ agree with the colors associated by $\mathcal{T}_{i+1}$ to pairs of edges touching the in-frontier of $\mathbf{S}_{i+1}$ (Figs. 2-*i*,2-*ii*). Below we define the notion of *transitivity coloring* that will allow us to perform a "sliced" transitive reduction on *DAGs*. We say that an edge $e$ of a slice $\mathbf{S}$ is marked by $\mathcal{T}$ if $\mathcal{T}(ee) = 11$ and unmarked if $\mathcal{T}(ee) = 00$.

**Definition 6 (Transitivity Coloring).** *Let* $\mathbf{S} = (I \cup \{v\} \cup O, E, l)$ *be a unit slice. Then a* transitivity coloring *of* $\mathbf{S}$ *is a partial function* $\mathcal{T} : E^2 \rightarrow \{0,1\}^2$ *such that*

1. *Undefinedness:* $\mathcal{T}(e_1e_2)$ *is not defined if and only if* $(e_1^s{=}e_2^t)$ *or* $(e_1^t{=}e_2^s)$
2. *Antisymmetry: If* $\mathcal{T}(e_1e_2) = ab$ *then* $\mathcal{T}(e_2e_1) = ba$.
3. *Marking:* $\mathcal{T}(ee) \in \{00, 11\}$. $e$ *is unmarked if* $\mathcal{T}(ee) = 00$ *and marked if* $\mathcal{T}(ee) = 11$.
4. *Transitivity:*
   (a) *If* $e_1$ *and* $e_2 \neq e_1$ *have the same source vertex, then* $\mathcal{T}(e_1e_2) = 00$.
   (b) *If* $e_1^s \in I$ *and* $e_1^t \in O$ *and* $e_2^s = v$ *then* $\mathcal{T}(e_1e_2) \in \{01, 11\}$ *and* $\mathcal{T}(e_1e_2) = 01$ *iff* $(\exists e, e^t{=}v)(\mathcal{T}(e_1e) \in \{00, 01\})$
5. *Relationship between marking and transitivity:*

$$\textit{If } e^t = v \textit{ then } e \textit{ is marked } \Leftrightarrow (\exists e_1, e_1^t = v)\mathcal{T}(ee_1) = 01.$$

We observe that an isolated unit slice may be transitivity colored in many ways. However as stated in the next lemma (Lemma 7), a unit decomposition $\mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n$ of a simple DAG $H$ with a unique minimal and a unique maximal vertex, can be coherently colored in a unique way. Furthermore, in this unique coloring, each superfluous edge of $H$ is marked.

**Lemma 7 (Sliced Transitive Reduction).** *Let* $\mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n$ *be a unit decomposition of a simple DAG* $H$ *with a unique minimal and a unique maximal vertex. Then*

1. $\mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n$ *has a unique transitivity coloring* $\mathcal{T}_1\mathcal{T}_2 \cdots \mathcal{T}_n$. *Furthermore,*
2. *an edge* $e$ *in* $\mathbf{S}_i$ *is marked by* $\mathcal{T}_i$ *if and only if* $e$ *is a sliced part of a superfluous edge of* $H$ *(Fig. 2-ii).*

*Proof.* Let $\mathcal{T}_1\mathcal{T}_2\cdots\mathcal{T}_n$ be a transitivity coloring of $\mathbf{S}_1\mathbf{S}_2\cdots\mathbf{S}_n$. By the rule of composition of colored slices and by conditions 1 to 5 of Definition 6, the value associated by $\mathcal{T}_i$ to each pair of distinct edges of $\mathbf{S}_i$ are completely determined by the values associated by $\mathcal{T}_{i-1}$ to edges touching the out-frontier of $\mathbf{S}_{i-1}$. Furthermore, since $H$ has a unique minimal vertex, $\mathcal{T}_1$ associates the value 00 to each pair of distinct edges of $\mathbf{S}_1$. Thus the values associated by each $\mathcal{T}_i$ to distinct edges of $\mathbf{S}_i$ are unique. It remains to show that the marking is unique.

Let $e$ be a superfluous edge of $H$, and $e_1e_2..e_k$ be a path from $e^s$ to $e^t$, then the transitivity conditions in Definition 6.4 assure that for any sliced part $e'$ of $e$ and any sliced part $e'_i$ of $e_i$ lying in the same slice $\mathbf{S}_j$, $\mathcal{T}_j(e', e'_i) = 00$ if $i = 1$ and $\mathcal{T}_j(e', e'_i) = 01$ for $2 \leq i \leq k$ (Fig. 2). Let $S_j$ be the slice that contains the target vertex of $e$, and let $e'$ and $e'_k$ be respectively the sliced parts of $e$ and $e_k$ lying in $S_j$. Then $\mathcal{T}(e'e'_k) = 01$ and thus, by condition 5 of Definition 6, $e'$ is marked, implying that any sliced part of $e$ lying in previous slices must be marked as well. Now suppose that $e_1$ and $e_2$ have the same target vertex, and that $e_1$ is not superfluous. Then for any sliced part $e'_1$ of $e_1$ and any sliced part $e'_2$ of $e_2$ lying in the same slice $S_j$, we must have $\mathcal{T}_j(e'_1, e'_2) = 10$ if $e_2$ is superfluous and $\mathcal{T}_j(e'_1, e'_2) = 11$ if $e_2$ is not superfluous. Thus by condition 5 of Definition 6, no sliced part of $e_1$ can be marked. We observe that $\mathcal{T}_j(e'_1, e'_2) \neq 00$ since otherwise $e_1$ and $e_2$ would have the same source and thus form a multiple edge. $\square$

In this extended abstract we deal only with the transitive reduction of slice graphs whose graph language does not contain DAGs with multiple edges (Theorem 8). We say that these slice graphs are *simple*. Lemma 7 is of special importance for its proof. The transitive reduction of general slice graphs (Theorem 1) is more technically involved and is treated in details in the full version of this paper.

**Theorem 8 (Transitive Reduction of Simple Slice Graphs).** *Let* $\mathcal{SG} = (\mathcal{V}, \mathcal{E}, \mathcal{S})$ *be a slice graph such that* $\mathcal{L}_G(\mathcal{SG})$ *has only simple DAGs. Then there exists a Hasse diagram generator* $\mathcal{HG}$ *such that* $\mathcal{L}_{PO}(\mathcal{SG}) = \mathcal{L}_{PO}(\mathcal{HG})$.

*Proof.* As a first step we construct an intermediary slice graph $\mathcal{SG}'$ as follows: we expand each vertex $v$ in $\mathcal{V}$ with a set of vertices $\{v_{\mathcal{T}}\}$ where $\mathcal{T}$ ranges over all transitivity colorings of $\mathcal{S}(v)$. Each vertex in $\{v_{\mathcal{T}}\}$ is labeled with $\mathcal{S}(v)$. We add an edge from $v_{\mathcal{T}}$ to $v'_{\mathcal{T}'}$ in $\mathcal{SG}'$ if and only if $v$ is connected to $v'$ in $\mathcal{SG}$ and if the values associated by $\mathcal{T}$ to the edges touching the out-frontier of $\mathcal{S}(v)$ agree with the values associated by $\mathcal{T}'$ to the edges touching the in-frontier of $\mathcal{S}(v')$. Finally we delete vertices that cannot be reached from an initial vertex, or that cannot reach a final vertex. We note that $\mathcal{T}_1\mathcal{T}_2\cdots\mathcal{T}_n$ is a transitivity coloring of the label $\mathcal{S}(v^1)\mathcal{S}(v^2)\cdots\mathcal{S}(v^n)$ of a walk from a initial vertex $v_1$ to a final vertex $v_2$ in $\mathcal{SG}$, if and only if $\mathcal{S}(v^1)\mathcal{S}(v^2)\cdots\mathcal{S}(v^n)$ also labels the walk $v^1_{\mathcal{T}_1}v^2_{\mathcal{T}_2}\cdots v^n_{\mathcal{T}_n}$ in the new slice graph $\mathcal{SG}'$. By Lemma 7 ( Item 1) a coloring exists for each such a walk and thus $\mathcal{L}_G(\mathcal{SG}) = \mathcal{L}_G(\mathcal{SG}')$. In order to get the Hasse diagram generator $\mathcal{HG}$ with the same partial order language as $\mathcal{SG}$, we relabel each vertex $v_{\mathcal{T}}$ with a version of $\mathcal{S}(v)$ in which the edges which are marked by $\mathcal{T}$ are deleted. By

Lemma 7.2 a DAG is in $\mathcal{L}_G(\mathcal{HG})$ if and only if it is the transitive reduction of a *DAG* in $\mathcal{SG}$, and thus $\mathcal{L}_{PO}(\mathcal{HG}) = \mathcal{L}_{PO}(\mathcal{SG})$. □

We end this section by giving a simple upper bound on the complexity of the transitive reduction of slice graphs (Corollary 9).

**Corollary 9.** *Let $\mathcal{SG}$ be a slice graph with $n$ vertices and let $s$ be the size of the greatest frontier of a slice labeling a vertex of $\mathcal{SG}$. Then the Hasse diagram generator constructed in in Theorem 8 has $n \cdot 2^{O(s^2)}$ vertices. In particular, the transitive reduction algorithm runs in polynomial time for $s = O(\sqrt{\log n})$.*

*Proof.* Let $v$ be a vertex of $\mathcal{SG}$ which is labeled with a slice $\mathbf{S}$ of width $s$. Then $\mathbf{S}$ can be colored in at most $2^{O(s^2)}$ ways, since each two edges touching the same frontier of $\mathbf{S}$ can be colored in at most a constant number of ways. Since the $\mathcal{SG}$ has $n$ vertices, the bound of $n \cdot 2^{O(s^2)}$ follows. □

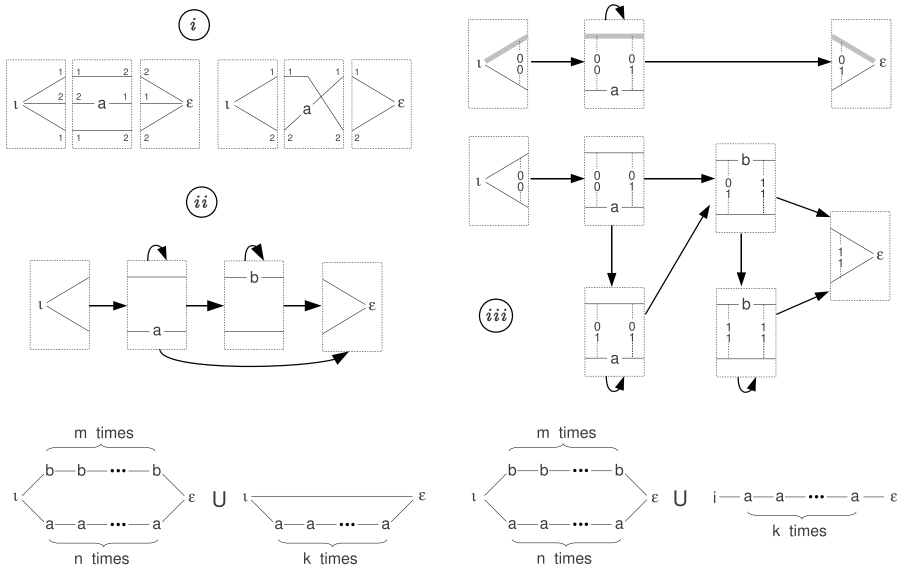


**Fig. 3.** *i*) How multiple edges are collapsed (dealt with in the full version) *ii*) A slice graph and an intuitive depiction of its graph language. *iii*) The Hasse diagram generator obtained from the slice graph to the left. The marked edges (in gray), which should be deleted, and the values of the coloring were left to illustrate the proof of Theorem 8.

**Acknowledgements.** Part of this work was done while the author was visiting the *Laboratoire de Recherche en Informatique, U. Paris Sud, France.* This work was partially supported by Julia Kempe's Israel Science Foundation grant, by Julia Kempe's European Research Council (ERC) Starting Grant QUCO, as well as by the Wolfson Family Charitable Trust.

## References

1. Alur, R., Yannakakis, M.: Model Checking of Message Sequence Charts. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 114–129. Springer, Heidelberg (1999)
2. Badouel, E., Darondeau, P.: Theory of Regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
3. Bauderon, M., Courcelle, B.: Graph expressions and graph rewritings. Mathematical Systems Theory 20(2-3), 83–127 (1987)
4. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of Petri nets from finite partial languages. Fundamenta Informaticae 88(4), 437–468 (2008)
5. Bergenthum, R., Desel, J., Mauser, S., Lorenz, R.: Synthesis of Petri nets from term based representations of infinite partial languages. Fundamenta Informaticae 95(1), 187–217 (2009)
6. Best, E., Wimmel, H.: Reducing $k$-Safe Petri Nets to Pomset-Equivalent 1-Safe Petri Nets. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 63–82. Springer, Heidelberg (2000)
7. Bossut, F., Dauchet, M., Warin, B.: A Kleene theorem for a class of planar acyclic graphs. Inform. and Comput. 117(2), 251–265 (1995)
8. Bozapalidis, Kalampakas: Recognizability of graph and pattern languages. Acta Informatica 42 (2006)
9. Courcelle, B.: Graph expressions and graph rewritings. Mathematical Systems Theory 20, 83–127 (1987)
10. Droste, M.: Concurrent automata and domains. International Journal of Foundations of Computer Science 3(4), 389–418 (1992)
11. Ehrenfeucht, A., Rozenberg, G.: Partial (set) 2-structures. Part I: Basic notions and the representation problem. Acta Informatica 27(4), 315–342 (1989)
12. Engelfriet, J., Vereijken, J.J.: Context-free graph grammars and concatenation of graphs. Acta Informatica 34 (1997)
13. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. Formal Methods in System Design 20(3), 285–310 (2002)
14. Gaifman, H., Pratt, V.R.: Partial order models of concurrency and the computation of functions. In: Proc. of LICS 1987, pp. 72–85 (1987)
15. Giammarresi, D., Restivo, A.: Two-dimensional finite state recognizability. Fundamenta Informaticae 25(3), 399–422 (1996)
16. Gischer, J.L.: The equational theory of pomsets. Theoret. Computer Science 61, 199–224 (1988)
17. Grabowski, J.: On partial languages. Fundamenta Informaticae 4(2), 427 (1981)
18. Hayman, J., Winskel, G.: The unfolding of general Petri nets. In: Proc. of FTTCS 2008. LIPIcs, vol. 2, pp. 223–234 (2008)
19. Henriksen, J.G., Mukund, M., Kumar, K.N., Sohoni, M.A., Thiagarajan, P.S.: A theory of regular MSC languages. Inform. and Comput. 202(1), 1–38 (2005)
20. Hoogers, P., Kleijn, H., Thiagarajan, P.: A trace semantics for Petri nets. Inform. and Comput. 117(1), 98–114 (1995)
21. Hoogers, P.W., Kleijn, H.C.M., Thiagarajan, P.S.: An event structure semantics for general Petri nets. Theoret. Computer Science 153(1-2), 129–170 (1996)
22. Husson, J.-F., Morin, R.: On Recognizable Stable Trace Languages. In: Tiuryn, J. (ed.) FOSSACS 2000. LNCS, vol. 1784, pp. 177–191. Springer, Heidelberg (2000)
23. Jategaonkar, L., Meyer, A.R.: Deciding true concurrency equivalences on safe, finite nets. Theoret. Computer Science 154(1), 107–143 (1996)

24. Kupferman, O., Lustig, Y., Vardi, M.Y., Yannakakis, M.: Temporal synthesis for bounded systems and environments. In: STACS 2011, pp. 615–626 (2011)
25. Kuske, D., Morin, R.: Pomsets for local trace languages. Journal of Automata, Languages and Combinatorics 7(2), 187–224 (2002)
26. Lodaya, K., Weil, P.: Series-parallel languages and the bounded-width property. Theoret. Computer Science 237(1-2), 347–380 (2000)
27. Lorenz, R., Juhás, G., Bergenthum, R., Desel, J., Mauser, S.: Executability of scenarios in Petri nets. Theor. Comput. Sci. 410(12-13), 1190–1216 (2009)
28. Mazurkiewicz, A.W.: Trace Theory. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 279–324. Springer, Heidelberg (1987)
29. Montanari, U., Pistore, M.: Minimal Transition Systems for History-Preserving Bisimulation. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 413–425. Springer, Heidelberg (1997)
30. Morin, R.: On Regular Message Sequence Chart Languages and Relationships to Mazurkiewicz Trace Theory. In: Honsell, F., Miculan, M. (eds.) FOSSACS 2001. LNCS, vol. 2030, pp. 332–346. Springer, Heidelberg (2001)
31. Muscholl, A., Peled, D., Su, Z.: Deciding Properties for Message Sequence Charts. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, pp. 226–242. Springer, Heidelberg (1998)
32. de Oliveira Oliveira, M.: Hasse diagram generators and Petri nets. Fundamenta Informaticae 105(3), 263–289 (2010)
33. Thomas, W.: Finite-state recognizability of graph properties. Theorie des Automates et Applications 172, 147–159 (1992)
34. Vogler, W.: Modular Construction and Partial Order Semantics of Petri Nets. LNCS, vol. 625. Springer, Heidelberg (1992)

# Ogden's Lemma for ET0L Languages

Max Rabkin

School of Computer Science
University of the Witwatersrand
Johannesburg, South Africa
max@cs.wits.ac.za

**Abstract.** We develop a necessary condition for ET0L languages inspired by Ogden's Lemma. Besides being useful for proving that individual languages are not ET0L languages, this result also gives an alternative proof of Ehrenfeucht and Rozenberg's theorem about rare and nonfrequent symbols in ET0L languages.

## 1 Introduction

Perhaps the best known theorems in formal language theory are the pumping lemmas for regular and context-free languages. A more powerful version of the latter – Ogden's Lemma – was proven in [6]. This theorem allows symbols in a word to be designated as marked, and guarantees that the pumping operation will affect the marked symbols. The main use of these theorems is proving that a given language does not belong to the corresponding class.

It is natural to ask whether there are analogous results for larger classes of languages, and in Sect. 4 of this paper we prove such a result for extended table-driven zero-context Lindenmayer (ET0L) languages. Like the pumping lemma for context-free languages, it generates new words in a language from a given word, by repeating subwords. Unlike the results for context-free languages, but like the pumping lemma for random permitting-context languages (Theorem 6 in [5]), the pumping operation can repeat more than two subwords, and can change their order. This makes it more difficult to apply but it seems unavoidable, due to the increased power of ET0L systems.

In Sect. 5 we prove some useful corollaries of our main result, including the theorem of Ehrenfeucht and Rozenberg about rare and nonfrequent symbols in ET0L languages (Theorem 1 in [4]).

There has been previous work on pumping lemmas for L-system languages, which we review briefly in Sect. 3: Beauquier [1] proved a version of Ogden's Lemma for finite-index ET0L languages. Ehrenfeucht and Rozenberg [3] give a pumping lemma for deterministic ET0L languages where only certain words can be pumped. In contrast to these, our result applies to all ET0L languages, and all sufficiently long words.

## 2 Definitions

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$, $\mathbb{N}$ for $\{0, 1, \ldots\}$, and $\mathbb{N}^+$ for $\{1, 2, \ldots\}$.

We let $\#_b(w)$ denote the number of appearances of the symbol $b$ in the word $w$, and let $\#_B(w) = \sum_{b \in B} \#_b(w)$ when $B$ is a set of symbols.

Our definition of an ET0L language differs slightly from the usual: we use a single symbol, rather than a word, as the axiom. This simplifies our work slightly, but has no effect on the class of languages generated (see Theorem 4 below).

**Definition 1.** *An ET0L system is a tuple $G = (V, \Sigma, \mathcal{T}, S)$ where $\Sigma$, the terminal alphabet, is a non-empty subset of the alphabet $V$, $S \in V$ and $\mathcal{T}$ is a collection $\{T_1, \ldots, T_n\}$ of tables. Each table is a set of productions $A \to u$, such that for every $A \in V$ there is a production $A \to u$ in $T_i$ for every $i \in [n]$.*

**Definition 2.** *If $G = (V, \Sigma, \mathcal{T}, S)$ is an ET0L system, with $A_1, A_2, \ldots, A_n \in V$, and $u_1, u_2, \ldots, u_n \in V^*$, then $A_1 A_2 \cdots A_n \Rightarrow u_1 u_2 \ldots u_n$ if there is a table $T \in \mathcal{T}$ such that $(A_i \to u_i) \in T$ for all $i \in [n]$. We write $\Rightarrow^*$ for the reflexive and transitive closure of $\Rightarrow$.*

**Definition 3.** *If $G = (V, \Sigma, \mathcal{T}, S)$ is an ET0L system, then it* generates *the language $L(G) = \{w \in \Sigma^* : S \Rightarrow^* w\}$. Such languages are called* ET0L languages*.*

**Theorem 4.** *If a starting word (axiom) $\omega$ is used instead of the starting symbol $S$ in the definition of ET0L languages, the same class of languages is generated.*

*Proof.* Any ET0L language by our definition is also an ET0L language with an axiom, since we can take $\omega = S$. To see that the converse is true, add a new non-terminal symbol $S$, and add a production $S \to \omega$ to every table. Then $S \Rightarrow \omega$, and since there are no other productions involving $S$, we have $S \Rightarrow^* w$ if and only if $\omega \Rightarrow^* w$. $\square$

**Definition 5.** *If $G = (V, \Sigma, \mathcal{T}, S)$ is an ET0L system and $w_1, w_2, \ldots, w_n \in V^*$, with $S = w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$, then a tree $\tau$ is a* derivation tree of $w_n$ in $G$ *if*

1. *the root of $\tau$ is labelled with $S$, and*
2. *if a symbol $A$ was rewritten to $u$, then the corresponding node has children labelled with the symbols of $u$, in order.*

A *level* refers to the set of nodes at the same depth (and therefore corresponds to one of the words in a derivation). The root is called the first level, its children the second level, and so on. The *maximum out-degree* of a tree is the maximum number of children of any node in the tree

We will have need of several concepts related to marking. A *word with marked symbols* can be defined formally as a pair $(w, M)$, where $M \subseteq [|w|]$: an instance of a symbol in $w$ is called *marked* if its position is in $M$. However, we will not use this notation, preferring for simplicity to identify a marked word with its underlying unmarked word. We extend the concept of marking to nodes of a derivation tree of a marked word $w$ thus:

1. a leaf is marked if it corresponds to a marked symbol in $w$, and
2. a non-leaf node is marked if any of its child nodes are marked.

We will call a node a *branch node* if it has more than one marked child.

The concepts of rare and nonfrequent symbols are used in Theorem 1 of [4], for which we provide an alternative proof.

**Definition 6.** *Let $L$ be a language over $\Sigma$, and $B \subseteq \Sigma$.*

1. *$B$ is called* nonfrequent *if there is a constant $c_B$ such that $\#_B(w) \leq c_B$ for all $w \in L$. Otherwise it is called* frequent.
2. *$B$ is called* rare *if for every $k \in \mathbb{N}^+$, there exists an $n_k \in \mathbb{N}^+$ such that if $\#_B(w) \geq n_k$ then the distance between any two appearances in $w$ of symbols from $B$ is at least $k$.*

When the meaning is clear from context, we will not always disambiguate between a symbol, an instance of a symbol in a word, the node in a derivation tree corresponding to an instance of a symbol, and the subtree rooted at that node.

The remaining definitions are used only in our discussion of previous work.

Since terminal symbols in an ET0L system can be rewritten, the definition of *index* for such a system needs a concept of active symbols.

**Definition 7.** *Let $G = (V, \Sigma, \mathcal{T}, S)$ be an ET0L system. A symbol $A \in V$ is called* active *if there is a production $A \to u$ with $A \neq u$ in some table in $\mathcal{T}$.*

**Definition 8.** *Let $G = (V, \Sigma, \mathcal{T}, S)$ be an ET0L system. $L(G)$ and $G$ are said to have* index $k$ *if every word $w \in L(G)$ has a derivation where no word contains more than $k$ active symbols. If such a $k \in \mathbb{N}$ exists, then $L(G)$ and $G$ are called* finite-index.

**Definition 9.** *An ET0L system $G = (V, \Sigma, \mathcal{T}, S)$ is called* deterministic *(or an* EDT0L *system) if every table $T \in \mathcal{T}$ is a homomorphism: i.e., if $(A \to u_1) \in T$ and $(A \to u_2) \in T$, then $u_1 = u_2$.*

**Definition 10.** *A function $f : \mathbb{N} \to \mathbb{N}$ is called* slow *if $f(n)$ is eventually dominated by $n^\epsilon$ for all $\epsilon > 0$.*

For example, the logarithmic and constant functions are slow.

**Definition 11.** *For any $f : \mathbb{N} \to \mathbb{N}$, a word $w$ is called* $f$-random *if there is no word $u$ with $|u| > f(|w|)$ such that $w = v_1 u v_2 u v_3$.*

## 3 Previous Work

Beauquier [1] proved this Ogden-like theorem about finite-index ET0L languages. Beauquier's result uses finite-index matrix grammars, but these generate the same languages as finite-index ET0L systems [7,8].

**Theorem 12.** *If $L$ is an ET0L language of index $k$, then there exists an $l \in \mathbb{N}$ such that for any $w \in L$ with at least $l$ marked positions, $w$ can be written as $w = u_0v_1u_1v_2u_2 \dots v_nu_n$ with $n \leq k$ such that for some $i$ $(0 \leq i \leq n)$,*

1. *$u_i$ contains a marked position;*
2. *there is a marked position either in both $v_i$ and $u_0v_1u_1 \dots u_{i-1}$, or in both $v_{i+1}$ and $u_{i+1}v_{i+2} \dots v_nu_n$;*
3. *$v_i$ and $v_{i+1}$ together contain at most $l$ marked positions;*
4. *$u_0v_1^tu_1v_2^tu_2 \dots v_n^tu_n \in L$ for all $t \in \mathbb{N}$.*

Ehrenfeucht and Rozenberg [3] proved the following pumping lemma for EDT0L languages.

**Theorem 13.** *If $L$ is an EDT0L language and $f$ is a slow function, then there exists an $l \in \mathbb{N}$ such that for any $f$-random word $w \in L$ with $|w| > l$, $w$ can be written as $w = u_0v_1u_1v_2u_2 \dots v_nu_n$ (with $|v_1v_2 \dots v_n| > 0$) such that $u_0v_1^tu_1v_2^tu_2 \dots v_n^tu_n \in L$ for all $t \in \mathbb{N}$.*

Note that only certain words can be pumped using this theorem, and not all languages contain such words (for example, languages over a singleton alphabet).

## 4 Main Results

The following simple combinatorial lemma on trees will be needed. It basically gives a lower bound on the height of a derivation tree in terms of the length of the derived word, but is modified slightly to be useful for words with marked positions.

**Lemma 14.** *For any $m \in \mathbb{N}$, if a tree with maximum out-degree at most $p$ has more than $p^m$ marked leaves, it must have a path from the root to a leaf with more than $m$ branch nodes.*

*Proof.* We prove this by structural induction on the tree.

A tree consisting of a single leaf has at most $1 = p^0$ marked leaf, and so trivially satisfies the lemma.

Let $\tau$ be a (non-leaf) tree with more than $p^m$ marked leaves, for some $m$, and suppose its children $\tau_1, \tau_2, \dots, \tau_n$ satisfy the lemma ($n \in [p]$). If only a single $\tau_i$ is marked ($i \in [n]$), then $\tau$ has the same number of marked leaves and a path with the same number branch nodes as $\tau_i$, and therefore $\tau$ satisfies the lemma.

Otherwise, the root of $\tau$ is a branch node. One of the $\tau_i$ must have more than $p^m/n \geq p^{m-1}$ marked leaves, and therefore has a path with more than $m-1$ branch nodes. Since the root of $\tau$ is also a branch node, $\tau$ has a path with more than $m$ branch nodes. □

We can now prove our main result.

**Theorem 15.** *If $L$ is an ET0L language, then there exists an $l \in \mathbb{N}$ (which we will call the* threshold *for $L$) such that for any word $w \in L$ with at least $l$ marked positions,*

1. *$w$ can be written as $w = u_1 u_2 \ldots u_n$ and each $u_i$ can be written $u_i = v_{(i,1)} v_{(i,2)} \ldots v_{(i,n_i)}$ (we will denote the set of subscripts of $v$, i.e. $\{(i,j) : i \in [n], j \in [n_i]\}$, by $I$);*
2. *there is a map $\phi : I \to [n]$ such that if each $v_{(i,j)}$ is replaced with $u_{\phi(i,j)}$, then the resulting word is still in $L$, and this process can be applied iteratively to always yield a word in $L$ ;*
3. *if $v_{(i,j)}$ contains a marked position then so does $u_{\phi(i,j)}$ ;*
4. *there is an $(\mathbf{i}, \mathbf{j}) \in I$ such that $\phi(\mathbf{i}, \mathbf{j}) = \mathbf{i}$, and there are at least two marked positions in $v_{(\mathbf{i},\mathbf{j})}$ and at least one in $u_{\mathbf{i}}$ but outside of $v_{(\mathbf{i},\mathbf{j})}$.*

*Proof.* Let $G = (V, \Sigma, \mathcal{T}, S)$ be an ET0L system. Let $p$ be the maximum length of the right-hand side of a rule in $G$, and let $m = |V| \, 4^{|V|}$. Let $w$ be a word in $L(G)$ with at least $p^m + 1$ marked positions.

Consider a derivation tree of $w$ in $G$. Since the tree has more than $p^m$ marked leaves, it must have a path with more than $m$ branch nodes.

There must be a symbol $A \in V$ which appears more than $m/|V| = 4^{|V|}$ times as the label of a branch node on this path. For each of these, consider the set of symbols which appear on the same level in the derivation tree, and the set of *marked* symbols which appear on the same level. Since there are only $4^{|V|}$ distinct such pairs of sets, there must be two branch nodes labelled by $A$, with one an ancestor of the other, with the same pair of sets. Call the ancestor node $A_1$ and the descendant $A_2$, and the words in which they appear $w_1$ and $w_2$, respectively.

The sequence of tables which were applied to $w_1$ to derive $w$ can be applied again to $w_2$. Due to non-determinism, there may be several possible results from this, but one result can be obtained by replacing each subtree corresponding to an instance of a symbol in $w_2$ with a subtree corresponding to an instance of the same symbol in $w_1$. To obtain the desired result, it is necessary to choose $A_1$ when an instance of $A$ is sought, and if there is a marked instance of a symbol, we must choose it. This replacement can be applied as many times as desired. This gives us parts 2 and 3 of the theorem.

Part 4 follows from the fact that $A_2$ is replaced with $A_1$, and both are branch nodes. Set $l = p^m + 1$ to complete the proof. □

We give here a more formal description of the replacement operation in part 2 (which we will call the *pumping operation*). This operation produces the words $w^{(t)}$ for all $t \in \mathbb{N}$, where

$$v^{(0)}_{(i,j)} = v_{(i,j)} \tag{1}$$

$$u^{(t)}_i = v^{(t)}_{(i,1)} v^{(t)}_{(i,2)} \ldots v^{(t)}_{(i,n_i)} \tag{2}$$

$$v^{(t+1)}_{(i,j)} = u^{(t)}_{\phi(i,j)} \tag{3}$$

$$w^{(t)} = u^{(t)}_1 u^{(t)}_2 \ldots u^{(t)}_n \, . \tag{4}$$

Note that $w^{(0)} = w$. We consider an instance of a symbol in $w^{(t+1)}$ to be marked if it was produced by copying a marked symbol in $w^{(t)}$.

It may be interesting to remark that $\{w^{(t)} : t \in \mathbb{N}\}$ is a homomorphic image of a deterministic zero-context Lindenmayer (D0L) language.

This pumping operation is more complicated than that in Theorems 12 and 13. At least some complication is necessary: those operations generate a sequence of words whose lengths form an infinite arithmetic progression, but there are infinite ET0L languages which do not contain such sequences.

There is a simpler form of Theorem 15 which may nevertheless sometimes be useful. Their relationship is analogous to the relationship between Ogden's Lemma and the pumping lemma for context-free languages.

**Corollary 16.** *If $L$ is an ET0L language, then there exists an $l \in \mathbb{N}$ such that for any word $w \in L$ with $|w| \geq l$,*

1. *$w$ can be written as $w = u_1u_2 \ldots u_n$ and each $u_i$ can be written $u_i = v_{(i,1)}v_{(i,2)} \ldots v_{(i,n_i)}$ (we will again use $I$ for the set of subscripts of $v$);*
2. *there is a map $\phi : I \to [n]$ such that if each $v_{(i,j)}$ is replaced with $u_{\phi(i,j)}$, then the resulting word is still in $L$, and this process can be applied iteratively to always yield a word in $L$ ;*
3. *there is an $(\mathbf{i}, \mathbf{j}) \in I$ such that $\phi(\mathbf{i}, \mathbf{j}) = \mathbf{i}$, and $v_{(\mathbf{i},\mathbf{j})}$ is a strict subword of $u_\mathbf{i}$.*

*Proof.* This follows directly from Theorem 15 if we mark all positions in $w$. □

We will now see how Theorem 15 and its proof apply to an example.

*Example 17.* The set $\{a^{F_n} : n \in \mathbb{N}^+\}$, where $F_n$ denotes the $n$th Fibonacci number, is an ET0L language generated by the system $(V, \Sigma, \mathcal{T}, S)$ where $V = \{Q, S, a\}$, $\Sigma = \{a\}$, and $\mathcal{T} = \{\{Q \to QS, S \to Q, a \to a\}, \{Q \to a, S \to a, a \to a\}\}$.

A derivation tree in this system for $a^5$ is given in Fig. 1. For simplicity, we will consider the case where all symbols are marked, as in Corollary 16.

Observe that the third and fourth level of the tree contain the same set of symbols – $\{Q, S\}$ – and since all symbols are marked, also the same set of marked symbols. Furthermore, the $Q$ in the third level is an ancestor of the leftmost $Q$ in the fourth level, and both are branch nodes, so the pumping operation can be applied to these two levels of the tree.

This leads us to break the string up as $\overbrace{\overline{aa}\,\overline{a}}\overbrace{\overline{aa}}$, where the outer brackets delimit the subwords $u_1$ and $u_2$, and the inner brackets, $v_{(1,1)}$, $v_{(1,2)}$ and $v_{(2,1)}$, and to set $\phi(1,1) = \phi(2,1) = 1$ and $\phi(1,2) = 2$. Applying the pumping operation yields $w^{(1)} = \overbrace{\overline{aaa}\,\overline{aa}}\overbrace{\overline{aaa}}$, which is indeed a string in the language. Applying it again yields $w^{(2)} = \overbrace{\overline{aaaaa}\,\overline{aaa}}\overbrace{\overline{aaaaa}}$, and so on.

Theorem 15 is a necessary condition for a language to be ET0L, but it is not sufficient. In fact, as the following example shows, a language which satisfies it may even be uncomputable, while all ET0L languages are computable [2, Table 0.3.1].

*Example 18.* Let $M_1, M_2, \ldots$ be an enumeration of the set of Turing machines, and let $L = \{a^{2^k(2i+1)} : M_i \text{ halts}, k \in \mathbb{N}\}$. This language is not computable, and is therefore not ET0L, but it satisfies the conditions of Theorem 15.

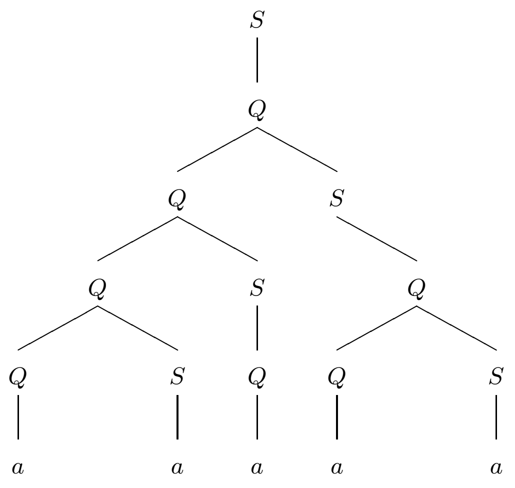


**Fig. 1.** An example ET0L derivation

The halting problem is reducible to $L$, since $a^{2i+1} \in L$ if and only if $M_i$ halts. Thus $L$ is uncomputable.

Now we will show that it satisfies Theorem 15 with threshold $l = 3$. Let $w = a^{2^k(2i+1)}$ be a word in $L$ with at least three marked positions, where $k \in \mathbb{N}$ and $M_i$ halts. Let $u_1 = w$ and let $v_{(1,1)}v_{(1,2)} = w$ in such a way that $v_{(1,1)}$ contains at least two marked positions and $v_{(1,2)}$ contains at least one. Let $\phi(1,1) = \phi(1,2) = 1$. Then applying the pumping operation yields the words $a^{2^{k+1}(2i+1)}, a^{2^{k+2}(2i+1)}, \ldots$, which are all in $L$, and all the other conditions of Theorem 15 are satisfied.

## 5 Applications

We will now prove two facts about the pumping operation which will help in using Theorem 15 to prove that certain languages are not ET0L languages.

The pumping operation can initially reduce the number of marked symbols by replacing subwords with many marked symbols by subwords with few. However, eventually the number of marked symbols must increase, as the following shows.

**Corollary 19.** *If $L$ is an ET0L language with threshold $l$, and $w \in L$ has at least $l$ marked symbols, then the number of marked symbols in $w^{(t)}$ tends to infinity, where $w^{(t)}$ is the result of applying the pumping operation $t$ times. Specifically, $w^{(t)}$ contains at least $t + 2$ marked symbols for all $t \in \mathbb{N}$.*

*Proof.* Let $u_i^{(t)}$ and $v_{(i,j)}^{(t)}$ be as in (1)–(4), and $\mathbf{i}$ and $\mathbf{j}$ as in part 4 of Theorem 15.

By part 4 of Theorem 15, $v_{(\mathbf{i},\mathbf{j})}^{(0)}$ contains at least two marked symbols.

Suppose $v^{(t)}_{(\mathbf{i},\mathbf{j})}$ contains at least $t+2$ marked symbols. Then $v^{(t+1)}_{(\mathbf{i},\mathbf{j})} = u^{(t)}_{\mathbf{i}}$; this has $v^{(t)}_{(\mathbf{i},\mathbf{j})}$ as a subword, which contributes $t+2$ marked symbols. However, $u_{\mathbf{i}}$ also contains a marked symbol outside of $v_{(\mathbf{i},\mathbf{j})}$, and since subwords containing a marked symbol are replaced with other such subwords, this property is inherited: $u^{(t)}_{\mathbf{i}}$ has a marked symbol outside of $v^{(t)}_{(\mathbf{i},\mathbf{j})}$. This contributes another marked symbol, so in total there are at least $t+3$ marked symbols in $v^{(t+1)}_{(\mathbf{i},\mathbf{j})}$.

By induction, we conclude that $w^{(t)}$ contains at least $t+2$ marked symbols for all $t \in \mathbb{N}$. □

On the other hand, the number of symbols cannot grow superexponentially by this pumping operation:

**Corollary 20.** *If $L$ is an ET0L language with threshold $l$, and $w \in L$ is at least $l$ symbols long, then $|w^{(t)}| \le |I|^t |w|$, where $w^{(t)}$ is the result of applying the pumping operation $t$ times and $I$ is the set defined in part 1 of Theorem 15.*

*Proof.* Let $u^{(t)}_i$ and $v^{(t)}_{(i,j)}$ be as in (1)–(4).

For all $(i,j) \in I$ and all $t$, we have $|v^{(t+1)}_{(i,j)}| = |u^{(t)}_{\phi(i,j)}| \le |w^{(t)}|$.

Since $|w^{(t+1)}| = \sum_{(i,j)\in I} |v^{(t+1)}_{(i,j)}|$, we get $|w^{(t+1)}| \le |I|\,|w^{(t)}|$. □

Although Corollary 19 relates to marked symbols and Corollary 20 to the total number of symbols, each applies to both, since the number of marked symbols is always less than or equal to the total number of symbols.

Corollaries 19 and 20 together with Theorem 15 can be useful in proving that certain languages are not ET0L languages.

*Example 21.* We wish to prove that $L = \{a^n b^m : n \in \mathbb{N}, m \ge 2^{2^n}\}$ is not an ET0L language. Suppose it is. Then, let $l$ be the number in Theorem 15, $m = 2^{2^l}$, and $w = a^l b^m$ with all the $a$'s and none of the $b$'s marked. Theorem 15 applies to $w$. As before, we will denote by $w^{(t)}$ the word obtained by applying the pumping operation $t$ times.

By Corollary 19, $\#_a\left(w^{(t)}\right) \ge t+2$; on the other hand, $\#_b\left(w^{(t)}\right) \le |w^{(t)}| \le |I|^t |w|$ by Corollary 20. Since $|I|^t |w|$ is eventually dominated by $2^{2^{t+2}}$, we find that $w^{(t)} \notin L$ for large enough $t$, which contradicts Theorem 15; thus $L$ cannot be an ET0L language. □

If all positions were marked, the pumping operation could increase the number of $b$'s while leaving the $a$'s, and so generating strings in $L$. So the marking aspect is really necessary.

A similar argument can be used if $2^{2^n}$ is replaced with any superexponential function.

The following theorem by Ehrenfeucht and Rozenberg [4] can be deduced from Theorem 15 without making any additional reference to the structure of ET0L derivations.

**Theorem 22.** *Let $L$ be an ET0L language over $\Sigma$, with $B$ a non-empty subset of $\Sigma$. If $B$ is rare in $L$, then $B$ is nonfrequent in $L$.*

*Proof.* Let $l$ be the number for $L$ from Theorem 15. Suppose $B$ is frequent in $L$. Then there must be a word $w \in L$ with more than $l$ symbols from $B$. If we mark them all, Theorem 15 applies. Again we will denote by $w^{(t)}$ the result of pumping this word $t$ times.

By Corollary 19, $w^{(t)}$ contains at least $t + 2$ symbols from $B$. However, by part 4 of Theorem 15, the subword $v_{(\mathbf{i},\mathbf{j})}$ contains two marked symbols, and this subword appears in all $w^{(t)}$, so these two symbols are a fixed distance apart. So $B$ is not rare in $L$. □

This can be used to prove that, for example, $\{(ga^k)^k : k \in \mathbb{N}^+\}$ is not an ET0L language. Theorem 15 is, however, stronger than this one: the language of Example 21 has no rare set of symbols, so Theorem 22 cannot show that it is not an ET0L language.

## 6 Concluding Remarks

We have developed an analogue of Ogden's Lemma for ET0L languages, which can be used to prove that certain languages do not belong to this class. We also used it to give a straightforward proof of a known result about these languages.

It would be interesting to investigate whether our theorem can be used to prove other results in the theory of ET0L languages. We are also interested to see if one can prove a marking variant, similar to Theorem 15, of Ewert and Van der Walt's pumping lemma for random permitting-context languages [5] and shrinking lemma for random forbidding-context languages (a superset of the ET0L languages) [9].

**Acknowledgements.** The author is grateful to Sigrid Ewert and the anonymous reviewers for their comments and suggestions, and to Bertin Zinsou for translation assistance.

This work is based upon research supported by the National Research Foundation of South Africa. Any opinion, findings and conclusions or recommendations expressed in this material are those of the author and therefore the NRF do not accept any liability thereto.

## References

1. Beauquier, J.: Deux familles de langages incomparables. Information and Control 43(2), 101–122 (1979)
2. Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. EATCS Monographs on Theoretical Computer Science. Springer, Berlin (1989)
3. Ehrenfeucht, A., Rozenberg, G.: A pumping theorem for EDT0L languages. Tech. Rep. CU-CS-047-74, University of Colorado (1974)

4. Ehrenfeucht, A., Rozenberg, G.: On proving that certain languages are not ETOL. Acta Informatica 6, 407–415 (1976)
5. Ewert, S., van der Walt, A.: A pumping lemma for random permitting context languages. Theoretical Computer Science 270(1-2), 959–967 (2002)
6. Ogden, W.: A helpful result for proving inherent ambiguity. Mathematical Systems Theory 2, 191–194 (1968)
7. Păun, G.: Some further remarks on the family of finite index matrix languages. RAIRO Informatique Théorique 13(3), 289–297 (1979)
8. Rozenberg, G.: More on ET0L systems versus random context grammars. Information Processing Letters 5(4), 102–106 (1976)
9. van der Walt, A., Ewert, S.: A shrinking lemma for random forbidding context languages. Theoretical Computer Science 237(1-2), 149–158 (2000)

# Patterns with Bounded Treewidth

Daniel Reidenbach and Markus L. Schmid [⋆]

Department of Computer Science, Loughborough University,
Loughborough, Leicestershire, LE11 3TU, UK
{D.Reidenbach,M.Schmid}@lboro.ac.uk

**Abstract.** We show that any parameter of patterns that is an upper bound for the treewidth of appropriate encodings of patterns as relational structures, if restricted to a constant, allows the membership problem for pattern languages to be solved in polynomial time. Furthermore, we identify a new such parameter, called the scope coincidence degree.

**Keywords:** Pattern Languages, Membership Problem, Treewidth, Extended Regular Expressions.

## 1 Introduction

A *pattern* $\alpha$ is a finite string that consists of *variables* and *terminal symbols* (taken from a fixed alphabet $\Sigma$), and its language is the set of all words that can be derived from $\alpha$ when substituting arbitrary words over $\Sigma$ for the variables. For example, the language $L$ generated by the pattern $\alpha := x_1 \mathtt{a} x_2 \mathtt{b} x_1$ (where $x_1, x_2$ are variables and $\mathtt{a}, \mathtt{b}$ are terminal symbols) consists of all words with an arbitrary prefix $u$, followed by the letter $\mathtt{a}$, an arbitrary word $v$, the letter $\mathtt{b}$ and a suffix that equals the prefix $u$. Thus, $w_1 := \mathtt{aaabbaa}$ is contained in $L$, whereas $w_2 := \mathtt{baaba}$ is not.

Patterns provide a compact and natural way to describe formal languages. In their original definition given by Angluin [1] variables can only be substituted by non-empty words; hence, the term *nonerasing pattern languages* (or, for short, *NE-pattern languages*) is used. *Extended* or *erasing pattern languages* (or, for short, *E-pattern languages*) where variables can also be substituted by the empty word have been introduced by Shinohara [18]. The original motivation for pattern languages (cf. Angluin [1]) is derived from *inductive inference*, i. e., the task of inferring a pattern from any given sequence of all words in its pattern language, for which numerous results can be found in the literature (see, e. g., Angluin [1], Shinohara [18], Lange and Wiehagen [8], Rossmanith and Zeugmann [16], Reidenbach [11,12] and, for a survey, Ng and Shinohara [10]). On the other hand, due to their simple definition, pattern languages have connections to many other areas of theoretical computer science and their general properties have been investigated in various contexts (for a survey, see, e. g., Mateescu and A. Salomaa [9]). For example, there exist several versions of regular expressions

⋆ Corresponding author.

that are extended in such a way that pattern languages can be defined (see, e. g., Bordihn et al. [3]). The problem to decide for a given word $w$ and a pattern $\alpha$ whether or not the variables of $\alpha$ can be substituted in such a way that $w$ is obtained, i. e., the *membership problem* for pattern languages, has been shown to be NP-complete by Angluin [1].

Besides the theoretical importance of pattern languages, the concept of patterns also finds practical application in so-called *extended regular expressions with backreferences* (*REGEX* for short) (see, e. g., Câmpeanu et al. [5]). REGEX can roughly be considered as classical regular expressions that are equipped with the possibility to define backreferences, i. e., to require factors to be repeated at several defined positions in the word; hence, backreferences correspond to the variables in patterns. While backreferences dramatically increase the expressive power of classical regular expressions, they are also responsible for the membership problem of this language class to become NP-complete. This is particularly worth mentioning as today's text editors and programming languages (such as Perl, Python, Java, etc.) all provide so-called *REGEX engines* that compute the solution to the membership problem for any language given by a REGEX and an arbitrary string. Hence, despite its theoretical intractability, algorithms that perform the matchtest for REGEX are a practical reality. While pattern languages merely describe a proper subset of REGEX languages, they cover what is computationally hard, i. e., the concept of backreferences. Hence, investigating the membership problem for pattern languages helps to improve algorithms solving the matchtest for extended regular expressions with backreferences.

Our main research task is to identify parameters of patterns that, if restricted to a constant, allow a polynomial time membership problem. The benefit of finding such parameters is twofold. Firstly, we can learn what properties of a pattern are actually responsible for the complexity of the membership problem, i. e., we achieve a refined complexity analysis of this problem. Secondly, restricting these parameters is likely to lead to improved algorithms for the membership problem of pattern languages. The first such parameter that comes to mind is the number of different variables in a pattern. Its restriction constitutes a trivial way to obtain a polynomial time membership problem, since the brute force algorithm that simply enumerates all possibilities to substitute the variables by terminal words in order to check whether the input word can be obtained is exponential in the number of variables. Nevertheless, the number of variables is a central parameter of patterns and important results about the learnability of pattern languages (see Angluin [1] and Reischuk and Zeugmann [15]) as well as recent results about the inclusion problem of pattern languages (see Bremer and Freydenberger [4]) are concerned with patterns with a restricted number of variables. The membership problem for pattern languages given by patterns with only one occurrence per variable (introduced by Shinohara [18]) is also solvable in polynomial time, simply because these patterns describe regular languages; hence, they are called *regular* patterns.

The arguably first nontrivial restriction of patterns that allow a polynomial time membership problem are Shinohara's *non cross* patterns [19], i. e., patterns where between any two occurrences of the same variable $x$ no other variable different from $x$ occurs. However, this result does not provide a structural parameter of patterns that can be considered to contribute to the complexity of the membership problem. Recently, in [14], Shinohara's result has been extended to an infinite hierarchy of classes of pattern languages with a polynomial time membership problem. The idea in [14] is to restrict a rather subtle parameter, namely the *distance* several occurrences of any variable $x$ may have in a pattern (i. e., the maximum number of different variables separating any two consecutive occurrences of $x$). This parameter is called the *variable distance* vd of a pattern $\alpha$, and in [14] it is demonstrated that the membership problem is solvable in time $O(|\alpha|^3 \times |w|^{(\mathrm{vd}(\alpha)+4)})$, so it is exponential only in the variable distance.

In this work, we approach the problem of identifying such parameters in a novel and quite general way. More precisely, we encode patterns and words as relational structures and, thus, reduce the membership problem to the homomorphism problem for relational structures. Our main result is that any parameter of patterns that is an upper bound for the treewidth of the corresponding relational structures, if restricted to a constant, allows the membership problem to be solved in polynomial time. In this new framework, we can restate the known results about the complexity of the membership problem mentioned above, as well as identifying a new parameter that is stronger than the variable distance. Therefore, we provide a convenient way to treat the membership problem for pattern languages, which, as shall be pointed out by our results, has still potential for further improvements.

Note that, due to space constraints, all proofs are omitted.

## 2 Preliminaries

Let $\mathbb{N} := \{0, 1, 2, 3, \ldots\}$. For an arbitrary alphabet $A$, a *string* (*over* $A$) is a finite sequence of symbols from $A$, and $\varepsilon$ stands for the *empty string*. The notation $A^+$ denotes the set of all nonempty strings over $A$, and $A^* := A^+ \cup \{\varepsilon\}$. For the *concatenation* of two strings $w_1, w_2$ we write $w_1 \cdot w_2$ or simply $w_1 w_2$. We say that a string $v \in A^*$ is a *factor* of a string $w \in A^*$ if there are $u_1, u_2 \in A^*$ such that $w = u_1 \cdot v \cdot u_2$. The notation $|K|$ stands for the size of a set $K$ or the length of a string $K$. The term $\mathrm{alph}(w)$ denotes the set of all symbols occurring in $w$. If we wish to refer to the symbol at a certain position $j$, $1 \leq j \leq n$, in a string $w = \mathtt{a}_1 \cdot \mathtt{a}_2 \cdot \cdots \cdot \mathtt{a}_n$, $\mathtt{a}_i \in A$, $1 \leq i \leq n$, we use $w[j] := \mathtt{a}_j$. Furthermore, for each $j, j'$, $1 \leq j < j' \leq |w|$, let $w[j, j'] := \mathtt{a}_j \cdot \mathtt{a}_{j+1} \cdot \cdots \cdot \mathtt{a}_{j'}$ and $w[j, -] := w[j, |w|]$. If $j > |w|$, we define $w[j, -] = \varepsilon$.

### Pattern Languages and the Scope Coincidence Degree

For any alphabets $A, B$, a *morphism* is a function $h : A^* \to B^*$ that satisfies $h(vw) = h(v)h(w)$ for all $v, w \in A^*$; $h$ is said to be *nonerasing* if and only if, for every $a \in A$, $h(a) \neq \varepsilon$. Let $\Sigma$ be a finite alphabet of so-called *terminal symbols* and $X$ a countably infinite set of *variables* with $\Sigma \cap X = \emptyset$. We normally assume

$X := \{x_1, x_2, x_3, \ldots\}$. A *pattern* is a nonempty string over $\Sigma \cup X$, a *terminal-free pattern* is a nonempty string over $X$ and a *word* is a string over $\Sigma$. For any pattern $\alpha$, we refer to the set of variables in $\alpha$ as $\mathrm{var}(\alpha)$.

A morphism $h : (\Sigma \cup X)^* \to \Sigma^*$ is called a *substitution* if $h(a) = a$ for every $a \in \Sigma$. We define the *E-pattern language* of a pattern $\alpha$ by $L_{\mathrm{E},\Sigma}(\alpha) := \{h(\alpha) \mid h : (\Sigma \cup X)^* \to \Sigma^* \text{ is a substitution}\}$. The *NE-pattern language* $L_{\mathrm{NE},\Sigma}(\alpha)$ of $\alpha$ is defined analogously, just with respect to nonerasing substitutions. Since in our work the impact of the choice of the alphabet $\Sigma$ is negligible, we mostly denote pattern languages by $L_{\mathrm{E}}(\alpha)$ and $L_{\mathrm{NE}}(\alpha)$.

The problem to decide for a given pattern $\alpha$ and a given word $w \in \Sigma^*$ whether $w \in L_{\mathrm{E}}(\alpha)$ (or $w \in L_{\mathrm{NE}}(\alpha)$) is called the *membership problem for* E-*pattern languages* (or NE-*pattern languages*, respectively). For every class $C \subseteq (\Sigma \cup X)^*$ and every $Z \in \{\mathrm{E}, \mathrm{NE}\}$, $Z$-PATMem$(C)$ denotes the membership problem for $Z$-pattern languages where the patterns are restricted to the class $C$.

The concept of the scope coincidence degree has already been introduced in [13]. However, here we shall define it in a slightly different (yet equivalent) way. Let $\alpha$ be a pattern. For every $y \in \mathrm{var}(\alpha)$, the *scope of $y$ in $\alpha$* is defined by $\mathrm{sc}_\alpha(y) := \{i, i+1, \ldots, j\}$, where $i$ is the leftmost and $j$ the rightmost occurrence of $y$ in $\alpha$. The scopes of $y_1, y_2, \ldots, y_k \in \mathrm{var}(\alpha)$ *coincide in* $\alpha$ if and only if $\bigcap_{1 \leq i \leq k} \mathrm{sc}_\alpha(y_i) \neq \emptyset$. Finally, the *scope coincidence degree* of $\alpha$ ($\mathrm{scd}(\alpha)$) is the maximum number of variables in $\alpha$ such that their scopes coincide. Let $\Sigma := \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and let the pattern $\beta \in (\Sigma \cup X)^*$ be given by $\beta := x_1 \mathtt{b} x_2 \mathtt{a} x_1 x_3 x_2 \mathtt{ab} x_3 x_4 x_2 x_4 x_5 \mathtt{bcb} x_1 x_4 x_5$. It is easy to see that each set $\{x_1, x_2, x_3\}$, $\{x_1, x_2, x_4\}$ and $\{x_1, x_4, x_5\}$ contain variables the scopes of which coincide, but there does not exist a set of more than 3 variables with the same property. Hence, $\mathrm{scd}(\beta) = 3$. It is straightforward to see that the scope coincidence degree can be computed in time linear in the length of the pattern.

## Relational Structures, Treewidth and Homomorphism Problem

For the sake of completeness, we repeat the following standard definitions very briefly, and for a comprehensive reference, the reader is referred to Chapters 4, 11 and 13 of Flum and Grohe [6].

A *(relational) vocabulary* $\tau$ is a finite set of relation symbols. Every relation symbol $R \in \tau$ has an *arity* $\mathrm{arity}(R) \geq 1$. A *$\tau$-structure* (or simply *structure*), comprises a finite set $A$ called the *universe* and, for every $R \in \tau$, an *interpretation* $R^{\mathcal{A}} \subseteq A^{\mathrm{arity}(R)}$. For example, every graph can be given as a relational structure over a vocabulary with one binary relation symbol representing the edges. Let $\mathcal{A}$ and $\mathcal{B}$ be structures of the same vocabulary $\tau$ with universes $A$ and $B$, respectively. A *homomorphism* from $\mathcal{A}$ to $\mathcal{B}$ is a mapping $h : A \to B$ such that for all $R \in \tau$, of arity $r$, and for all $a = (a_1, a_2, \ldots, a_r) \in A^r$, $a \in R^{\mathcal{A}}$ implies $h(a) \in R^{\mathcal{B}}$, where $h(a) = (h(a_1), h(a_2), \ldots, h_r(a_r))$.

Next, we introduce the concept of a *treewidth* of a graph $\mathcal{G}$, denoted by $\mathrm{tw}(\mathcal{G})$. We omit the standard definition of the treewidth, that makes use of the concept of tree decompositions of graphs (see, e. g., Bodlaender [2]). Instead, we apply an alternative characterisation in terms of a game due to Seymour and Thomas [17].

In the *robber-cops-game* (called *jump searching* in [17]), a number of cops try to catch a robber on a graph. Let $\mathcal{G} := (V, E)$ be a graph. A *position (of the robber-cops-game)* is a pair $(C, R)$, where $C \subseteq V$ and $R$ is a connected subgraph of $\mathcal{G}$ that does not contain any vertex of $C$. The set $C$ contains the vertices currently occupied by cops. The set $R$, on the other hand, is the position of the robber. Since the robber can move with infinite speed we can interpret it to occupy all vertices of the connected subgraph $R$ at the same time. The initial position of the game is $(C_0, R_0)$, where $C_0 = \emptyset$ and $R_0$ is some connected subgraph chosen by the robber. At the start of the $i^{\text{th}}$ step of the game we have position $(C_{i-1}, R_{i-1})$. Now all the cops are removed from the graph and then again placed on some vertices $C_i \subseteq V$. After that, the robber chooses (if possible) a new connected subgraph $R_i$ that does not contain any vertex from $C_i$ and *touches* $R_{i-1}$, i. e., $R_{i-1}$ and $R_i$ have a common vertex or an edge connects a vertex of $R_{i-1}$ with a vertex of $R_i$. If in any step $i$ of the robber cops game, the cops could choose a set of vertices $C_i$ such that there does not exists a position $(C_i, R_i)$, i. e., for every possible connected subgraph $R_i$ that touches $R_{i-1}$, $R_i \subseteq C_i$ is satisfied, then the cops win the robber-cops-game. We say that $k \in \mathbb{N}$ *cops can catch a robber on* $\mathcal{G}$ if the robber-cops-game can be won by the cops such that, for every position $(C_i, R_i)$ in the game, $|C_i| \leq k$.

For example, one cop is not enough to catch a robber even on a single path, since the robber can always outrun the cop as soon as it changes to another vertex. Two cops can catch a robber on a path and also on arbitrary trees. To catch a robber on a simple circle three cops are required: one is permanently placed on some vertex, which turns the cycle in a path, and then the other two can corner the robber in one of the two dead ends. The next theorem establishes the relation between the robber-cops-game and the treewidth of a graph.

**Theorem 1 (Seymour and Thomas [17]).** *Let $\mathcal{G}$ be an arbitrary graph. Then $k \in \mathbb{N}$ cops can catch a robber on $\mathcal{G}$ if and only if* $\text{tw}(\mathcal{G}) \leq k - 1$.

In order to define the treewidth of relational structures, we need the concept of the *Gaifman graph* of a $\tau$-structure $\mathcal{A}$, which is the graph that has the universe $A$ of $\mathcal{A}$ as vertices and two vertices are connected if and only if they occur together in some relation (see Chapter 11 of Flum and Grohe [6]). Now we can state the definition of the treewidth that shall be used for our results:

**Definition 2.** *The treewidth of a structure equals $k-1$, where $k$ is the minimum number of cops that are sufficient to catch a robber on its Gaifman graph.*

The *homomorphism problem HOM* is the problem to decide, for given structures $\mathcal{A}$ and $\mathcal{B}$, whether there exists a homomorphism from $\mathcal{A}$ to $\mathcal{B}$. For any set of structures $C$, by HOM($C$) we denote the homomorphism problem, where the left hand structure is restricted to be from $C$. If $C$ is a class of structures with bounded treewidth, then HOM($C$) can be solved in polynomial time. This is a classical result that has been first achieved in terms of *constraint satisfaction problems* by Freuder [7] (see also Chapter 13 of Flum and Grohe [6]).

**Theorem 3 (Freuder [7]).** *Let $C$ be a set of structures with bounded treewidth. HOM($C$) is solvable in polynomial time.*

## 3 Patterns and Words as Relational Structures

In this section, we introduce a way to represent patterns and terminal words as relational structures. Our overall goal is to reduce the membership problem for pattern languages to the homomorphism problem for relational structures.

Representing words as relational structures is a common technique when mathematical logic is applied to language theory (see, e. g., Thomas [20] for a survey). However, our representations of patterns and words by structures differ from the standard technique, since our approach is tailored to the homomorphism problem of structures and, furthermore, we want to exploit the treewidth.

In order to encode patterns and terminal words, i. e., an instance of the membership problem for pattern languages, we use the relational vocabulary $\tau_\Sigma := \{E, S, L, R\} \cup \{D_a \mid a \in \Sigma\}$, where $E, S$ are binary relations and $L, R, D_a$, $a \in \Sigma$, are unary relations. The vocabulary depends on $\Sigma$, the alphabet under consideration. In order to represent a pattern $\alpha$ by a $\tau_\Sigma$-structure, we interpret the set of positions of $\alpha$ as the universe. The roles of $S$, $L$, $R$ and $D_a$, $a \in \Sigma$, are straightforward: $S$ relates adjacent positions, $L$ and $R$ are singletons that contain the leftmost and rightmost position, respectively, and, for every $a \in \Sigma$, the relation $D_a$ contains the positions in $\alpha$ where the terminal symbol $a$ occurrs. For the encoding of the variables, we do not explicitly store their positions in the pattern, which seems impossible, since the number of different variables can be arbitrarily large. Instead, we use the relation $E$ in order to record pairs of positions where the same variable occurs and, furthermore, this is done in a "sparse" way. More precisely, the relation $E$ relates *some* positions with the same variable, i. e., positions $i$, $j$ with $\alpha[i] = \alpha[j]$, in such a way that the symmetric transitive closure of $E$ contains *all* pairs $(i, j)$ with $\alpha[i] = \alpha[j]$ and $\alpha[i] \in X$. This way of interpreting relation $E$ is crucial for our results.

We now state the formal definition and shall illustrate it afterwards.

**Definition 4.** *Let $\alpha$ be a pattern and let $\mathcal{A}_\alpha$ be a $\tau_\Sigma$-structure. $\mathcal{A}_\alpha$ is an $\alpha$-*structure *if it has universe $P_\alpha := \{1, 2, \ldots, |\alpha|\}$ and $S^{\mathcal{A}_\alpha} := \{(i, i+1) \mid 1 \leq i \leq |\alpha| - 1\}$, $L^{\mathcal{A}_\alpha} := \{1\}$, $R^{\mathcal{A}_\alpha} := \{|\alpha|\}$, for every $a \in \Sigma$, $D_a^{\mathcal{A}_\alpha} := \{i \mid \alpha[i] = a\}$, and $E^{\mathcal{A}_\alpha}$ is such that, for all $i, j \in P_\alpha$,*

- *$(i, j) \in E^{\mathcal{A}_\alpha}$ implies $\alpha[i] = \alpha[j]$ and $i \neq j$,*
- *$\alpha[i] = \alpha[j]$ implies that $(i, j)$ is in the symmetric transitive closure of $E^{\mathcal{A}_\alpha}$.*

Since $\tau_\Sigma$ contains only unary and binary relation symbols, it is straightforward to derive the Gaifman graph from an $\alpha$-structure, which is simply a graph with two different kinds of edges due to $S^{\mathcal{A}_\alpha}$ and $E^{\mathcal{A}_\alpha}$. Hence, we shall switch between these two models at our convenience without explicitly mentioning it. In the previous definition, the universe as well as the interpretations for the relation symbols $S$, $L$, $R$ and $D_a$, $a \in \Sigma$, are uniquely defined for a fixed pattern $\alpha$, while there are several possibilities to define an interpretation of $E$. Intuitively, a valid interpretation of $E$ is created by connecting different occurrences of the same variable by edges in such a way that all the occurrences of some variable describe a connected component. The simplest way to do this is to add an edge between *every two* occurrences of the same variable, i. e., $E^{\mathcal{A}_\alpha} := \{(i, j) \mid \alpha[i] = \alpha[j]\}$. However, we shall see that for our results the interpretation of $E$ is crucial and using the one just mentioned is not advisable. Another example of a valid interpretation of $E$ is the following one. For every $x \in \text{var}(\alpha)$, let $l_x$ be the leftmost occurrence of $x$ in $\alpha$. Defining $E^{\mathcal{A}_\alpha} := \bigcup_{x \in \text{var}(\alpha)} \{(l_x, i) \mid l_x < i \leq |\alpha|, \alpha[i] = x\}$ yields another possible $\alpha$-structure.

Next, we define a canonical $\alpha$-structure, i. e., the interpretation of $E$ is such that every occurrence of a variable $x$ at position $i$ is connected to the next occurrence of $x$ to the right of position $i$.

**Definition 5.** *Let $\alpha$ be a pattern. The* standard $\alpha$-structure ($\mathcal{A}^s_\alpha$) *is the $\alpha$-structure where $E^{\mathcal{A}^s_\alpha} := \{(i, j) \mid 1 \leq i < j \leq |\alpha|, \exists\, x \in X$ such that $x = \alpha[i] = \alpha[j]$ and $\alpha[k] \neq x, i < k < j\}$.*

As an example, we consider the standard $\alpha$-structure $\mathcal{A}^s_\alpha$ for the pattern $\alpha := x_1 \cdot \mathtt{a} \cdot \mathtt{b} \cdot x_1 \cdot \mathtt{b} \cdot x_2 \cdot \mathtt{a} \cdot x_1 \cdot x_2 \cdot x_1$. The universe of $\mathcal{A}^s_\alpha$ is $P_\alpha = \{1, 2, \ldots, 10\}$ and the relations are interpreted in the following way. $S^{\mathcal{A}^s_\alpha} = \{(1, 2), (2, 3), \ldots, (9, 10)\}$, $L^{\mathcal{A}^s_\alpha} = \{1\}$, $R^{\mathcal{A}^s_\alpha} = \{10\}$, $D_{\mathtt{a}}^{\mathcal{A}^s_\alpha} = \{2, 7\}$, $D_{\mathtt{b}}^{\mathcal{A}^s_\alpha} = \{3, 5\}$ and, finally, $E^{\mathcal{A}^s_\alpha} = \{(1, 4), (4, 8), (6, 9), (8, 10)\}$.

We continue with representing words over the terminal alphabet $\Sigma$ as $\tau_\Sigma$-structures. We recall that it is our goal to represent the membership problem for pattern languages as homomorphism problem for relational structures. Hence, the way we represent terminal words by $\tau_\Sigma$-structures must cater for this purpose. Furthermore, we have to distinguish between the E case and the NE case. We first introduce the NE case and shall afterwards point out how to extend the constructions for the E case. We choose the universe to be the set of all possible factors of $w$, where these factors are represented by their unique start and end positions in $w$; thus, two factors that are equal but occur at different positions in $w$ are different elements of the universe. The interpretation of $L$ contains all prefixes and the interpretation of $R$ contains all suffixes of $w$. The interpretation of $S$, which for patterns contains pairs of adjacent variables, contains now pairs of adjacent (non-overlapping) factors of $w$. The relation $E$ is interpreted such that it contains *all* pairs of factors that are equal and non-overlapping. Finally, for every $a \in \Sigma$, $D_a$ contains all factors of length one that equal $a$. This is necessary for the possible terminal symbols in the pattern.

For the E case, the empty factors of $w$ need to be represented as well. To this end, for every $i$, $0 \leq i \leq |w|$, we add an element $i_\varepsilon$ to the universe denoting the empty factor between positions $i$ and $i + 1$ in $w$. The interpretations of $S$ and $R$ are extended to contain the empty prefix and the empty suffix, respectively, and relation $S$ is extended to relate non-empty factors to adjacent empty factors and, in addition, each empty factor is also related to itself. Next, we formally define this construction for the NE case and its extension to the E case.

**Definition 6.** *Let $w \in \Sigma^*$ be a terminal word. The* standard-NE-$w$-structure ($\text{NE}-\mathcal{A}^{\text{s}}_w$) *with universe $P_w$ is defined by*

- $P_w := \{(i,j) \mid 1 \leq i \leq j \leq |w|\}$,
- $E^{\mathrm{NE}-\mathcal{A}^s_w} := \{((i,j),(i',j')) \mid j < i' \text{ or } j' < i, w[i,j] = w[i',j']\}$,
- $S^{\mathrm{NE}-\mathcal{A}^s_w} := \{((i,j),(j+1,j')) \mid 1 \leq i \leq j, j+1 \leq j' \leq |w|\}$,
- $L^{\mathrm{NE}-\mathcal{A}^s_w} := \{(1,j) \mid 1 \leq j \leq |w|\}$,
- $R^{\mathrm{NE}-\mathcal{A}^s_w} := \{(i,|w|)| \mid 1 \leq i \leq |w|\}$ *and,*
- *for every* $a \in \Sigma$, $D_a^{\mathrm{NE}-\mathcal{A}^s_w} := \{(i,i) \mid w[i] = a\}$.

*Let* $\mathrm{NE}-\mathcal{A}^s_w$ *be the standard-NE-*$w$*-structure with universe* $P_w$*. We define the* standard-E-$w$-structure $(\mathrm{E}-\mathcal{A}^{\mathrm{s}}_w)$ *with universe* $P^{\mathrm{E}}_w$ *as follows:*

- $P^{\mathrm{E}}_w := P_w \cup \{i_\varepsilon \mid 0 \leq i \leq |w|\}$,
- $E^{\mathrm{E}-\mathcal{A}^s_w} := E^{\mathrm{NE}-\mathcal{A}^s_w} \cup \{(i_\varepsilon, j_\varepsilon) \mid 0 \leq i,j \leq |w|, i \neq j\}$,
- $S^{\mathrm{E}-\mathcal{A}^s_w} := S^{\mathrm{NE}-\mathcal{A}^s_w} \cup \{(i_\varepsilon, i_\varepsilon) \mid 0 \leq i \leq |w|\} \cup$
  $\{((i,j), j_\varepsilon)) \mid 1 \leq i \leq j \leq |w|\} \cup \{(i_\varepsilon,(i+1,j)) \mid 0 \leq i \leq j \leq |w|\}$,
- $L^{\mathrm{E}-\mathcal{A}^s_w} := L^{\mathrm{NE}-\mathcal{A}^s_w} \cup \{0_\varepsilon\}$,
- $R^{\mathrm{E}-\mathcal{A}^s_w} := R^{\mathrm{NE}-\mathcal{A}^s_w} \cup \{|w|_\varepsilon\}$ *and,*
- *for every* $a \in \Sigma$, $D_a^{\mathrm{E}-\mathcal{A}^s_w} := D_a^{\mathrm{NE}-\mathcal{A}^s_w}$.

In the following lemma we state that the membership problem for pattern languages can be reduced to the homomorphism problem for relational structures. We shall informally explain this for the case of terminal-free NE-pattern languages. If there exists a substitution that maps the pattern $\alpha$ to the word $w$, then we can construct a homomorphism $g$ from $\mathcal{A}_\alpha$ to $\mathrm{NE}-\mathcal{A}^{\mathrm{s}}_w$ by mapping the positions of $\alpha$ to the factors of $w$ according to the substitution $h$. If two positions in $\alpha$ are adjacent, then so are their images under $h$ in $w$ and the same holds for equal variables in $\alpha$; hence, $g$ is a valid homomorphism. If, on the other hand, there exists a homomorphism $g$ from $\mathcal{A}_\alpha$ to $\mathrm{NE}-\mathcal{A}^{\mathrm{s}}_w$, then the elements of the universe of $\mathcal{A}_\alpha$, i. e., positions of $\alpha$, are mapped onto factors of $w$ such that a factorisation of $w$ is described. This is enforced by the relations $S$, $L$ and $R$. Furthermore, this mapping from $\alpha$ to $w$ induced by $g$ is a substitution, since the symmetric transitive closure of $E^{\mathcal{A}_\alpha}$ contains all pairs $(i,j)$ with $\alpha[i] = \alpha[j]$ and $\alpha[i] \in X$. For general patterns with terminal symbols and for the E case the idea is the same, but the situation is technically more complex.

**Lemma 7.** *Let* $\alpha$ *be a pattern,* $w \in \Sigma^*$ *and let* $\mathcal{A}_\alpha$ *be an* $\alpha$*-structure. Then* $w \in L_{\mathrm{NE}}(\alpha)$ *(or* $w \in L_{\mathrm{E}}(\alpha)$*) if and only if there exists a homomorphism from* $\mathcal{A}_\alpha$ *to* $\mathrm{NE}-\mathcal{A}^s_w$ *(or from* $\mathcal{A}_\alpha$ *to* $\mathrm{E}-\mathcal{A}^s_w$*, respectively).*

From Lemma 7 and Theorem 3, we can conclude that, for patterns that can be encoded by $\alpha$-structures with a bounded treewidth, the membership problem is solvable in polynomial time.

**Theorem 8.** *Let* $C \subseteq (X \cup \Sigma)^+$ *and let* $g$ *be a mapping that, in polynomial time, maps every* $\alpha \in C$ *to an* $\alpha$*-structure, such that* $\widehat{C} := \{g(\alpha) \mid \alpha \in C\}$ *has bounded treewidth. Then* NE-PATMem$(C)$ *and* E-PATMem$(C)$ *are decidable in polynomial time.*

Due to Theorem 8, the task of identifying parameters of patterns that, if bounded, allow a polynomial time membership problem, can now be seen from a different angle, i. e., as the problem of finding classes of patterns that can be encoded by $\alpha$-structures with a bounded treewidth. The fact that we can easily rephrase known results about the complexity of the membership problem for pattern languages in terms of standard $\alpha$-structures with a bounded treewidth, pointed out by the following proposition, indicates that this point of view is natural and fits with our current knowledge of the membership problem.

**Proposition 9.** *Let $\alpha$ be a pattern. If $\alpha$ is non-cross or regular then* $\mathrm{tw}(\mathcal{A}^s_\alpha) \leq 2$. *Furthermore,* $\mathrm{tw}(\mathcal{A}^s_\alpha) \leq |\operatorname{var}(\alpha)|$.

While Proposition 9 is trivially true, an analogous result can also be given for the variable distance of patterns (see Section 1 and [14]), which is a more subtle parameter the restriction of which is known to yield a polynomial time membership problem. This follows from the main result of the subsequent section, which shows that a stronger parameter than the variable distance, namely the already mentioned scope coincidence degree, also allows a polynomial time membership problem if restricted to a constant. The question arises why we do not simply consider the treewidth of a pattern $\alpha$, i. e., $\mathrm{tw}(\alpha) := \min\{\mathrm{tw}(\mathcal{A}_\alpha) \mid \mathcal{A}_\alpha \text{ is an } \alpha\text{-structure}\}$, to be an appropriate parameter of patterns that should be bounded in order to solve the membership problem efficiently. The problem here is that, firstly, for a pattern $\alpha$ there exists an exponential number of $\alpha$-structures and, secondly, computing the treewidth of graphs is an NP-complete problem. Consequently, it is not clear whether this parameter can be computed in polynomial time and, in order to conclude complexity theoretical results from Theorem 8, we rely on finding easily computable parameters of patterns that – ideally as tight as possible – are upper bounds for $\mathrm{tw}(\alpha)$.

## 4 Patterns with Restricted Scope Coincidence Degree

In this section we show that, for every pattern $\alpha$, the treewidth of the standard $\alpha$-structure is bounded by the scope coincidence degree of $\alpha$. To this end, we shall play the robber-cops-game defined in Section 2 on the Gaifman graph of standard $\alpha$-structures. For the sake of convenience, we shall not distinguish anymore between a pattern $\alpha$ and the Gaifman graph of its standard $\alpha$-structure, i. e., we change at will between interpreting the positions in the pattern as occurrences of variables or as vertices in the Gaifman graph of the standard $\alpha$-structure. Similarly, we allow some leeway with respect to the robber-cops-game and shall play it directly on a pattern $\alpha$, meaning to actually playing it on the Gaifman graph of its standard $\alpha$-structure. Next, we define a strategy to search a pattern in terms of the robber-cops-game.

**Definition 10.** *Let $\alpha$ be a pattern. We define the* inspector search strategy *(on $\alpha$). We assume that we have an infinite number of cops, where one distinct cop is called the* inspector *and all other cops are called* constables*. Let $m$, $1 \leq m \leq |\alpha|$,* *be the leftmost occurrence of a variable in $\alpha$. Initially, the inspector is placed on vertex $m$. For every $i$, $m \leq i \leq |\alpha| - 1$, when the inspector is located on vertex $i$, the following steps are executed:*

1. *If $\alpha[i]$ is the leftmost occurrence of some variable $x$ in $\alpha$, then a constable is placed on vertex $i$. If $\alpha[i]$ is an occurrence of some variable $x$, but not the leftmost one, then the constable from vertex $j$ is moved to vertex $i$, where $j < i$, is the rightmost occurrence of $x$ that is currently occupied by a constable.*
2. *The inspector moves from vertex $i$ to vertex $i + 1$.*
3. *If $i$ is the rightmost occurrence of some variable $x$ in $\alpha$, then the constable on vertex $i$ is removed.*

*The number of constables that are required to carry out the inspector search strategy on $\alpha$ is called the* constable number of *$\alpha$.*

Informally, the inspector search strategy on some pattern $\alpha$ can be described in the following way. The inspector moves through the pattern from left to right. Every occurrence of a terminal symbol is ignored and the inspector just moves on. If it enters an occurrence of a variable, it places a constable there. A new constable is required if this is the first occurrence of some variable $x$. If, on the other hand, there exists an earlier occurrence of $x$ in $\alpha$, then, by definition of the search strategy, a constable is located at the next occurrence of $x$ to the left of the current inspector position. This constable is now moved forward to the position of the inspector. If the inspector reaches a rightmost occurrence of a variable, then also a constable is moved to this position, but removed immediately after the inspector moves on. However, it is important that the constable is placed there before the inspector moves on and remains there while the inspector moves to the next position. Obviously, this procedure terminates as soon as the inspector reaches position $|\alpha|$. We note that as long as there exists at least one variable in the pattern, the constable number is at least one, hence, we can assume that the constable number is always at least one.

We observe the following property of the inspector search strategy.

**Proposition 11.** *Let $\alpha$ be a pattern. Every time step* 1 *of the inspector search strategy on $\alpha$ is executed, the following condition is satisfied. Let $p_1, p_2, \ldots, p_k \in \{1, 2, \ldots, |\alpha|\}$ with $p_1 < p_2 < \ldots < p_k$ be the positions occupied by constables. Then there are $k$ different variables $y_1, y_2, \ldots, y_k$, such that $\mathrm{var}(\alpha[p_1, p_k]) = \{y_1, y_2, \ldots, y_k\}$ and, for every $i$, $1 \leq i \leq k$, $p_i$ is the rightmost occurrence of $y_i$ in $\alpha[p_1, p_k]$.*

The next lemma describes how a certain area of the pattern can be sealed off by the constables, such that the robber cannot reach this area.

**Lemma 12.** *Let $\alpha$ be a pattern and let $p_1, p_2, \ldots, p_k \in \{1, 2, \ldots, |\alpha|\}$ with $p_1 < p_2 < \ldots < p_k$ such that $\mathrm{var}(\alpha[p_1, p_k]) = \{y_1, y_2, \ldots, y_k\}$ and, for every $i$, $1 \leq i \leq k$, $p_i$ is the rightmost occurrence of $y_i$ in $\alpha[p_1, p_k]$. If vertices $p_1, \ldots, p_k$ are occupied by constables, then the robber cannot reach a vertex $t$, $p_1 \leq t \leq p_k$, from a vertex $s$, $p_k < s$.*

The previous results can be used in order to show that, for every pattern $\alpha$, a robber can be caught by applying the inspector search strategy on $\alpha$.

**Lemma 13.** *Let $\alpha$ be a pattern with constable number $k$. If $k = 1$, then 3 cops are sufficient to catch a robber on the Gaifman graph of $\mathcal{A}^s_\alpha$, and if $k \geq 2$, then $k + 1$ cops are sufficient to catch a robber on the Gaifman graph of $\mathcal{A}^s_\alpha$.*

It is worth mentioning that the special case in the above lemma concerning patterns with a constable number of 1 is caused by the fact that the patterns may contain terminals. For terminal-free patterns $\alpha$ with a constable number of 1, a robber can be caught on the Gaifman graph of $\mathcal{A}^s_\alpha$ by 2 cops. Next, we show that the constable number of a pattern equals its scope coincidence degree.

**Lemma 14.** *For every pattern $\alpha$, the constable number of $\alpha$ equals $\operatorname{scd}(\alpha)$.*

By the previous lemmas, we can conclude that, for every pattern $\alpha$ with $\operatorname{scd}(\alpha) = 1$, a robber can be caught on $\alpha$ by using one inspector and 2 constables, and for every pattern $\alpha$ with $\operatorname{scd}(\alpha) = k$, $k \geq 2$, a robber can be caught on $\alpha$ by using one inspector and $k$ constables. Hence, referring to Theorem 1, the treewidth of the standard $\alpha$-structure is bounded by the scope coincidence degree. Furthermore, the standard $\alpha$-structures of the class of patterns with restricted variable distance have a bounded treewidth as well.

**Theorem 15.** *Let $\alpha$ be a pattern. Then $\operatorname{tw}(\mathcal{A}^s_\alpha) \leq \operatorname{scd}(\alpha) \leq \operatorname{vd}(\alpha) + 1$.*

Theorems 8 and 15 imply the following complexity result.

**Corollary 16.** *Let $k \in \mathbb{N}$ and $Z \in \{\mathrm{E}, \mathrm{NE}\}$. The problem $Z$-PATMem$(\{\alpha \mid \operatorname{scd}(\alpha) \leq k\})$ is solvable in polynomial time.*

We conclude this work by some questions not explicitly addressed so far. Since in Definition 4 we leave the exact definition of the relation symbol $E$ open, there are many possible $\alpha$-structures for a pattern $\alpha$ that all permit an application of Theorem 8. However, the standard way of encoding patterns (Definition 5) has turned out to be sufficient for all results in the present paper. Hence, it would be interesting to know whether or not, for some pattern $\alpha$, there exists a better $\alpha$-structure than the standard one, i. e., $\operatorname{tw}(\alpha) < \operatorname{tw}(\mathcal{A}^s_\alpha)$. This question is open, but we conjecture that it can be answered in the negative.

Another question is whether the scope coincidence degree of a pattern $\alpha$ is a tight upper bound for the treewidth of the standard $\alpha$-structure. This question can be answered in the negative. Consider for example the pattern $\alpha := x_1 \cdot x_2 \cdot \cdots \cdot x_{k-1} \cdot x_k \cdot x_k \cdot x_{k-1} \cdot \cdots \cdot x_2 \cdot x_1$. It is easy to see that $\operatorname{scd}(\alpha) = k$. On the other hand, we can catch a robber on $\alpha$ with 3 cops. Thus $\operatorname{tw}(\mathcal{A}^s_\alpha) \leq 2 < \operatorname{scd}(\alpha)$. This examples also gives an indication that it might be possible to identify a parameter closer to $\operatorname{tw}(\alpha)$ still preserving polynomial time computability.

**Acknowledgements.** The authors wish to thank Sanjay Jain and the anonymous referees for their detailed and helpful remarks and suggestions, which will be taken into account for the full version of this paper.

## References

1. Angluin, D.: Finding patterns common to a set of strings. Journal of Computer and System Sciences 21, 46–62 (1980)
2. Bodlaender, H.L.: Treewidth: Characterizations, Applications, and Computations. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 1–14. Springer, Heidelberg (2006)
3. Bordihn, H., Dassow, J., Holzer, M.: Extending regular expressions with homomorphic replacement. RAIRO Theoretical Informatics and Applications 44, 229–255 (2010)
4. Bremer, J., Freydenberger, D.D.: Inclusion Problems for Patterns with a Bounded Number of Variables. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 100–111. Springer, Heidelberg (2010)
5. Câmpeanu, C., Salomaa, K., Yu, S.: A formal study of practical regular expressions. International Journal of Foundations of Computer Science 14, 1007–1018 (2003)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer-Verlag New York, Inc., Secaucus (2006)
7. Freuder, E.C.: Complexity of k-tree structured constraint satisfaction problems. In: Proceedings of the 8th National Conference on Artificial Intelligence, pp. 4–9 (1990)
8. Lange, S., Wiehagen, R.: Polynomial-time inference of arbitrary pattern languages. New Generation Computing 8, 361–370 (1991)
9. Mateescu, A., Salomaa, A.: Patterns. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 230–242. Springer, Heidelberg (1997)
10. Ng, Y.K., Shinohara, T.: Developments from enquiries into the learnability of the pattern languages from positive data. Theoretical Computer Science 397, 150–165 (2008)
11. Reidenbach, D.: A non-learnable class of E-pattern languages. Theoretical Computer Science 350, 91–102 (2006)
12. Reidenbach, D.: Discontinuities in pattern inference. Theoretical Computer Science 397, 166–193 (2008)
13. Reidenbach, D., Schmid, M.L.: Finding Shuffle Words That Represent Optimal Scheduling of Shared Memory Access. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 465–476. Springer, Heidelberg (2011)
14. Reidenbach, D., Schmid, M.L.: A Polynomial Time Match Test for Large Classes of Extended Regular Expressions. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 241–250. Springer, Heidelberg (2011)
15. Reischuk, R., Zeugmann, T.: An average-case optimal one-variable pattern language learner. Journal of Computer and System Sciences 60, 302–335 (2000)
16. Rossmanith, P., Zeugmann, T.: Stochastic finite learning of the pattern languages. Machine Learning 44, 67–91 (2001)
17. Seymour, P.D., Thomas, R.: Graph searching and a min-max theorem for treewidth. Journal of Combinatorial Theory, Series B 58, 22–33 (1993)
18. Shinohara, T.: Polynomial Time Inference of Extended Regular Pattern Languages. In: Goto, E., Furukawa, K., Nakajima, R., Nakata, I., Yonezawa, A. (eds.) RIMS 1982. LNCS, vol. 147, pp. 115–127. Springer, Heidelberg (1983)
19. Shinohara, T.: Polynomial time inference of pattern languages and its application. In: Proc. 7th IBM MFCS, pp. 191–209 (1982)
20. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, ch. 7, vol. 3, pp. 389–455. Springer, Heidelberg (1997)

# P–NP Threshold for Synchronizing Road Coloring

Adam Roman

Faculty of Mathematics and Computer Science
Jagiellonian University, Cracow, Poland
roman@ii.uj.edu.pl

**Abstract.** The parameterized Synchronizing-Road-Coloring Problem (in short: SRCP$^\ell_C$) in its decision version can be formulated as follows: given a digraph $G$ with constant out-degree $\ell$, check if $G$ can be synchronized by some word of length $C$ for some synchronizing labeling. We consider the family $\{SRCP^\ell_C\}_{C,\ell}$ of problems parameterized with constants $C$ and $\ell$ and try to find for which $C$ and $\ell$ SRCP$^\ell_C$ is **NP**-complete. It is known that SRCP$^3_C$ is **NP**-complete for $C \geq 8$. We improve this result by showing that it is so for $C \geq 4$ and for $\ell \geq 3$. We also show that SRCP$^\ell_C$ is in **P** for $C \leq 2$ and $\ell \geq 1$. Hence, we solve SRCP almost completely for alphabet with 3 or more letters. The case $C = 3$ is still an open problem.

## 1 Introduction

We will call a digraph $G$ the RCP-graph, if it is strongly connected and has constant out-degree. The Road Coloring Problem (RCP) originates in [2] and it was stated explicitly in the paper by Adler et al. [1]. It can be formulated as follows: let $G$ be an RCP-graph such that the greatest common divisor (gcd) of the lengths of all cycles in $G$ equals 1. Is there an edge labeling, turning $G$ into a deterministic finite synchronizing automaton? The problem is of great importance in automata theory, because the synchronizing property makes the automaton behavior resistant to errors that could occur in an input word: after the error is detected, the synchronizing word can reset the automaton to its initial state, as if there were no error. In this way we regain the control over the automaton action. Trahtman [7] solved the RCP by showing that a synchronizing labeling exists for any RCP-graph $G$ if and only if the gcd of the lengths of all cycles in $G$ equals 1. RCP uses a notion of synchronization, which in fact was introduced few years before the work of Adler et al. The 'classical' version of the synchronizing problem (SP) can be defined as follows:

**(SP)** given an automaton (a *labeled* graph $G$ with constant out-degree) and a natural number $C$, check if there exists a synchronizing word of length $C$ for $G$.

In this problem there is no restriction on the length of the synchronizing sequence – this value is given as a part of the input. It is known that (SP) is **NP**-complete [4] for binary alphabets. The solution of RCP opened a new, broad field of research. For example, [3,6] deal with algorithms for finding a synchronizing labeling given a graph $G$. We can reformulate all classical synchronizing problems in the 'RCP fashion', using the following schema:

**(classical version)** given a constant out-degree graph $G$ with labeling $\delta$, check if $COND(G, \delta)$ holds,

**(RCP version)** given a constant out-degree graph $G$, check if $COND(G, \delta)$ holds for some labeling $\delta$.

In this way we may also reformulate (SP): given an RCP-graph $G$ and a natural number $C$, check if $G$ can be synchronized by some word of length $C$ for some synchronizing labeling. However, we will consider more restricted problem (in fact, a family of problems parameterized by two constants):

**($\mathbf{SRCP}^{\ell}_{C}$)** given a graph $G$ with constant out-degree $\ell$, check if $G$ can be synchronized by some word of length $C$ for some synchronizing labeling.

Notice that here $C$ is not a part of the input, but is a constant number. The corresponding classical problem would be: given an automaton, check if it can be synchronized by some word of length $C$. This problem can be obviously solved in polynomial time $O(|Q| \cdot |A|^C)$ by checking the synchronizing property for all possible words of length $C$. Our main result states that the RCP version of this problem is **NP**-complete for almost all $C$ and $\ell$ values.

Thanks to Trahtman's result [7] we may restrict our research only to graphs with the gcd of the lengths of all cycles equal to 1. The motivation for this work was the question about the complexity of Synchronizing-Road-Coloring Problem posted by Volkov during the Wroclaw Conference on the Černý Conjecture [8] and a partial solution published in [5]. The other motivation is that not only synchronizing issues, but also their RPC versions are of practical nature. A good example of application of the RCP-type problems to the real life is given in [9].

The paper is constructed as follows: in Section 2 we give some necessary definitions and notations. In Section 3 we show that $\mathrm{SRCP}^{\ell}_{C}$ is **NP**-complete for $C \geq 4$ and $\ell \geq 3$. This improves the result from [5] saying that it is so for $C \geq 8$ and $\ell = 3$. The proof for $\mathrm{SRCP}^{3}_{4}$ is even simpler and shorter than the one from [5] for $\mathrm{SRCP}^{3}_{8}$. In Section 4 we show that $\mathrm{SRCP}^{3}_{C}$ is in **P** for $C = 1, 2$ and $\ell \geq 1$. In Section 5 we summarize our results and discuss some open problems.

## 2 Preliminaries

An *automaton* is a triple $\mathcal{A} = (Q, A, \delta)$, where $Q$ is a nonempty, finite set of states, $A$ is a finite alphabet and $\delta : Q \times A \to Q$ is a *transition function* called also the automaton action. By $A^*$ we denote a free monoid over $A$ consisting of all finite words over $A$. By $\varepsilon$ we denote the empty word of length 0. We define $A^+ = A^* \setminus \{\varepsilon\}$ and $A^n = \{w \in A^* : |w| = n\}$. For the sake of simplicity, we will write $p.a = q$ instead of $\delta(p, a) = q$. It is convenient to extend $\delta$ to the subsets of $Q$ in the usual way: for $P \subset Q$ we define $P.\varepsilon = P$, $P.a = \bigcup_{p \in P}\{p.a\}$ and

$P.a\omega = (P.a).\omega$ for all $\omega \in A^+$. We say that $W \in A^*$ *synchronizes* $\mathcal{A} = (Q, A, \delta)$ if $|Q.W| = 1$. If such a word exists, $\mathcal{A}$ is called a *synchronizing automaton*. Sometimes we will be interested in the 'local' synchronization only: if $P \subset Q$, then we say that $W \in A^*$ is $P$-synchronizing (or that $W$ synchronizes $P$) if $|P.W| = 1$. By $q.a^{-1}$ we understand the set $\{p : p.a = q\}$.

From now on, we will consider only strongly connected, directed graphs with constant out-degree. Let $G = (\mathcal{V}, \mathcal{E})$ be such a graph, where $\mathcal{V}$ is a finite set of vertices and $\mathcal{E}$ is a (multi)set of edges. Let each vertex have an out-degree $k$ and let $A$ be a $k$-element alphabet. An *automaton with underlying graph* $G = (\mathcal{V}, \mathcal{E})$ is an automaton $\mathcal{A}_G = (\mathcal{V}, A, \delta)$ such that for each $v \in \mathcal{V}$ we have $\{v.a : a \in A\} = \{v' : (v, v') \in \mathcal{E}\}$, where the equality has to be understood in the multiset sense (note: sometimes we will use the sentence "all other edges of $G$ can be labeled arbitrarily" – it means that the resultant labeling must fulfill the above condition). A *path* in $G$ is a sequence of vertices $(v_1, v_2, ..., v_n)$, $n \geq 2$ such that $(v_i, v_{i+1}) \in \mathcal{E}$ for each $1 \leq i \leq n-1$. The *length* of the path is $n-1$. The path $(v_1, v_2, ..., v_{n+1})$ is called an $n-(v_1, v_{n+1})$ path. By $reach_n(v)$ we denote the set of all vertices that can be reached from $v$ by some path of length $n$: $reach_n(v) = \{v.w : w \in A^n\}$.

**Lemma 1.** *Let $\mathcal{A}_G = (\mathcal{V}, A, \delta)$ be an automaton with the underlying graph $G = (\mathcal{V}, \mathcal{E})$. If there exists a synchronizing word $W$ of length $C$ for $\mathcal{A}_G$, then*

$$\bigcap_{v \in \mathcal{V}} reach_C(v) \neq \emptyset.$$

*Proof.* Let $W$ be a synchronizing word of length $C$. Then there exists $v' \in \mathcal{V}$, such that $v.W = v'$ for all $v \in \mathcal{V}$ and obviously $v.W \in reach_C(v)$ for all $v \in \mathcal{V}$. Therefore, $v' \in \bigcap_{v \in \mathcal{V}} \neq \emptyset$. $\square$

A *labeling* of $G$ is a function $L : \mathcal{E} \to A$ which assigns letters from $A$ to the edges in such a way that for each vertex $v \in \mathcal{V}$ all of its $k$ outgoing edges have pairwise different labels (colors). The labeled graph becomes an automaton and $L$ is equivalent to $\delta$, so slightly abusing the notation sometimes we will refer to $\delta$ as a labeling. If $v \in \mathcal{V}$, by $out(v)$ we denote the (multi)set of outgoing edges from $v$.

## 3 $\mathbf{SRCP^3_C}$ Is NP-complete for $C \geq 4$ and $\ell \geq 3$

Our reasoning for proving **NP**-completeness of $\mathrm{SRCP}^\ell_C$ for $C \geq 4$ and $\ell \geq 3$ consists of three steps. In the first step we prove **NP**-completeness of $\mathrm{SRCP}^3_4$. In the second step we show that this result can be extended to $C \geq 4$. In the last step we show that the result from the second step can be extended to $\ell \geq 3$.

First, consider the Synchronizing-Road-Coloring problem $\mathrm{SRCP}^3_4$:

– INPUT: RCP-graph $G$ with constant out-degree 3;
– OUTPUT: "YES" if there exists a synchronizing word of length 4 for some synchronizing labeling of $G$. "NO" otherwise.

It is clear that $\mathrm{SRCP}^{\ell}_{C}$ is in **NP** for any $C$ and $\ell$: given a labeling and a word $W$ one can check if $W$ synchronizes $G$ and it can be done in $O(|W| \cdot |G|)$ time, where $|G|$ is the number of vertices in $G$. We will prove that $\mathrm{SRCP}^{3}_{4}$ is **NP**-complete by reducing 3-SAT to this problem.

Let $\varphi$ be a 3-SAT formula in a conjunctive normal form with $n$ clauses $c_1, ..., c_n$ and $z$ variables $x_1, ..., x_z$. Let us define a graph $\Gamma_\varphi = (X, Y)$ corresponding to $\varphi$ in the following way: $X = \{1, ..., z\}$ and for $j \neq j'$ $\{j, j'\} \in Y$ iff there is a clause $c_i$ such that both $\{x_j, \neg x_j\} \cap c_i$ and $\{x_{j'}, \neg x_{j'}\} \cap c_i$ are nonempty. We will consider only formulas in a so-called standard form. By a standard form we mean that: (1) if literal belongs to $c_i$, then its negation does not; (2) for each $j = 1, ..., z$ there exist two clauses $c_i$ and $c_{i'}$, $i \neq i'$ such that $x_j \in c_i$ and $\neg x_j \in c_{i'}$; (3) $\Gamma_\varphi$ is connected. Notice that if $\Gamma_\varphi$ is not connected and has $\kappa$ components, the formula can be split into $\kappa$ formulas and for each of them we can assign variables values independently.

We will construct a digraph $G = G(\varphi) = (\mathcal{V}, \mathcal{E})$ such that there exists a synchronizing word of length 4 for some synchronizing labeling of $G$ if and only if $\varphi$ is satisfiable. The main construction of $G$ is presented in Fig. 1. The digraph $G$ has constant out-degree 3, so we will consider automata over 3-letter alphabet $A = \{a, b, c\}$. The gcd of all cycles in $G$ equals 1 (because, for example, of the cycles $(t, v, x, t)$ and $(t, v, x, u, t)$ of lengths 3 and 4 resp.). The digraph $G$ consists of $2n + 8z + 22$ vertices, so its size is polynomial in $n, z$. For the sake of simplicity, some edges are not shown in Fig. 1. These are: $\overline{c}_i \rightarrow c_i$ for each $i \in \{1, \ldots, z\}$ and $\overline{l} \rightarrow l$ for each $l \in \{s_1, r_1, \ldots, s_z, r_z, t, u, v, w, x, y, q\}$. Each bolded arrow denotes a set of three edges. Small black and white disks are used for simplifying the presentation. All edges outgoing to black (resp. white) disks are in fact the edges outgoing to state $k$ (resp. $r_1$).

The top part of the graph, denoted by $G_\varphi$, is a "formula gadget" and it depends on the $\varphi$ formula. It is constructed as follows: if a clause $c_j$ contains a literal $l_i \in \{x_i, \neg x_i\}$, we put $(c_j, l_i) \in \mathcal{E}$. This property will be utilized in Lemma 2. The most important part of the construction consists of the subgraph $G^*$, together with the states $r_i, s_i, x_i, \neg x_i$, $i = 1, \ldots, z$. This part is called the "main gadget". It has a property stated in Lemmata 5 and 6, which plays a key role in the proof of the main theorem. The formula gadget, together with the main gadget, has the desired property: the formula $\varphi$ is satisfiable if and only if there exists a synchronizing word of length 4 for some $\mathcal{A}_G$. The Road Coloring Problem was stated for strongly connected digraphs, so we need to introduce one more gadget to provide the strong connectivity of $G$. This is the graph $\overline{G}$. It is easy to observe that adding it makes $G$ strongly connected, because all states can reach $t$, $t$ can reach all states in $\overline{G}$ and each state in $\overline{G}$ can reach any state in $G$.

**Lemma 2.** *Let $C = \{c_1, c_2, ..., c_n\} \subset \mathcal{V}$. The following statements are equivalent:*

*(a) $\varphi$ is satisfiable,*
*(b) there exists a labeling $\delta$ for $G_\varphi$ such that*

$$\forall i \in \{1, ..., z\} \; |\{x_i, \neg x_i\} \cap C.\alpha| \leq 1$$

*for some $\alpha \in A$.*

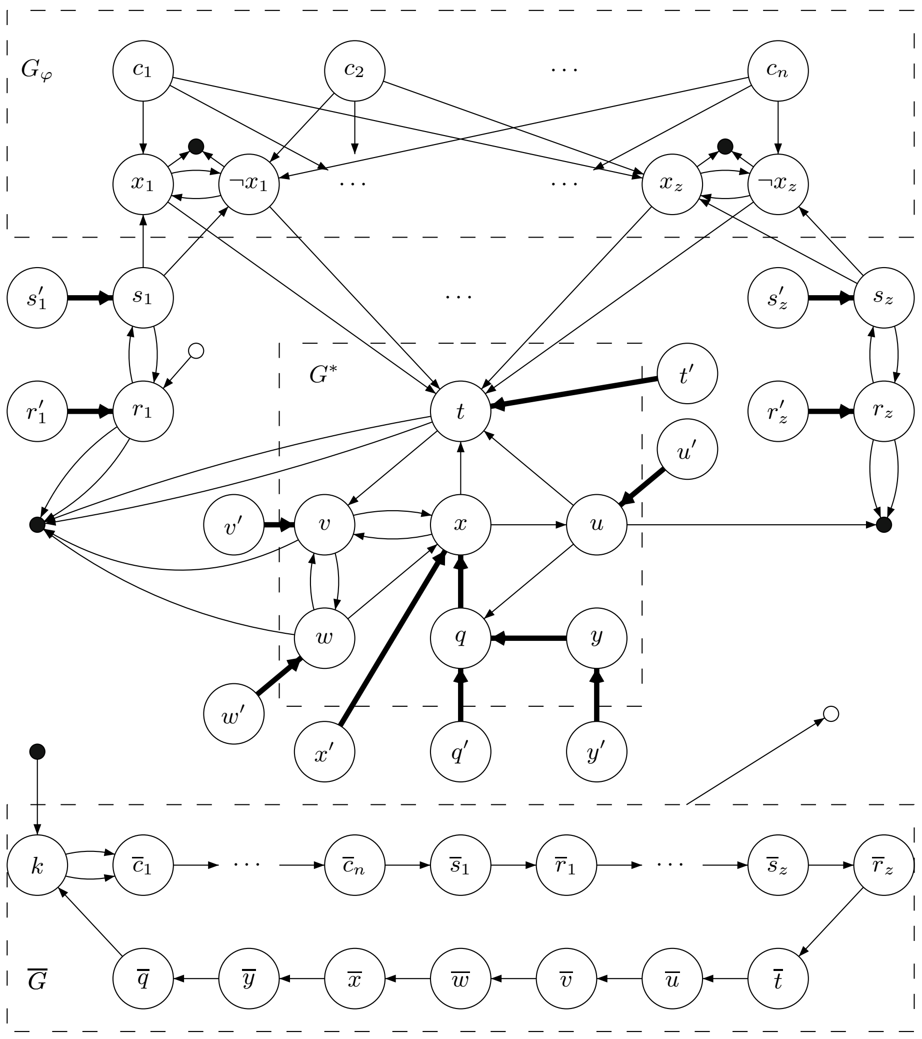


**Fig. 1.** Main construction (not all edges are shown)

*Moreover, there exists at least one index* $j$, $1 \leq j \leq z$ *such that* $|\{x_j, \neg x_j\} \cap C.\alpha| = 1$.

*Proof.* ($a \Rightarrow b$) Let $\varphi$ be satisfiable. Then there exists an assignment such that in each clause $c_i$ there is at least one literal $l_j \in \{x_j, \neg x_j\}$ with the true value. We put

$$\delta(c_i, \alpha) = l_j. \tag{1}$$

If two or more literals in $c_i$ take true values, we can take as $\delta(c_i, \alpha)$ the literal with the smallest index $j$. It is clear that if $c_i.\alpha = l_j$ for some $i, j$, then for each $i' \neq i$ we have $c_{i'}.\alpha \neq \neg l_j$.

($b \Rightarrow a$) Let $\delta$ be a labeling fulfilling (b). We assign true value to $l_j$ if and only if $\delta(c_i, \alpha) = l_j$ for some $i \in \{1, ..., n\}$. Such an assignment is correct and there is at least one literal with true value in each clause.

The existence of $j$ such that $|\{x_j, \neg x_j\} \cap C.\alpha| = 1$ comes from the fact that $\bigcup_{i=1}^{z}\{x_i, \neg x_i\}.\alpha^{-1} = C \neq \emptyset$. □

Let $\mathcal{A}_G = (\mathcal{V}, \{a, b, c\}, \delta)$ be an automaton with the underlying graph $G$ from Fig. 1. Let us define the sets: $G_i = \{r_i, s_i\}$, $1 \leq i \leq z$, $D = \bigcup_{i=1}^{z} G_i$ and $E = \{t, u, v, w, x, y, q\}$. The following Proposition is straightforward.

**Proposition 3.** *For each automaton $\mathcal{A}_G$ and for every word $W$ of length 1 the set $\mathcal{V}.W$ contains $D \cup E$ and has a nonempty intersection with $\bigcup_{i=1}^{z}\{x_i, \neg x_i\}$.*

Proposition 3 states that irrespective of a labeling for $G$, the set $D \cup E$ is always contained in the set being the image of $\mathcal{V}$ after applying *any* word of length 1.

**Lemma 4.** *The formula $\varphi$ is satisfiable if and only if there exists a labeling $\delta$ of $G(\varphi)$ and $W \in A$ such that for each $i \in \{1, ..., z\}$ the set $\mathcal{V}.W$ contains at most one of the elements $x_i, \neg x_i$.*

*Proof.* The proof follows directly from Lemma 2, Proposition 3 and the analysis of $G$. □

Now we formulate two key lemmata that will be used to prove **NP**-completeness of the SYNCHRONIZING-ROAD-COLORING problem.

**Lemma 5.** *Let $\mathcal{A}_G$ be an automaton and let $i \in \{1, 2, ..., z\}$. If there is a word $W \in A^*$ which synchronizes the set $G_i \cup E \cup \{x_i\}$, then $W$ has length at least 3. If $W$ synchronizes $G_i \cup E \cup \{\neg x_i\}$, then $W$ also has length at least 3. Moreover, in both cases, there exists such a word of length 3.*

*Proof.* We will prove the first implication. The second one can be proved in the same way. Fix $i$ and put $H = G_i \cup E \cup \{x_i\}$. It is easy to observe that $\bigcap_{h \in H} reach_\lambda(h) = \emptyset$ for $\lambda = 1, 2$, so by Lemma 1 there exists no $H$-synchronizing word of length $\leq 2$ for any labeling $\delta$. Notice that $reach_3(y) \cap reach_3(r_i) = \{t\}$ and $H \subset t^{-3}$, so $\bigcap_{h \in H} reach_3(h) = \{t\}$. Hence, if there is an $H$-synchronizing word $W$ of length 3, then $H.W = t$.

Let $W = \alpha\beta\gamma$, $\alpha, \beta, \gamma \in A$. We want to find all possible labelings of $H$ for which there is an $H$-synchronizing word of length 3. Notice that each path going from any vertex from $H$ through $k$ to $t$ must be of length $> 3$, because $t \notin reach_2(k)$. The only $3-(x, t)$ path is $x \xrightarrow{\alpha} v \xrightarrow{\beta} x \xrightarrow{\gamma} t$, so $\alpha \neq \gamma$. The only $3-(q, t)$ path is $q \xrightarrow{\alpha} x \xrightarrow{\beta} u \xrightarrow{\gamma} t$, so $\beta \neq \alpha$ and $\beta \neq \gamma$. Having this labeling it is easy to observe that necessarily $t \xrightarrow{\alpha} v$, $v \xrightarrow{\alpha} w$, $w \xrightarrow{\beta} x$, $w \xrightarrow{\alpha} v$, $u \xrightarrow{\alpha} q$, $x_i \xrightarrow{\alpha} y_i$, $\neg x_i \xrightarrow{\beta} x_i$, $x_i \xrightarrow{\gamma} t$, $r_i \xrightarrow{\alpha} s_i$, $s_i \xrightarrow{\alpha} \neg x_i$, $s_i \xrightarrow{\beta} x_i$, $s_i \xrightarrow{\gamma} r_i$. Edge $\neg x_i \to t$ may be labeled by $\alpha$ or $\gamma$. The above necessary labeling is shown in Fig. 2. The shortest $H$-synchronizing word is $W = \alpha\beta\gamma$, where $\alpha, \beta, \gamma$ are pairwise different letters from $A$. □

**Lemma 6.** *Let $\mathcal{A}_G$ be an automaton and let $i \in \{1, 2, ..., z\}$. If there is a word $W \in A^*$ which synchronizes the set $G_i \cup E \cup \{x_i, \neg x_i\}$, then $W$ has length greater than 3.*

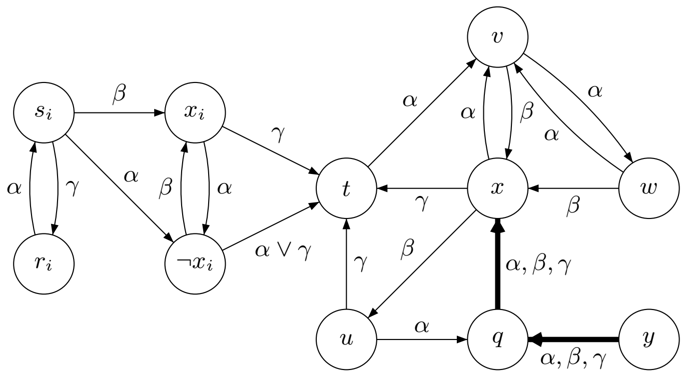


**Fig. 2.** Graph $H \cup \{\neg x_i\}$ with the necessary synchronizing labeling for a $H$-synchronizing word of length 3

*Proof.* Put $H = G_i \cup E \cup \{x_i, \neg x_i\}$. Suppose that there exists an $H$-synchronizing word $W$ of length $\leq 3$. But then $W$ synchronizes also $H \setminus \{\neg x_i\}$, so from Lemma 5 we know that $W = \alpha\beta\gamma$, where $\alpha, \beta, \gamma$ are pairwise different and that the synchronization takes place in $t$. Figure 2 presents the necessary labeling for any automaton with an $(H \setminus \{\neg x_i\})$-synchronizing word of length 3. The only $3-(\neg x_i, t)$ path is $\neg x_i \to x_i \to \neg x_i \to t$, but we have $\neg x_i \xrightarrow{\beta} x_i$ – a contradiction with the form of $W = \alpha\beta\gamma$. □

**Theorem 7.** *$SRCP^3_4$ is **NP**-complete.*

*Proof.* Let $\varphi$ be a 3-SAT formula in a normal form. We construct $G = G(\varphi) = (\mathcal{V}, \mathcal{E})$ as in Fig. 1 and let $\mathcal{A}_G = (\mathcal{V}, A, \delta)$ be an automaton with the underlying graph $G$. We will show that $\varphi$ is satisfiable if and only if there exists a synchronizing word of length 4 for some $\mathcal{A}_G$.

If $\varphi$ is not satisfiable, then by Lemma 4 for each $\mathcal{A}_G$ and for any $W \in A$ there exists $i \in \{1, 2, ..., z\}$ such that $H = D \cup E \cup \{x_i, \neg x_i\} \subset \mathcal{V}.W$. By Lemma 6 there is no $H$-synchronizing word of length $\leq 3$, so obviously there is no synchronizing word for $\mathcal{A}_G$ of length 4. If $\varphi$ is satisfiable, consider the following labeling:

$\delta|_{G_\varphi}$is given by (1) for $\alpha = a$

$\delta|_E$ is given as in Fig. 2 for $\alpha = a, \beta = b, \gamma = c$

$\delta(k, a) = \delta(\overline{t}, a) = \delta(\overline{u}, a) = \delta(\overline{v}, a) = \delta(\overline{w}, a) = \delta(\overline{x}, a) = \delta(\overline{y}, a) = \delta(\overline{q}, a) = r_1$

$\forall i \in \{1, ..., z\}\ \delta(r_i, a) = s_i, \delta(s_i, c) = r_i, \delta(x_i, c) = \delta(\neg x_i, c) = t$

$\forall i \in \{1, ..., n\}\ \delta(\overline{c}_i, a) = r_1,\ \forall i \in \{1, ..., z\}\ \delta(\overline{r}_i, a) = \delta(\overline{s}_i, a) = r_1$

$$\forall i \in \{1, ..., z\}\ \delta(x_i, a) = \begin{cases} \neg x_i \text{ if } x_i.a^{-1} \neq \emptyset \\ k \text{ otherwise} \end{cases}, \delta(x_i, b) = \begin{cases} k \text{ if } x_i.a^{-1} \neq \emptyset \\ \neg x_i \text{ otherwise} \end{cases}$$

$$\forall i \in \{1, ..., z\}\ \delta(\neg x_i, a) = \begin{cases} x_i \text{ if } \neg x_i.a^{-1} \neq \emptyset \\ k \text{ otherwise} \end{cases}, \delta(\neg x_i, b) = \begin{cases} k \text{ if } \neg x_i.a^{-1} \neq \emptyset \\ x_i \text{ otherwise} \end{cases}$$

Other edges can be labeled arbitrarily. It is easy to see that $\mathcal{V}.a = D \cup E \cup \bigcup_{i=1}^{z} F_i$, where $F_i = \{x_i^{m-4}\}$ or $F_i = \{y_i^{m-4}\}$ or $F_i = \emptyset$ and for at least one $i$ we have $F_i \neq \emptyset$. From Lemma 5 we have that $|\mathcal{V}.aabc| = 1$, so there is a labeling for which there exists a synchronizing word of length 4. $\square$

So far we have shown that $SRCP_C^3$ is **NP**-complete for $C = 4$. To obtain the similar result for any $C > 4$ we have to slightly modify the construction from Fig. 1. We have to add $C - 4$ states between each pair of states $l'$ and $l$ for $l \in \{s_1, \dots s_z, t, u, v, w, x, y, q\}$. These 'chains' of states enforce us to use a word $W \in AA^{C-4}$ of length $C-3$ before we reach the set $\mathcal{V}.W = D \cup E \cup \bigcup_{i=1}^{z} F_i$ which can be still synchronized by the word of length 3 if and only if the corresponding formula $\varphi$ is satisfiable. This reasoning leads us to the following result:

**Theorem 8.** *$SRCP_C^3$ is **NP**-complete for any $C \geq 4$.*

Now it remains to show the **NP**-completeness for arbitrary $\ell \geq 3$. Let $|A| = \ell > 3$. Take the construction from Fig. 1 and add $\ell-3$ edges to each state in the following way: for each $c_i$, $i = 1, \dots, z$, add $\ell-3$ edges $(c_i, l)$, where $l$ is an arbitrarily chosen literal from clause $c_i$; for each $l \in \{r_1, s_1, \dots, r_z, l_z, t, u, v, w, x, y, q\}$ add $\ell - 3$ edges $(l', l)$; for each vertex $l$ of $\overline{G}$ add $\ell - 3$ edges $(l, next(l))$, where $next(l)$ is a vertex of $\overline{G}$ such that $(l, next(l))$ is already an edge in $G$; add $\ell - 3$ edges $(y, q)$ and $(q, x)$; for each of remaining states add $\ell - 3$ edges going to $k$. It is easy to see that the whole reasoning from the proof of Theorem 7 remains unchanged. This allows us to formulate the main theorem of this section:

**Theorem 9.** *$SRCP_C^\ell$ is **NP**-complete for $C \geq 4$ and $\ell \geq 3$.*

## 4 Complexity of SRCP$_C^\ell$ for $C = 1, 2$ and $\ell \geq 1$

In this section we deal with the complexity of SRCP$_C^\ell$ for $C = 1, 2$ and arbitrary $\ell$. Let $G = (\mathcal{V}, \mathcal{E})$ be a directed graph. For $v \in \mathcal{V}$ define the pre-image of vertex $v$ as $v^{-1} = \{w \in \mathcal{V} : \ (w, v) \in \mathcal{E}\}$ and extend this notion to pre-images for sets: for $V \subset \mathcal{V}$ let $V^{-1} = \bigcup_{v \in V} v^{-1}$. For $i > 1$ define $v^{-i} = (v^{-1})^{-(i-1)}$ and $V^{-i} = (V^{-1})^{-(i-1)}$.

First let us consider some easy boundary cases.

**Proposition 10.** *$SCRP_C^\ell$ is in **P** for $C = 1$ or $\ell = 1$.*

*Proof.* Notice that we consider only strongly connected digraphs. If $\ell = 1$, then $G$ is a cyclic graph and the synchronizing word of any length for some labeling of $G$ exists only for graphs with $|\mathcal{V}| = 1$. If $C = 1$, then the synchronizing word of length 1 exists for some labeling of $G = (\mathcal{V}, \mathcal{E})$ if and only if there exists $v \in \mathcal{V}$ such that $v^{-1} = \mathcal{V}$. Both conditions can be checked in polynomial time. $\square$.

The necessary condition from Lemma 1 (for the existence of synchronizing word of length $C$ for some labeling of a given graph $G$) can be expressed equivalently using the notion of pre-images:

$$\exists v \in \mathcal{V} : \ v^{-C} = \mathcal{V}. \tag{2}$$

Before we go to case $\ell > 1$, notice that to show that a problem is in **P** it suffices to give an algorithm solving any instance of the problem in polynomial time. We will use the following, general algorithm scheme for checking if there exists a synchronizing word of some length for some labeling of graph $G$:

```
     FIND-SYNCHR-LABELING
 in: $G = (\mathcal{V}, \mathcal{E})$, integer $C$
out: YES, if there exists a synchronizing word of length $C$ for some labeling of
     $G$; NO, otherwise
  1. foreach $v \in \mathcal{V}$ such that $v \in v^{-1}$ do
     1.1. // assume that the synchronization will take place in $v$
     1.2. if condition (2) is fulfilled for $v$ and $C$ then return YES // Prop. 11
  2. foreach $v \in \mathcal{V}$ such that $v \notin v^{-1}$ and condition (2) is fulfilled for $v$ and $C$
     do
     2.1. // assume that the synchronization will take place in $v$
     2.2. if CHECK-STATE$(G, v, C)$ = YES then return YES
  3. return NO
```

It is obvious that condition (2) can be checked in time polynomial in size of $G$ and $C$. Procedure CHECK-STATE$(G, v, C)$ checks if $G$ can be synchronized to $v$ with a word of length $C$ for some labeling of $G$. This procedure will be different for different $C$'s. Notice that it may be called only for vertices with no loops. For proving the correctness of FIND-SYNCHR-LABELING when it returns YES in line 1.2. it is enough to show that condition (2) is a necessary and sufficient condition in case $v \in v^{-1}$:

**Proposition 11.** *Let $G = (\mathcal{V}, \mathcal{E})$ be a graph and let $v \in \mathcal{V}$ such that $v \in v^{-1}$. There exists a labeling $L$ of $G$ such that $\mathcal{V}$ can be synchronized to $v$ by a word of length $C$ if and only if condition (2) holds for $v$.*

*Proof.* "$\Rightarrow$" implication is trivial. We prove the "$\Leftarrow$" one. Let (2) holds. Consider the following labeling:

$$v.a = v,$$

$$\forall w \in v^{-i} \setminus \bigcup_{j=0}^{i-1} v^{-j},\ i = 1, ..., C,\ w.a = x \text{ s.t. } (w, x) \in \mathcal{E} \wedge x \in v^{-(i-1)} \setminus \bigcup_{j=0}^{i-2} v^{-j},$$

where $v^0 := \{v\}$. All other edges are labeled arbitrarily. It is clear that for this labeling we have $\mathcal{V}.a^C = v$. □

The labeling procedure in Proposition 11 is just a BFS algorithm that starts in $v$, enqueue it and while the queue $Q$ is not empty it takes vertex $w$ from $Q$, goes to all non-visited vertices $w' \in w^{-1}$, labels $(w', w)$ with $a$ and enqueues $w'$. Because of (2), we know that in the BFS procedure we will reach each vertex of $G$ in at most $C$ steps starting from $v$.

Now we return to the case $\ell > 1$.

**Theorem 12.** *$SCRP_C^\ell$ is in **P** for $C = 2$ and any $\ell > 1$.*

*Proof.* We use Find-Synchr-Labeling with the following Check-State$(G, v)$ procedure for $C = 2$:

```
     Check-State
in:  $G = (\mathcal{V}, \mathcal{E})$, $v \in \mathcal{V}$ such that $v \notin v^{-1}$
out: YES, if there exists $W \in A^2$, such that $\mathcal{V}.W = v$ for some labeling of $G$; NO,
     otherwise
  1. if condition (2) is fulfilled then return YES
  2. else return NO
```

Correctness of Check-State. Let $E_1 = \{(u, v) : u \in v^{-1}\}$, $E_2 = \{(u, w) : u \in v^{-2} = \mathcal{V}, w \in v^{-1}\}$. We have $E_1 \cap E_2 = \emptyset$, because $v \notin v^{-1}$. Define labeling $\delta$ as follows: for each $w \in \mathcal{V}$ put $w.a = u$ for some $u \in v^{-1}$ and for each $u \in v^{-1}$ put $u.b = v$. All other edges can be labeled arbitrarily. It is clear that $W = ab$ synchronizes $G$ with labeling $\delta$. □

## 5 Conclusions and Future Work

Table 1 summarizes our results on the complexities of SRCP$^{\ell}_{C}$ for different word lengths $C$ and alphabet sizes $\ell$. Question marks denote problem versions for which we do not know the complexity.

**Table 1.** Synchronizing-Road-Coloring$^{\ell}_{C}$ complexities for different $C$ and $\ell$

| | $C = 1$ | $C = 2$ | $C = 3$ | $C \geq 4$ |
|---|---|---|---|---|
| $\ell = 1$ | **P** (Prop. 10) | **P** (Prop. 10) | **P** (Prop. 10) | **P** (Prop. 10) |
| $\ell = 2$ | **P** (Prop. 10) | **P** (Th. 12) | ? | ? |
| $\ell = 3$ | **P** (Prop. 10) | **P** (Th. 12) | ? | **NPC** (Th. 8) |
| $\ell \geq 4$ | **P** (Prop. 10) | **P** (Th. 12) | ? | **NPC** (Th. 9) |

## References

1. Adler, R.L., Goodwyn, L.W., Weiss, B.: Equivalence of topological Markov shifts. Israel J. of Math. 27, 49–63 (1977)
2. Adler, R.L., Weiss, B.: Similarity of automorphisms of the torus. Memoirs of the Amer. Math. Soc. 98 (1970)
3. Béal, M.P., Perrin, D.: A quadratic algorithm for road coloring, arXiv:0803.0726v6 (2008)
4. Eppstein, D.: Reset sequences for monotonic automata. SIAM J. of Computing 19, 500–510 (1990)
5. Roman, A.: NP-completeness of the road coloring problem. Information Processing Letters 111, 342–347 (2011)
6. Trahtman, A.: A subquadratic algorithm for road coloring, arXiv:0801.2838v1 (2008)
7. Trahtman, A.N.: Road coloring problem. Israel J. of Mathematics 172, 51–60 (2009)
8. Volkov, M.: Open problems on synchronizing automata. In: Conference 'Around the Černý Conjecture' (2008)
9. Volkov, M.: Synchronizing automata and the road coloring theorem. In: Tutorial on Workshop on Algebra, Combinatorics and Complexity (2008)

# $k$-Automatic Sets of Rational Numbers

Eric Rowland and Jeffrey Shallit

University of Waterloo, Waterloo, ON N2L 3G1 Canada
`erowland@uwaterloo.ca, shallit@cs.uwaterloo.ca`

**Abstract.** The notion of a $k$-automatic set of integers is well-studied. We develop a new notion — the $k$-automatic set of rational numbers — and prove basic properties of these sets, including closure properties and decidability.

## 1 Introduction

Let $k$ be an integer $\geq 2$, and let $\mathbb{N} = \{0, 1, 2, \ldots\}$ denote the set of non-negative integers. Let $\Sigma_k = \{0, 1, \ldots, k-1\}$ be an alphabet. Given a word $w = a_1 a_2 \cdots a_t \in \Sigma_k^*$, we let $[w]_k$ denote the integer that it represents in base $k$; namely,

$$[w]_k = \sum_{1 \leq i \leq t} a_i k^{t-i}, \tag{1}$$

where (as usual) an empty sum is equal to 0. For example, $[101011]_2 = 43$.

Note that in this framework, every element of $\mathbb{N}$ has infinitely many distinct representations as words, each having a certain number of *leading zeroes*. Among all such representations, the one with no leading zeroes is called the *canonical representation*; it is an element of $C_k := \{\epsilon\} \cup (\Sigma_k - \{0\})\Sigma_k^*$. For an integer $n \geq 0$, we let $(n)_k$ denote its canonical representation.

Given a language $L \subseteq \Sigma_k^*$, we can define the set of integers it represents, as follows:

$$[L]_k = \{[w]_k : w \in L\}. \tag{2}$$

**Definition 1.** *We say that a set $S \subseteq \mathbb{N}$ is $k$-automatic if there exists a regular language $L \subseteq \Sigma_k^*$ such that $S = [L]_k$.*

The class of $k$-automatic sets of natural numbers has been widely studied (e.g., [3,4,1]), and many properties of these sets are known. For example, it is possible to state an equivalent definition involving only canonical representations:

**Definition 2.** *A set $S \subseteq \mathbb{N}$ is $k$-automatic if the language $(S)_k := \{(n)_k : n \in S\}$ is regular.*

To see the equivalence of Definitions 1 and 2, note that if $L$ is a regular language, then so is the language $L'$ obtained by removing all leading zeroes from each word in $L$.

A slightly more general concept is that of $k$-automatic *sequence*. Let $\Delta$ be a finite alphabet. Then a sequence (or infinite word) $(a_n)_{n\geq 0}$ over $\Delta$ is said to be $k$-automatic if, for every $c \in \Delta$, the set of *fibers* $F_c = \{n \in \mathbb{N} : a_n = c\}$ is a $k$-automatic set of natural numbers. Again, this class of sequences has been widely studied [3,4,1]. The following result is well-known [4]:

**Theorem 3.** *The sequence $(a_n)_{n\geq 0}$ is $k$-automatic iff its $k$-kernel, the set of its subsequences $K = \{(a_{k^e n+f})_{n\geq 0} : e \geq 0,\ 0 \leq f < k^e\}$, is finite.*

In a previous paper [11], the second author extended the notion of $k$-automatic sets over $\mathbb{N}$ to sets over $\mathbb{Q}^{\geq 0}$, the set of non-negative rational numbers. In this paper, we will obtain some basic results about this new class. Our principal results are Theorem 13 (characterizing the sets of integers), Corollary 18 (showing that $k$-automatic sets of rationals are not closed under intersection), and Theorem 23 (showing that it is decidable if a $k$-automatic set is infinite).

Our class has some similarity to another class studied by Even [5] and Hartmanis and Stearns [6]; their class corresponds to the topological closure of our $k$-automatic sets, where the denominators are restricted to powers of $k$.

## 2 Representing Rational Numbers

A natural representation for the non-negative rational number $p/q$ is the pair $(p, q)$ with $q \neq 0$. Of course, this representation has the drawback that every element of $\mathbb{Q}^{\geq 0}$ has infinitely many representations, each of the form $(jp/d, jq/d)$ for some $j \geq 1$, where $d = \gcd(p, q)$.

We might try to ensure uniqueness of representations by considering only "reduced" representations (those in "lowest terms"), which amounts to requiring that $\gcd(p, q) = 1$. However, this condition cannot be checked by automata — see Remark 19 below — and deciding if an arbitrary regular language consists entirely of reduced representations is not evidently computable. Furthermore, insisting on reduced representations rules out the representation of some reasonable sets of rationals, such as $\{(k^m - 1)/(k^n - 1) : m, n \geq 1\}$ (see Theorem 17). For these reasons, **we allow the rational number $p/q$ to be represented by *any* pair of integers $(p', q')$ with $p/q = p'/q'$.**

Next, we need to see how to represent a pair of integers as a word over a finite alphabet. Here, we follow the ideas of Salon [8,9,10]. Consider the alphabet $\Sigma_k^2$. A finite word $w$ over $\Sigma_k^2$ can be considered as a sequence of pairs $w = [a_1, b_1][a_2, b_2] \cdots [a_n, b_n]$ where $a_i, b_i \in \Sigma_k$ for $1 \leq i \leq n$. We can now define the projection maps $\pi_1$, $\pi_2$, as follows:

$$\pi_1(w) = a_1 a_2 \cdots a_n; \qquad \pi_2(w) = b_1 b_2 \cdots b_n.$$

Note that the domain and range of each projection are $(\Sigma_k^2)^*$ and $\Sigma_k^*$, respectively. Then we define $[w]_k = ([\pi_1(w)]_k, [\pi_2(w)]_k)$. Thus, for example, if $w = [1,0][0,1][1,0][0,0][1,1][1,0]$, then $[w]_2 = (43, 18)$.

In this framework, every pair of integers $(p, q)$ again has infinitely many distinct representations, arising from padding on the left by leading pairs of zeroes, that is, by $[0, 0]$. Among all such representations, the *canonical representation* is the one having no leading pairs of zeroes. We write it as $(p, q)_k$.

We now state the fundamental definitions of this paper:

**Definition 4.** *Given a word $w \in (\Sigma_k^2)^*$ with $[\pi_2(w)]_k \neq 0$, we define*

$$\mathrm{quo}_k(w) := \frac{[\pi_1(w)]_k}{[\pi_2(w)]_k}.$$

*If $[\pi_2(w)]_k = 0$, then $\mathrm{quo}_k(w)$ is not defined. Further, if $[\pi_2(w)]_k \neq 0$ for all $w \in L$, then $\mathrm{quo}_k(L) := \{\mathrm{quo}_k(w) : w \in L\}$. A set of rational numbers $S \subseteq \mathbb{Q}^{\geq 0}$ is* $k$-automatic *if there exists a regular language $L \subseteq (\Sigma_k^2)^*$ such that $\mathrm{quo}_k(L) = S$.*

Note that in our definition, a given rational number $\alpha$ can have multiple representations in two different ways: it can have non-canonical representations that begin with leading zeroes, and it can have "unreduced" representations $(p, q)$ where $\gcd(p, q) > 1$.

Given a set $S \subseteq \mathbb{Q}$, if $S$ contains a non-integer, then by calling it $k$-automatic, it is clear that we intend this to mean automatic in the sense of this section. But what if $S \subseteq \mathbb{N}$? Then calling it "automatic" might mean automatic in the usual sense, as in Section 1, or in the extended sense introduced in this section, treating $S$ as a subset of $\mathbb{Q}$. In Theorem 13 we will see that these two interpretations actually *coincide* for subsets of $\mathbb{N}$, but in order to prove this, we need some notation to distinguish between the two types of representations. So, by $(\mathbb{N}, k)$-automatic we mean the interpretation in Section 1 and by $(\mathbb{Q}, k)$-automatic we mean the interpretation in this section.

So far we have only considered representations where the leftmost digit is the most significant digit. However, sometimes it is simpler to deal with *reversed representations* where the leftmost digit is the least significant digit. In other words, sometimes it is easier to deal with the reversed word $w^R$ and reversed language $L^R$ instead of $w$ and $L$, respectively. Since the regular languages are (effectively) closed under reversal, for most of our results it does not matter which representation we choose, and we omit extended discussion of this point.

We use the following notation for intervals: $I[\alpha, \beta]$ denotes the closed interval $\{x : \alpha \leq x \leq \beta\}$, and similarly for open- and half-open intervals.

## 3 Examples

To build intuition, we give some examples of $k$-automatic sets of rationals.

*Example 5.* Let $k = 2$, and consider the regular language $L_1$ defined by the regular expression $A^*\{[0, 1], [1, 1]\}A^*$, where $A = \{[0, 0], [0, 1], [1, 0], [1, 1]\}$. This regular expression specifies all pairs of integers where the second component has at least one nonzero digit — the point being to avoid division by 0. Then $\mathrm{quo}_k(L) = \mathbb{Q}^{\geq 0}$, the set of all non-negative rational numbers.

*Example 6.* Consider the regular language

$$L_2 = \{w \in (\Sigma_k^2)^* \, : \, \pi_1(w) \in C_k \ \cup \ \{0\} \text{ and } [\pi_2(w)]_k = 1\}.$$

Then $\text{quo}_k(L_2) = \mathbb{N}$.

*Example 7.* Let $k = 3$, and consider the regular language $L_3$ defined by the regular expression $[0,1]\{[0,0],[2,0]\}^*$. Then $\text{quo}_k(L_2)$ is the 3-*adic Cantor set*, the set of all rational numbers in the "middle-thirds" Cantor set with denominators a power of 3 [2].

*Example 8.* Let $k = 4$, and consider the set $S = \{0, 1, 3, 4, 5, 11, 12, 13, \ldots\}$ of all non-negative integers that can be represented using only the digits $0, 1, -1$ in base 4. Consider the set $L_5 = \{(p, q)_4 \, : \, p, q \in S\}$. It is not hard to see that $L_5$ is $(\mathbb{Q}, 4)$-automatic. The main result in [7] can be phrased as follows: $\text{quo}_4(L_5)$ contains every odd integer. In fact, an integer $t$ is in $\text{quo}_4(L_5)$ if and only if the exponent of the largest power of 2 dividing $t$ is even.

*Example 9.* Note that $\text{quo}_k(L_1 \cup L_2) = \text{quo}_k(L_1) \cup \text{quo}_k(L_2)$ but the analogous identity involving intersection need not hold. As an example, consider $L_1 = \{[2,1]\}$ and $L_2 = \{[4,2]\}$. Then $\text{quo}_{10}(L_1 \cap L_2) = \emptyset \neq \{2\} = \text{quo}_{10}(L_1) \cap \text{quo}_{10}(L_2)$.

## 4 Basic Results

In this section we obtain some basic results about automatic sets of rationals. First we state a result from [11]:

**Theorem 10.** *Let $\beta$ be a non-negative real number and define*

$$L_{\leq\beta} = \{w \in (\Sigma_k^2)^* \, : \, \text{quo}_k(w) \leq \beta\},$$

*and analogously for the other relations $<, =, \geq, >, \neq$. Then $L_{\leq\beta}$ (resp., $L_{<\beta}$, $L_{=\beta}$, $L_{\geq\beta}$, $L_{>\beta}$) is regular iff $\beta$ is a rational number.*

**Lemma 11.** *Let $M$ be a DFA with input alphabet $\Sigma_k^2$ and let $F \subseteq \mathbb{Q}^{\geq 0}$ be a finite set of integers. Then the following problems are recursively solvable:*

1. *Is $F \subseteq \text{quo}_k(L(M))$?*
2. *Is $\text{quo}_k(L(M)) \subseteq F$?*

*Proof.* To decide if $F \subseteq \text{quo}_k(L(M))$, we simply check, using Theorem 10, whether $x \in \text{quo}_k(L(M))$ for each $x \in F$.

To decide if $\text{quo}_k(L(M)) \subseteq F$, we create DFA's accepting $L_{=x}$ for each $x \in F$, using Theorem 10. Now we create a DFA accepting all representations of all elements of $F$ using the usual "direct product" construction for the union of regular languages. Since $F$ is finite, the resulting DFA $A$ is finite. Now, using the usual direct product construction, we create a DFA accepting $L(A) - L(M)$ and check to see if its language is nonempty. $\square$

Suppose $S$ is a set of real numbers, and $\alpha$ is a real number. We introduce the following notation:

$$S + \alpha := \{x + \alpha \,:\, x \in S\} \qquad S \dot{-} \alpha := \{\max(x - \alpha, 0) \,:\, x \in S\}$$
$$\alpha \dot{-} S := \{\max(\alpha - x, 0) \,:\, x \in S\} \qquad \alpha S := \{\alpha x \,:\, x \in S\}.$$

**Theorem 12.** *The class of $k$-automatic sets of rational numbers is closed under the following operations: (i) union; (ii) $S \rightarrow S + \alpha$ for $\alpha \in \mathbb{Q}^{\geq 0}$; (iii) $S \rightarrow S \dot{-} \alpha$ for $\alpha \in \mathbb{Q}^{\geq 0}$; (iv) $S \rightarrow \alpha \dot{-} S$ for $\alpha \in \mathbb{Q}^{\geq 0}$; (v) $S \rightarrow \alpha S$ for $\alpha \in \mathbb{Q}^{\geq 0}$; (vi) $S \rightarrow \{1/x \,:\, x \in S \setminus \{0\}\}$.*

*Proof.* We prove only item (2), with the others being similar. We will use the reversed representation, with the least significant digit appearing first. Write $\alpha = p/q$. Let $M$ be a DFA with $\text{quo}_k(L(M)) = S$. To accept $S + \alpha$, on input a base-$k$ representation of $x = p'/q'$, we transduce the numerator to $p'q - pq'$ and the denominator to $qq'$ (hence effectively computing a representation for $x - \alpha$), and simultaneously simulate $M$ on this input, digit-by-digit, accepting if $M$ accepts. $\square$

We now state one of our main results.

**Theorem 13.** *Let $S \subseteq \mathbb{N}$. Then $S$ is $(\mathbb{N}, k)$-automatic if and only if it is $(\mathbb{Q}, k)$-automatic.*

The proof requires a number of preliminary results. First, we introduce some terminology and notation. We say a set $S \subseteq \mathbb{N}$ is *ultimately periodic* if there exist integers $n_0 \geq 0, p \geq 1$ such that $n \in S \iff n + p \in S$, provided $n \geq n_0$. In particular, every finite set is ultimately periodic. We let $\mathcal{P} = \{2, 3, 5, \ldots\}$ denote the set of prime numbers. Given a positive integer $n$, we let $\text{pd}(n)$ denote the set of its prime divisors. For example, $\text{pd}(60) = \{2, 3, 5\}$. Given a subset $D \subset \mathcal{P}$, we let $\pi(D) = \{n \geq 1 \,:\, \text{pd}(n) \subseteq D\}$, the set of all integers that can be factored completely using only the primes in $D$. Finally, let $k \geq 2$, $n \geq 1$ be integers, and define $\nu_k(n) := \max\{i \,:\, k^i \mid n\}$, the exponent of the largest power of $k$ dividing $n$.

The first result we need is the following theorem, which is of independent interest.

**Theorem 14.** *Let $D \subseteq \mathcal{P}$ be a finite set of prime numbers, and let $S \subseteq \pi(D)$. Then $S$ is $k$-automatic iff*

*1. $F := \{\frac{s}{k^{\nu_k(s)}} \,:\, s \in S\}$ is finite*
*and*
*2. For all $f \in F$ the set $U_f = \{i \,:\, k^i f \in S\}$ is ultimately periodic.*

First, we need a lemma.

**Lemma 15.** *Let $D$ be a finite set of prime numbers, and let $S \subseteq \pi(D)$. Let $s_1, s_2, \ldots$ be an infinite sequence of (not necessarily distinct) elements of $S$. Then there is an infinite increasing sequence of indices $i_1 < i_2 < \cdots$ such that $s_{i_1} \mid s_{i_2} \mid \cdots$.*

*Proof.* Case 1: The sequence $(s_i)$ is bounded. In this case infinitely many of the $s_i$ are the same, so we can take the indices to correspond to these $s_i$.

Case 2: The sequence $(s_i)$ is unbounded. In this case we prove the result by induction on $|D|$. If $|D| = 1$, then we can choose a strictly increasing subsequence of the $(s_i)$; since all are powers of some prime $p$, this subsequence has the desired property.

Now suppose the result is true for all sets $D$ of cardinality $t-1$. We prove it for $|D| = t$. Since only $t$ distinct primes figure in the factorization of the $s_i$, some prime, say $p$, must appear with unbounded exponent in the $(s_i)$. So there is some subsequence of $(s_i)$, say $(t_i)$, with strictly increasing exponents of $p$. Now consider the infinite sequence $(u_i)$ given by $u_i = t_i/p^{\nu_p(t_i)}$. Each $u_i$ has a prime factorization in terms of the primes in $D - \{p\}$, so by induction there is an infinite increasing sequence of indices $i_1, i_2, \ldots$ such that $u_{i_1} \mid u_{i_2} \mid \cdots$. Then $p^{\nu_p(t_{i_1})}u_{i_1} \mid p^{\nu_p(t_{i_2})}u_{i_2} \mid \cdots$, which corresponds to an infinite increasing sequence of indices of the original sequence $(s_i)$. □

Now we can prove Theorem 14.

*Proof.* $\Longleftarrow$: $S$ can be written as the disjoint finite union $\bigcup_{f\in F} f \cdot \{k^i : i \in U_f\}$, where $U_f$ is an ultimately periodic set of integers. Each term in the union has base-$k$ representation $(f)_k\{0^i : i \in U_f\}$ and hence is regular. It follows that $S$ is $k$-automatic.

$\Longrightarrow$: Suppose $F$ is infinite. Notice that, from the definition, no element of $F$ is divisible by $k$. Therefore we can write $S$ as the disjoint union $\bigcup_{i\geq 0} k^i H_i$, where $H_i := \{f \in F : k^i f \in S\}$. Then there are two possibilities: either (a) the sets $H_i$ are finite for all $i \geq 0$, or (b) at least one $H_i$ is infinite.

In case (a), define $u_i := \max H_i$ for all $i \geq 0$. Then the set $\{u_0, u_1, \ldots\}$ must be infinite, for otherwise $F$ would be finite. Choose an infinite subsequence of the $u_i$ consisting of distinct elements, and apply Lemma 15. Then there is an infinite increasing subsequence of indices $i_1 < i_2 < \cdots$ such that $u_{i_1} \mid u_{i_2} \mid \cdots$. So the sequence $(u_{i_j})_{j\geq 1}$ is strictly increasing.

Now consider the characteristic sequence of $S$, say $(f(n))_{n\geq 0}$, taking values 1 if $n \in S$ and 0 otherwise. Consider the subsequences $(f_j)$ in the $k$-kernel of $f$ defined by $f_j(n) = f(k^{i_j} n)$ for $n \geq 0$, $j \geq 1$. By our construction, the largest $n$ in $\pi(D)$ such that $k \nmid n$ and $f_j(n) = 1$ is $n = u_{i_j}$. Since the $u_{i_j}$ are strictly increasing, this shows the (infinitely many) sequences $(f_j)$ are pairwise distinct. Hence, by Theorem 3, $f$ is not $k$-automatic and neither is $S$.

In case (b), we have $H_i$ is infinite for some $i$. As mentioned above, $S$ can be written as the disjoint union $\bigcup_{i\geq 0} k^i H_i$. Let $L$ be the language of canonical base-$k$ expansions of elements of $H_i$ (so that, in particular, no element of $L$ starts with 0). The base-$k$ representation of elements of $k^i H_i$ end in exactly $i$ 0's, and no other member of $S$ has this property. Since $S$ is assumed to be $k$-automatic, it follows that $L$ is regular. Note that no two elements of $H_i$ have a quotient which is divisible by $k$, because if they did, the numerator would be divisible by $k$, which is ruled out by the condition.

Since $L$ is infinite and regular, by the pumping lemma, there must be words $u, v, w$, with $v$ nonempty, such that $uv^jw \in L$ for all $j \geq 0$. Note that for all integers $j \geq 0$ and $c \geq 0$ we have

$$[uv^{j+c}w]_k = [uv^jw]_k \cdot k^{c|v|} + ([v^cw]_k - [w]_k \cdot k^{c|v|}). \tag{3}$$

Let $D = \{p_1, p_2, \ldots, p_t\}$. Since $[uv^jw]_k \in S \subseteq \pi(D)$, it follows that there exists a double sequence $(f_{r,j})_{1 \leq r \leq t; j \geq 1}$ of non-negative integers such that

$$[uv^jw]_k = p_1^{f_{1,j}} \cdots p_t^{f_{t,j}} \tag{4}$$

for all $j \geq 0$. From (4), we see that $k^{|uw|+j|v|} < p_1^{f_{1,j}} \cdots p_t^{f_{t,j}}$, and hence (assuming $p_1 < p_2 < \cdots < p_t$) we get

$$(|uw| + j|v|) \log k < (\sum_{1 \leq r \leq t} f_{r,j}) \log p_t.$$

Therefore, there are constants $0 < c_1$ and $J$ such that $c_1 j < \sum_{1 \leq r \leq t} f_{r,j}$ for $j \geq J$.

For each $j \geq J$ now consider the indices $r$ such that $f_{r,j} > c_1 j/t$; there must be at least one such index, for otherwise $f_{r,j} \leq c_1 j/t$ for each $r$ and hence $\sum_{1 \leq r \leq t} f_{r,j} \leq c_1 j$, a contradiction. Now consider $t + 1$ consecutive $j$'s; for each $j$ there is an index $r$ with $f_{r,j} > c_1 j/t$, and since there are only $t$ possible indices, there must be an $r$ and two integers $l'$ and $l$, with $0 \leq l < l' \leq t$, with $f_{r,j+l} > c_1(j + l)/t$ and $f_{r,j+l'} > c_1(j + l')/t$. This is true in any block of $t + 1$ consecutive $j$'s that are $\geq J$. Now there are infinitely many disjoint blocks of $t+1$ consecutive $j$'s, and so there must be a single $r$ and a single difference $l' - l$ that occurs infinitely often. Put $\delta = l' - l$.

Now use (3) and take $c = \delta$. We get infinitely many $j$ such that

$$p_1^{f_{1,j+\delta}} \cdots p_t^{f_{t,j+\delta}} = k^{\delta|v|} p_1^{f_{1,j}} \cdots p_t^{f_{t,j}} + E,$$

where $E = [v^\delta w]_k - [w]_k \cdot k^{\delta|v|}$ is a constant that is independent of $j$. Now focus attention on the exponent of $p_r$ on both sides. On the left it is $f_{r,j+\delta}$, which we know to be at least $c_1(j+\delta)/t$. On the right the exponent of $p_r$ dividing the first term is $f_{r,j} + \delta|v|e_r$ (where $k = p_1^{e_1} \cdots p_t^{e_t}$); this is at least $c_1 j/t$. So $p_r^h$ divides $E$, where $h \geq c_1 j/t$. But this quantity goes to $\infty$, and $E$ is a constant. So $E = 0$. But then

$$\frac{[uv^{j+\delta}w]_k}{[uv^jw]_k} = k^{\delta|v|}.$$

which is impossible, since, as we observed above, two elements of $H_i$ cannot have a quotient which is a power of $k$. This contradiction shows that $H_i$ cannot be infinite.

So now we know that $F$ is finite. Fix some $f \in F$ and consider $T_f = \{k^i : k^i f \in S\}$. Since $S$ is $k$-automatic, and the set of base-$k$ expansions $(T_f)_k$ essentially is formed by stripping off the bits corresponding to $(f)_k$ from the front of each element of $S$ of which $(f)_k$ is a prefix, and replacing it with "1", this is just a left quotient followed by concatenation, and hence $(T_f)_k$ is regular. Let $M'$ be a DFA for $(T_f)_k$, and consider an input of 1 followed by $l$ 0's for $l = 0, 1, \ldots$ in $M'$. Evidently we eventually get into a cycle, so this says that $U_f = \{i : k^i f \in S\}$ is ultimately periodic.

This completes the proof of Theorem 14. $\square$

**Corollary 16.** *Suppose $S$ is a $k$-automatic set of integers accepted by a finite automaton $M$. There is an algorithm to decide, given $M$, whether there exists a finite set $D \subseteq \mathcal{P}$ such that $S \subseteq \pi(D)$. Furthermore, if such a $D$ exists, we can also determine the sets $F$ and $U_f$ in Theorem 14.*

*Proof.* To determine if such a $D$ exists, it suffices to remove all trailing zeroes from words in $(S)_k$ and see if the resulting language is finite. If it is, we know $F$, and then it is a simple matter to compute the $U_f$. $\square$

We can now prove Theorem 13.

*Proof.* One direction is easy, since if $S$ is $(\mathbb{N}, k)$-automatic, then there is an automaton accepting $(S)_k$. We can now easily modify this automaton to accept all words over $(\Sigma_k^2)^*$ whose $\pi_2$ projection represents the integer 1 and whose $\pi_1$ projection is an element of $(S)_k$. Hence $S$ is $(\mathbb{Q}, k)$-automatic.

Now assume $S \subseteq \mathbb{N}$ is $(\mathbb{Q}, k)$-automatic. If $S$ is finite, then the result is clear, so assume $S$ is infinite. Let $L$ be a regular language with $\text{quo}_k(L) = S$. Without loss of generality we may assume every representation in $L$ is canonical; there are no leading $[0, 0]$'s. Furthermore, by first intersecting with $L_{\neq 0}$ we may assume that $L$ contains no representations of the integer 0. Finally, we can also assume, without loss of generality, that no representation contains *trailing* occurrences of $[0, 0]$, for removing all trailing zeroes from all words in $L$ preserves regularity, and it does not change the set of numbers represented, as it has the effect of dividing the numerator and denominator by the same power of $k$. Since the words in $L$ represent integers only, the denominator of every representation must divide the numerator, and hence if the denominator is divisible by $k$, the numerator must also be so divisible. Hence removing trailing zeroes *also* ensures that no denominator is divisible by $k$. Let $M$ be a DFA of $n$ states accepting $L$.

We first show that the set of possible denominators represented by $L$ is actually finite. Write $S = S_1 \cup S_2$, where $S_1 = S \cap I[0, k^{n+1})$ and $S_2 = S \cap I[k^{n+1}, \infty)$. Let $L_1 = L \cap L_{<k^{n+1}}$, the representations of all numbers $< k^{n+1}$ in $L$, and $L_2 = L \cap L_{\geq k^{n+1}}$. Both $L_1$ and $L_2$ are regular, by Theorem 10. It now suffices to show that $S_2$ is $(\mathbb{N}, k)$-automatic.

Consider any $t \in S_2$. Let $z \in L_2$ be a representation of $t$. Since $t \geq k^{n+1}$, clearly $|z| \geq n$, and so $\pi_2(z)$ must begin with at least $n$ 0's. Then, by the pumping lemma, we can write $z = uvw$ with $|uv| \leq n$ and $|v| \geq 1$ such that $uv^iw \in L$ for all $i \geq 0$. However, by the previous remark about $\pi_2(z)$, we see that $\pi_2(v) = 0^j$ for $1 \leq j \leq n$. Hence $[\pi_2(z)]_k = [\pi_2(uvw)]_k = [\pi_2(uw)]_k$. Since $uw$ must also represent a member of $S$, it must be an integer, and hence $[\pi_2(z)]_k \mid [\pi_1(uw)]_k$ as well as $[\pi_2(z)]_k \mid [\pi_1(uvw)]_k$. Hence

$$[\pi_2(z)]_k \mid [\pi_1(uvw)]_k - [\pi_1(uw)]_k = ([\pi_1(uv)]_k - [\pi_1(u)]_k) \cdot k^{|w|}.$$

The previous reasoning applies to any $z \in L_2$. Furthermore, $0 < [\pi_1(uv)]_k - [\pi_1(u)]_k < k^n$. It follows that every possible denominator $d$ of elements in $L_2$ can be expressed as $d = d_1 \cdot d_2$, where $1 \le d_1 < k^n$ and $d_2 \mid k^m$ for some $m$. It follows that the set of primes dividing all denominators $d$ is finite, and we can therefore apply Theorem 14. Since $k$ divides no denominator, the set of possible denominators is finite.

We can therefore decompose $L_2$ into a finite disjoint union corresponding to each possible denominator $d$. Next, we use a finite-state transducer to divide the numerator and denominator of the corresponding representations by $d$. For each $d$, this gives a new regular language $A_d$ where the denominator is 1. Writing $T := \bigcup_d A_d$, we have $S_2 = \text{quo}_k(T) = \bigcup_d \text{quo}_k(A_d)$. Now we project, throwing away the second coordinate of elements of $T$; the result is regular and hence $S$ is a $k$-automatic set of integers. □

As corollaries, we get that the $k$-automatic sets of rationals are (in contrast with sets of integers) not necessarily closed under the operations of intersection and complement.

**Theorem 17.** *Let $S_1 = \{(k^n - 1)/(k^m - 1) : 1 \le m < n\}$ and $S_2 = \mathbb{N}$. Then $S_1 \cap S_2$ is not a $k$-automatic set.*

*Proof.* We can write every element of $S_1$ as $p/q$, where $p = (k^n - 1)/(k - 1)$ and $q = (k^m - 1)/(k - 1)$. The base-$k$ representation of $p$ is $1^n$ and the base-$k$ representation of $q$ is $1^m$. Thus a representation for $S_1$ is given by the regular expression $[1, 0]^+[1, 1]^+$. We know that $\mathbb{N}$ is $k$-automatic from Example 6.

From a classical result we know that $(k^m - 1) \mid (k^n - 1)$ if and only if $m \mid n$. It follows that $S_1 \cap S_2 = T$, where $T = \{(k^n - 1)/(k^m - 1) : 1 < m < n \text{ and } m \mid n\}$. If the $k$-automatic sets of rationals were closed under intersection, then $T$ would be $(\mathbb{N}, k)$-automatic. Writing $n = md$, we have $(k^n - 1)/(k^m - 1) = k^{(d-1)m} + \cdots + k^m + 1$, whose base-$k$ representation is $(10^{m-1})^{d-1}1$. Hence $(T)_k = \{(10^{m-1})^{d-1}1 : m \ge 1, d > 1\}$. Assume this is regular. Intersecting with the regular language $10^*10^*10^*1$ we get $\{10^n10^n10^n1 : n \ge 1\}$. But a routine argument using the pumping lemma shows this is not even context-free, a contradiction. □

**Corollary 18.** *The class of $(\mathbb{Q}, k)$-automatic sets is not closed under the operations of intersection or complement.*

*Proof.* We have just shown that this class is not closed under intersection. But since it is closed under union, if it were closed under complement, too, it would be closed under intersection, a contradiction. □

*Remark 19.* The technique above also allows us to prove that the languages $L_d = \{(p, q)_k : q \mid p\}$, $L_r = \{(p, q)_k : \gcd(p, q) > 1\}$, and $L_g = \{(p, q)_k : \gcd(p, q) = 1\}$ are not context-free.

We can also prove a decidability result.

**Theorem 20.** *Given a regular language $L \subseteq (\Sigma_k^2)^*$, it is decidable whether*

$$\text{(a)} \quad \text{quo}_k(L) \subseteq \mathbb{N}; \qquad \text{(b)} \quad \text{quo}_k(L) \setminus \mathbb{N} \textit{ is finite.}$$

*Proof.* For (a), given $M$ accepting $L$, we first create (using Theorem 10) a DFA $M_1$ accepting the set $T_1 := (\text{quo}_k(L) \cap I[0, k^{n+1})) \setminus \{0, 1, \ldots, k^{n+1} - 1\}$. If $T_1 \neq \emptyset$, then answer "no" and stop. Next, create a DFA $M_2$ accepting the set $T_2 := (\text{quo}_k(L) \cap I[k^{n+1}, \infty))$. Now project onto the second coordinate, and, using Corollary 16, decide if the integers so represented are factorable into a finite set of primes. If not, answer "no" and stop. Finally, using Corollary 16 again, create the decomposition in that corollary (resulting in only finitely many denominators) and check for each denominator whether it divides all of its corresponding numerators, using a transducer.

For (b) we need to check instead whether $T_1$ is a finite set, and whether there are finitely many exceptions to the second and third steps. □

## 5 Solvability Results

In this section we show that a number of problems involving $k$-automatic sets of rational numbers are recursively solvable. We start with a useful lemma.

**Lemma 21.** *Let $u, v, w \in (\Sigma_k^2)^*$ such that $|v| \geq 1$, and such that $[\pi_1(uvw)]_k$ and $[\pi_2(uvw)]_k$ are not both $0$. Define*

$$U := \begin{cases} \text{quo}_k(w), & \textit{if } [\pi_1(uv)]_k = [\pi_2(uv)]_k = 0; \\ \infty, & \textit{if } [\pi_1(uv)]_k > 0 \textit{ and } [\pi_2(uv)]_k = 0; \\ \frac{[\pi_1(uv)]_k - [\pi_1(u)]_k}{[\pi_2(uv)]_k - [\pi_2(u)]_k}, & \textit{otherwise.} \end{cases} \tag{5}$$

*(a) Then exactly one of the following cases occurs:*

*(i)* $\text{quo}_k(uw) < \text{quo}_k(uvw) < \text{quo}_k(uv^2w) < \cdots < U$ ;
*(ii)* $\text{quo}_k(uw) = \text{quo}_k(uvw) = \text{quo}_k(uv^2w) = \cdots = U$ ;
*(iii)* $\text{quo}_k(uw) > \text{quo}_k(uvw) > \text{quo}_k(uv^2w) > \cdots > U$ .

*(b) Furthermore,* $\lim_{i\to\infty} \text{quo}_k(uv^iw) = U$.

*Proof.* Part (a) is proved in [11]. We now prove the assertion about the limit.

Let $j \in \{1, 2\}$, let $i$ be an integer $\geq 1$, and consider the base-$k$ representation of the rational number

$$\frac{[\pi_j(uv^iw)]_k}{k^{i|v|+|w|}}; \qquad \text{it looks like} \qquad \pi_j(u).\overbrace{\pi_j(v)\pi_j(v)\cdots\pi_j(v)}^{i}.$$

On the other hand, the base-$k$ representation of

$$[\pi_j(u)]_k + \frac{[\pi_j(v)]_k}{k^{|v|} - 1} \qquad \text{looks like} \qquad \pi_j(u).\pi_j(v)\pi_j(v)\cdots .$$

Subtracting, we get

$$\left| \frac{[\pi_j(uv^iw)]_k}{k^{i|v|+|w|}} - \left( [\pi_j(u)]_k + \frac{[\pi_j(v)]_k}{k^{|v|}-1} \right) \right| < k^{-i|v|}.$$

It follows that

$$\lim_{i\to\infty} \frac{[\pi_j(uv^iw)]_k}{k^{i|v|+|w|}} = [\pi_j(u)]_k + \frac{[\pi_j(v)]_k}{k^{|v|}-1}.$$

Furthermore, this limit is 0 if and only if $[\pi_j(uv)]_k = 0$. Hence, provided $[\pi_2(uv)]_k \neq 0$, we get

$$\lim_{i\to\infty} \mathrm{quo}_k(uv^iw) = \lim_{i\to\infty} \frac{[\pi_1(uv^iw)]_k}{[\pi_2(uv^iw)]_k} = \lim_{i\to\infty} \frac{\frac{[\pi_1(uv^iw)]_k}{k^{i|v|+|w|}}}{\frac{[\pi_2(uv^iw)]_k}{k^{i|v|+|w|}}} = \frac{\lim_{i\to\infty} \frac{[\pi_1(uv^iw)]_k}{k^{i|v|+|w|}}}{\lim_{i\to\infty} \frac{[\pi_2(uv^iw)]_k}{k^{i|v|+|w|}}}$$

$$= \frac{[\pi_1(u)]_k + \frac{[\pi_1(v)]_k}{k^{|v|}-1}}{[\pi_2(u)]_k + \frac{[\pi_2(v)]_k}{k^{|v|}-1}} = \frac{[\pi_1(uv)]_k - [\pi_1(u)]_k}{[\pi_2(uv)]_k - [\pi_2(u)]_k} = U.$$

□

We now give some solvability results.

**Theorem 22.** *The following problems are recursively solvable: given a DFA $M$, a rational number $\alpha$, and a relation $\lhd$ chosen from $=, \neq, <, \leq, >, \geq$, does there exist $x \in \mathrm{quo}_k(L(M))$ with $x \lhd \alpha$?*

*Proof.* The following gives a procedure for deciding if $x \lhd \alpha$. First, we create a DFA $M'$ accepting the language $L_{\lhd\alpha}$ as described in Theorem 10 above. Next, using the usual direct product construction, we create a DFA $M''$ accepting $L(M) \cap L_{\lhd\alpha}$. Then, using breadth-first or depth-first search, we check to see whether there exists a path from the initial state of $M''$ to some final state of $M''$. □

**Theorem 23.** *The following problem is recursively solvable: given a DFA $M$, and an integer $k$, is $\mathrm{quo}_k(L(M))$ of infinite cardinality?*

Note that this is *not* the same as asking whether $L(M)$ itself is of infinite cardinality, since a number may have infinitely many representations.

*Proof.* Without loss of generality, we may assume that the representations in $M$ are canonical (contain no leading $[0,0]$'s). Define

$$\gamma_k(u,v) = \frac{[\pi_1(uv)]_k - [\pi_1(u)]_k}{[\pi_2(uv)]_k - [\pi_2(u)]_k},$$

and let $\mathrm{pref}(L)$ denote the language of all prefixes of all words of $L$. Let $n$ be the number of states in $M$. We claim that $\mathrm{quo}_k(L(M))$ is of finite cardinality if and only if $\mathrm{quo}_k(L(M)) \subseteq T$, where

$$T = \{\mathrm{quo}_k(x) \,:\, x \in L(M) \text{ and } |x| < n\} \cup \{\gamma_k(u,v) \,:\, uv \in \mathrm{pref}(L) \text{ and } |v| \geq 1 \text{ and } |uv| \leq n\}. \quad (6)$$

One direction is easy, since if $\text{quo}_k(L(M)) \subseteq T$, then clearly $\text{quo}_k(L(M))$ is of finite cardinality, since $T$ is.

Now suppose $\text{quo}_k(L(M)) \subsetneq T$, so there exists some $x \in L(M)$ with $\text{quo}_k(x) \notin T$. Since $T$ contains all words of $L(M)$ of length $< n$, such an $x$ is of length $\geq n$. So the pumping lemma applies, and there exists a decomposition $x = uvw$ with $|uv| \leq n$ and $|v| \geq 1$ such that $uv^iw \in L$ for all $i \geq 0$. Now apply Lemma 21. If case (b) of that lemma applies, then $\text{quo}_k(x) = \gamma_k(u, v) \in T$, a contradiction. Hence either case (a) or case (c) must apply, and the lemma shows that $\text{quo}_k(uv^iw)$ for $i \geq 0$ gives infinitely many distinct elements of $\text{quo}_k(L(M))$.

To solve the decision problem, we can now simply enumerate the elements of $T$ and use Lemma 11. □

**Theorem 24.** *Given $p/q \in \mathbb{Q}^{\geq 0}$, and a DFA $M$ accepting a $k$-automatic set of rationals $S$, it is decidable if $p/q$ is an accumulation point of $S$.*

*Proof.* The number $\alpha$ is an accumulation point of a set of real numbers $S$ if and only if at least one of the following two conditions holds:

$$\text{(i)}\quad \alpha = \sup(\, S \,\cap\, I(-\infty, \alpha)\,); \qquad \text{(ii)}\quad \alpha = \inf(\, S \,\cap\, I(\alpha, \infty)\,).$$

By Theorem 10 we can compute a DFA accepting $S' := S \,\cap\, I(-\infty, \alpha)$ (resp., $S \,\cap\, I(\alpha, \infty)$). By [11, Thm. 2] we can compute $\sup S'$ (resp., $\inf S'$). □

## References

1. Allouche, J.P., Shallit, J.: Automatic Sequences: Theory, Applications, Generalizations. Cambridge University Press (2003)
2. Cantor, G.: Über unendliche, lineare Punktmannigfaltigkeiten V. Math. Annalen 21, 545–591 (1883)
3. Cobham, A.: On the base-dependence of sets of numbers recognizable by finite automata. Math. Systems Theory 3, 186–192 (1969)
4. Cobham, A.: Uniform tag sequences. Math. Systems Theory 6, 164–192 (1972)
5. Even, S.: Rational numbers and regular events. IEEE Trans. Electr. Comput. 13, 740–741 (1964)
6. Hartmanis, J., Stearns, R.E.: Sets of numbers defined by finite automata. Amer. Math. Monthly 74, 539–542 (1967)
7. Loxton, J.H., van der Poorten, A.J.: An awful problem about integers in base four. Acta Arith. 49, 193–203 (1987)
8. Salon, O.: Suites automatiques à multi-indices. Séminaire de Théorie des Nombres de Bordeaux, 4.01–4.27 (1986-1987)
9. Salon, O.: Suites automatiques à multi-indices et algébricité. C. R. Acad. Sci. Paris 305, 501–504 (1987)
10. Salon, O.: Propriétés arithmétiques des automates multidimensionnels. Ph.D. thesis, Université Bordeaux I (1989)
11. Shallit, J.: The critical exponent is computable for automatic sequences. In: Elect. Proc. Theor. Comput. Sci., vol. 63, pp. 231–239, ArXiv e-prints (2011), `http://arxiv.org/abs/1104.2303v2`

# On Stable and Unstable Limit Sets of Finite Families of Cellular Automata[⋆]

Ville Salo and Ilkka Törmä

University of Turku, Finland
{vosalo,iatorm}@utu.fi

**Abstract.** In this paper, we define the notion of limit set for a finite family of cellular automata, which is a generalization of the limit set of a single automaton. We prove that the hierarchy formed by increasing the number of automata in the defining set is infinite, and study the boolean closure properties of different classes of limit sets.

**Keywords:** Cellular Automata, Symbolic Dynamics, Limit Sets.

## 1 Introduction

Cellular automata are discrete dynamical systems that, despite their simple definition, can have very complex dynamics. They are defined as transformations on a space of configurations, that is, infinite sequences of symbols, that operate by applying the same local rule in every coordinate. The research of CA dates back to the 60's [2].

We are interested in the long-term time evolution of the configuration space under the dynamics given by cellular automata, which can be studied using the concept of limit sets. The limit set of a cellular automaton consists of those configurations that can appear arbitrarily late in the evolution of the system. A CA is called stable if its evolution actually reaches the limit set at some point in time. In general, limit sets can be very complicated, and it has been shown that all their nontrivial properties are undecidable, given the defining CA [4].

In this paper, we generalize the notion of a limit set and define limit sets of finite families of cellular automata, as opposed to a single automaton. By studying these more general objects, we hope to shed light on some of the problems concerning the conventional limit sets. The idea of a limit set of multiple CA stands somewhere between tiling systems and conventional limit sets, although our model does not directly give a subclass of tilings (we explore this connection in Section 3). Also, comparing the limit sets of two automata to their common limit set can give information about their relationship. For instance, the common limit set of a family of commuting automata is just the union of their limit sets, which is not true in general.

The paper is organized as follows. Section 1 consists of this introduction. In Section 2 we define the notions used in the rest of the paper.

⋆ Research supported by the Academy of Finland Grant 131558.

In Section 3, we prove some basic lemmas about limit sets of families of cellular automata. Using the notion of projective subdynamics and results from [7], we obtain a necessary condition for a subshift to be the limit set of some family of CA.

In Section 4, we focus on the classes of limit sets of a given number of cellular automata, and the hierarchy formed by increasing this number. The main results of this chapter state that both the stable and unstable hierarchies are infinite, with proper inclusions at every step. We also prove that the stable hierarchy, when restricted to transitive subshifts, collapses to a single level.

In Section 5, we study the closure properties of classes of limit sets under set-theoretic operations, that is, unions and intersections. The class of stable limit sets turns out to be closed under union, but our counterexamples show that neither the stable nor unstable class is closed under (nonempty) intersection. We also prove that the hierarchy formed by considering finite unions of stable limit sets of a given number of CA is finite: all stable limit sets of finite families of cellular automata can be expressed as finite unions of limit sets of just two automata. An open question is whether the hierarchy has only one level.

Section 6 consists of our conclusions about this paper.

## 2 Definitions

Let $S$ be a finite set, the *alphabet*, which is given the discrete topology. We denote by $S^* = \bigcup_{n \in \mathbb{N}} S^n$ the set of *words* over $S$, and if $w \in S^n$, we denote $|w| = n$. The space $S^{\mathbb{Z}}$ with the induced product topology is called the *full shift over* $S$. The topology of $S^{\mathbb{Z}}$ is also given by the metric $d$ defined by

$$d(x, y) = 2^{-\min\{|i| \mid x_i \neq y_i\}}.$$

For a subset $X \subset S^{\mathbb{Z}}$ and $\epsilon > 0$, we define

$$B_\epsilon(X) = \{y \in S^{\mathbb{Z}} \mid \exists x \in X : d(x, y) < \epsilon\}.$$

We also consider the two-dimensional shift space $S^{\mathbb{Z}^2}$ with the product topology.

If $x \in S^{\mathbb{Z}}$, we denote the $i$th coordinate of $x$ with $x_i$, and abbreviate the expression $x_i x_{i+1} \cdots x_{i+n-1}$ by $x_{[i,i+n-1]}$. If $u, v, w, t \in S^*$, we denote by ${}^\infty uv.wt^\infty$ the element $x \in S^{\mathbb{Z}}$ defined by $x_{[-n|u|,-(n-1)|u|-1]-|v|} = u$, $x_{[-|v|,-1]} = v$, $x_{[0,|w|-1]} = w$ and $x_{|w|+[n|t|,(n+1)|t|-1]} = w$, and the element ${}^\infty vv.vv^\infty$ is denoted ${}^\infty v^\infty$. We may use the notation $x = {}^\infty uvw^\infty$ when the position of the origin is irrelevant or can be inferred from the context. If $a \in S$, an element of the form ${}^\infty awa^\infty$ is called *$a$-finite*, and an element of the form ${}^\infty a^\infty$ is called a *uniform configuration*. For a word $w \in S^n$, we say that $w$ *occurs in* $x$ and denote $w \sqsubset x$, if there exists $i$ such that $w = x_{[i,i+n-1]}$. On $S^{\mathbb{Z}}$ we define the *shift map* $\sigma_S$ (or simply $\sigma$, if $S$ is clear from the context) by $\sigma_S(x)_i = x_{i+1}$ for all $i$. Clearly $\sigma$ is bijective and continuous w.r.t. the topology of $S^{\mathbb{Z}}$.

A finite set of words $W \subset S^*$ is said to be *mutually unbordered* if whenever two words $v, w \in W$ occur in some $x \in S^{\mathbb{Z}}$ as $v = x_{[0,|v|-1]}$ and $w = x_{[m,m+|w|-1]}$

where $m > 0$, then we must have $m \geq |v|$. This means that the words cannot overlap in any way. Note that we may have $v = w$ in this definition, so even self-overlaps are forbidden. Given any configuration $x \in S^{\mathbb{Z}}$, the set $W$ now uniquely partitions $\mathbb{Z}$ into $|W| + 1$ disjoint sets $A_w$ for $w \in W$ and $A$, defined by $i \in A_w$ if and only if $x_{[m,m+|w|-1]} = w$ for some $m \in [i - |w| + 1, i]$, and $A$ being the complement of $\bigcup_{w \in W} A_w$.

A closed, shift-invariant subset of $S^{\mathbb{Z}}$ is called a *subshift.* Alternatively, given a set of *forbidden words* $F \subset S^*$, a subshift can be defined by those points of $S^{\mathbb{Z}}$ in which no word from $F$ occurs. If $F$ is finite, the resulting subshift is a *subshift of finite type*, abbreviated SFT. We define $\mathbb{Z}^2$ subshifts and $\mathbb{Z}^2$ SFT's analogously, as closed shift-invariant subsets of $S^{\mathbb{Z}^2}$, which are also defined by a set of forbidden two-dimensional patterns. The set of words of length $n$ occurring in a subshift $X$ is denoted by $\mathcal{B}_n(X)$, and we define $\mathcal{B}(X) = \bigcup_n \mathcal{B}_n(X)$. If $X$ has the property that for all $v, w \in \mathcal{B}(X)$ and $N \in \mathbb{N}$ there exists an $n \geq N$, a word $z \in S^n$ and a point $x \in X$ with $vzw \sqsubset x$, then we say that $X$ is *transitive.* If there exists an $N$ such that the previous holds for all $n \geq N$, we say that $X$ is *mixing.* A point $x \in X$ is *doubly transitive*, if for all $w \in \mathcal{B}(X)$ and for all $N \in \mathbb{N}$, there exist $m \leq -N$ and $n \geq N$ with $x_{[m,m+|w|-1]} = x_{[n,n+|w|-1]} = w$. Clearly every transitive subshift contains a doubly transitive point. The restriction of $\sigma_S$ to $X$ is denoted by $\sigma_X$. The *entropy* of a subshift $X$ is defined as $h(X) = \lim_{n \longrightarrow \infty} \frac{1}{n} \log |\mathcal{B}_n(X)|$.

Given an $n \times n$ integral matrix $M$ with $M_{ij} \geq 0$ for all $i$ and $j$, we can construct an SFT from it by taking $S = \{(i, j, m) \mid i, j \in [1, n], 0 \leq m < M_{ij}\}$ as the state set and $\{((i_1, j_1, m_1), (i_2, j_2, m_2)) \mid j_1 \neq i_2\}$ as the set of forbidden words. This SFT is called the *edge shift defined by* $M$. If there is an $N$ such that $(M^n)_{ij} > 0$ for all $i$ and $j$ whenever $n \geq N$, we say that $M$ is *primitive.* It is known that $M$ is primitive if and only if its edge shift is mixing.

Let $X$ and $Y$ be two subshifts. A *block map from* $X$ *to* $Y$ is a continuous map $\psi : X \to Y$ such that $\psi \circ \sigma_X = \sigma_Y \circ \psi$. It is known that all block maps are defined by *local rules* $\Psi : \mathcal{B}_{2r+1}(X) \to \mathcal{B}_1(Y)$ so that $\psi(x)_i = \Psi(x_{[i-2r,i+2r]})$ for all $x \in X$ and $i \in \mathbb{N}$. The number $r$ is called the *radius* of $\psi$. If $r = 1$ and $\Psi$ does not depend on the rightmost coordinate, we say that $\psi$ has radius $\frac{1}{2}$ and give $\Psi$ as a function from $\mathcal{B}_2(X)$ to $\mathcal{B}_1(Y)$. The block map $\psi$ is said to be *preinjective* if $\psi(x) \neq \psi(y)$ whenever $x_i = y_i$ for all but a finite number of $i$. If $\psi$ is surjective, it is called a *factor map*, and then $Y$ is a *factor* of $X$. A factor of an SFT is called *sofic.* A *cellular automaton* is a block map from $S^{\mathbb{Z}}$ to itself.

## 3 Limit Sets and Basic Results

Let $\mathcal{F}$ be a finite family of cellular automata. We define the sets $L_i(\mathcal{F})$ for $i \in \mathbb{N}$ by $L_0(\mathcal{F}) = S^{\mathbb{Z}}$, and $L_i(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} f(L_{i-1}(\mathcal{F}))$ for $i > 0$. The *limit set of* $\mathcal{F}$ is the set $L(\mathcal{F}) = \bigcap_{i \in \mathbb{N}} L_i(\mathcal{F})$. We say that $\mathcal{F}$ is *stable* if $L(\mathcal{F})$ is equal to one of the $L_i(\mathcal{F})$.

We denote by $k-\mathrm{LIM}_s$ and $k-\mathrm{LIM}_u$ the classes of stable and unstable limit sets of families of at most $k$ cellular automata, respectively. The notation $k-\mathrm{LIM}_x$

refers to both classes (not their union), thinking of $x$ as a variable ranging over $\{s, u\}$. We also denote $\infty{-}\mathrm{LIM}_x = \bigcup_k k{-}\mathrm{LIM}_x$ for $x \in \{s, u\}$.

To illustrate the concept of limit sets of finite families of CA, we give an example of a complex limit set of two automata.

*Example 1.* Consider the two automata $f_0$ and $f_1$ on the alphabet $\{0, 1, \#\}$ where each $f_i$ has radius $\frac{1}{2}$, and the local rule of $f_i$ is given by the following table:

|  | 0 | 1 | # |
|---|---|---|---|
| 0 | 0 | 0 | # |
| 1 | 1 | 1 | # |
| # | $i$ | $i$ | # |

Now the limit set $L(\{f_0, f_1\})$ is the subshift defined by the forbidden words $\{\#uv\#w \mid n \in \mathbb{N}, u, w \in \{0,1\}^n, v \in \{0,1,\#\}^*, u \neq w\}$. That is, every two #-symbols in a configuration of $L(\{f_0, f_1\})$ must be followed by the same words over $\{0, 1\}$.

The following two lemmas are direct generalizations of well-known properties of limit sets of a single cellular automaton.

**Lemma 2.** *Let $\mathcal{F}$ be a finite family of cellular automata. Then we have $L(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} f(L(\mathcal{F}))$.*

*Proof.* It is clear that $f(L(\mathcal{F})) \subset L(\mathcal{F})$ for all $f \in \mathcal{F}$, since

$$f(L(\mathcal{F})) = f(\bigcap_i L_i(\mathcal{F})) \subset \bigcap_i f(L_i(\mathcal{F})) \subset \bigcap_i L_{i+1}(\mathcal{F}) = L(\mathcal{F}).$$

For the other inclusion, consider an arbitrary point $x \in L(\mathcal{F})$, and let $Z = \bigcup_{f \in \mathcal{F}} f^{-1}(x)$. Since $x \in L_{i+1}(\mathcal{F})$, we have $Z \cap L_i(\mathcal{F}) \neq \emptyset$ for all $i$. Since we now have a descending chain of nonempty compact sets, the intersection

$$\bigcap_i (Z \cap L_i(\mathcal{F})) = Z \cap L(\mathcal{F})$$

is nonempty. Therefore, $x$ has a preimage in $L(\mathcal{F})$. □

**Lemma 3.** *Let $\mathcal{F}$ be a finite family of cellular automata. For all $\epsilon > 0$, there exists $k$ such that $L_k(\mathcal{F}) \subset B_\epsilon(L(\mathcal{F}))$.*

*Proof.* The set $Z = S^{\mathbb{Z}} - B_\epsilon(L(\mathcal{F}))$ is compact. Therefore, if $Z \cap L_i(\mathcal{F})$ were nonempty for all $i$, we would also have $Z \cap L(\mathcal{F}) \neq \emptyset$ as in the previous proof. □

By the definition of the metric in $S^{\mathbb{Z}}$, this means that for all $m$, there exists a $k_m$ such that $\mathcal{B}_m(L_k(\mathcal{F})) = \mathcal{B}_m(L(\mathcal{F}))$ for all $k \geq k_m$. In particular, a limit set that is an SFT must be stable.

We will use the following result from [7] to prove certain languages not to be unstable limit sets of any families of cellular automata. First, we define the notion of projective subdynamics.

**Definition 4.** *Let $X$ be a $\mathbb{Z}^2$ SFT. The set of horizontal rows appearing in points of $X$ is called the* $\mathbb{Z}$-projective subdynamics *of $X$.*

**Definition 5.** *If $X$ is a one-dimensional sofic shift, we say $l$ is a* universal period *for $X$ if there exists $M$ such that for all $x \in X$ there exists $y$ with $y = \sigma^l(y)$ such that $|\{i \mid x_i \neq y_i\}| \leq M$.*

**Lemma 6.** ([7]) *A zero-entropy proper one-dimensional sofic shift $X$ is realizable as the $\mathbb{Z}$-projective subdynamics of a $\mathbb{Z}^2$ SFT if and only if it has no universal period.*

**Corollary 7.** *A zero-entropy proper sofic shift with a universal period is not the limit set of any finite family of CA.*

*Proof.* Any family $\mathcal{F}$ of cellular automata defined on $S^{\mathbb{Z}}$ defines a $\mathbb{Z}^2$ SFT $X$ over the alphabet $S \times \mathcal{F}$ as follows: The $\mathcal{F}$-component must be constant in every horizontal row. If $f \in \mathcal{F}$ is the CA of row $i+1$, then the $S$-component of row $i$ must be the $f$-image of the $S$-component of row $i+1$. Now the $\mathbb{Z}$-projective subdynamics of $X$ is precisely $L(\mathcal{F}) \times \{{}^\infty f^\infty \mid f \in \mathcal{F}\}$, so it cannot be a zero-entropy proper sofic shift with a universal period. But then $L(\mathcal{F})$ cannot have that property either. □

We present here a realization theorem for limit sets, which uses techniques already found in [7].

**Theorem 8.** *If $X$ is the limit set of a family of cellular automata containing at least two periodic points, then $X$ is realizable as the $\mathbb{Z}$-projective subdynamics of a $\mathbb{Z}^2$ SFT.*

*Proof.* Let $S$ be the alphabet of $X = L(\{f_1, \dots, f_k\})$, and let $q_1, q_2 \in X$ be distinct periodic points such that $q_1 = {}^\infty a^\infty$ and $q_2$ has period $p$. Using the identification $S^{\mathbb{Z}^2} \cong (S^{\mathbb{Z}})^{\mathbb{Z}}$, we think of configurations of $S^{\mathbb{Z}^2}$ as bi-infinite vertical words over the alphabet $S^{\mathbb{Z}}$ of bi-infinite horizontal words. Then, choose $k$ mutually unbordered vertical words $w_i \in \{q_1, q_2\}^m$.

We now construct the $\mathbb{Z}^2$ SFT $Y$ having $X$ as its $\mathbb{Z}$-projective subdynamics. The local rule of $Y$ works on a coordinate $(a, b)$ of a configuration $x$ as follows: If $x_{[a,a+p-1]\times[b+r,b+r+m-1]}$ does not appear in any $w_i$ for any $r \in [1-m, 0]$, then both $x_{[a,a+p-1]\times[b+1,b+m]}$ and $x_{[a,a+p-1]\times[b-m,b-1]}$ must do so. Then, since the $w_i$ are mutually unbordered, each row of $Y$ must be either part of some vertical word $w_i$, or between two such words. Additionally, if $\substack{x \\ w_i \\ y}$ appears as a vertical subword in $Y$, where $x$ and $y$ are not part of any $w_j$, then we require that $y = f_i(x)$. It is now clear that $X$ is the $\mathbb{Z}$-projective subdynamics of $Y$. □

**Corollary 9.** *All stable limit sets $X$ of finite families of cellular automata are realizable as the $\mathbb{Z}$-projective subdynamics of a $\mathbb{Z}^2$ SFT.*

*Proof.* Since $X$ is stable, it must be sofic. If $X$ is an SFT, the claim is trivial. Now, if $|X| > 1$ and $X$ is proper sofic but does not contain two distinct periodic points, then it contains a unique periodic point, which must then be unary. But then it is clear that $X$ has zero entropy and a universal period, contradicting Corollary 7. □

## 4 Hierarchies

We now turn to the relations between the classes $k-\mathrm{LIM}_x$. The following theorem is obvious since all classes $k-\mathrm{LIM}_s$ contain SFT's, and all classes $k'-\mathrm{LIM}_u$ contain subshifts that are not sofic.

**Theorem 10.** *The classes $k-\mathrm{LIM}_s$ and $n-\mathrm{LIM}_u$ are incomparable for $k, n \geq 1$.*

It is also known that the classes are not completely disjoint [1].

*Example 11.* Let $S = \{0, 1, \#\}$, and let $g_0$ and $g_1$ be the CA of radius 0 on $S$ defined as follows:

$$g_0(x)_i = \begin{cases} \#, \text{ if } x_i = \# \\ 0, \text{otherwise} \end{cases} \text{ and } g_1(x)_i = \begin{cases} \#, \text{ if } x_i = \# \\ 1, \text{otherwise} \end{cases}$$

Now the limit set of the family $\{g_0, g_1\}$ is $X = \{0, \#\}^{\mathbb{Z}} \cup \{1, \#\}^{\mathbb{Z}}$, and is reached in one step. Since this subshift is not transitive, it cannot be the limit set of a single stable CA.

It can even be proven that $X$ is not the limit set of any unstable CA, using argumentation similar to that used in the proof of Theorem 23.

It is slightly more complicated to prove that both hierarchies are infinite. This will be our goal for the remainder of this section. We begin by finding arbitrarily large families of mixing SFT's $X_i$ that have at least one uniform configuration with the property that $X_i$ does not factor onto $X_j$ for any $i \neq j$. We need some lemmas from [5].

**Definition 12.** *If $A$ is a primitive integral matrix, let $\lambda_A$ be its greatest eigenvalue with respect to absolute value, and $\mathrm{sp}^\times(A)$ the unordered list (or multiset) of its eigenvalues, called the* nonzero spectrum *of $A$. We use the notation $\langle \lambda_1, \ldots, \lambda_k \rangle$ for the unordered list containing the elements $\lambda_i$.*

**Lemma 13.** ([5]) *The entropy of the edge shift $X$ defined by a primitive integral matrix $A$ is $\log \lambda_A$.*

**Lemma 14.** ([5]) *If the edge shifts $X$ and $Y$ defined by two primitive integral matrices $A$ and $B$, respectively, have the same entropy and $X$ factors onto $Y$, then $\mathrm{sp}^\times(B) \subset \mathrm{sp}^\times(A)$.*

The previous lemma gives us a necessary condition for factoring between edge shifts of equal entropy, so it is enough to find, for each $k$, a family of $k$ matrices $M_i$ with the property that $\lambda_{M_i} = \lambda_{M_j}$ for all $i$ and $j$, but $\mathrm{sp}^\times(M_i) \not\subset \mathrm{sp}^\times(M_j)$ whenever $i \neq j$. The following lemmas are the tools we need for this.

**Definition 15.** *Let $\Lambda = \langle \lambda_1, \ldots, \lambda_k \rangle$ be an unordered list of complex numbers. Denoting $\Lambda^d = \langle \lambda_1^d, \ldots, \lambda_k^d \rangle$, let*

$$\mathrm{tr}(\Lambda) = \sum_{i=1}^{k} \lambda_i$$

*be the* trace *of $\Lambda$, and*

$$\mathrm{tr}_n(\Lambda) = \sum_{d|n} \mu(\frac{n}{d})\mathrm{tr}(\Lambda^d)$$

*the* $n$th trace *of $\Lambda$, for all $n \in \mathbb{N}$, where $\mu : \mathbb{N} \to \{-1, 0, 1\}$ is the Möbius function.*

**Lemma 16.** ([5]) *Let $A$ be a primitive integral matrix and $B$ an integral matrix such that*

- *$\lambda_B < \lambda_A$, and*
- *$\mathrm{tr}_n(\mathrm{sp}^\times(A)) + \mathrm{tr}_n(\mathrm{sp}^\times(B)) \geq 0$ for all $n \geq 1$.*

*Then there is a primitive integral matrix $C$ such that* $\mathrm{sp}^\times(C) = \mathrm{sp}^\times(A) \cup \mathrm{sp}^\times(B)$.

Now let $A$ be a matrix $[\lambda]$ with a single entry much greater than $k$, and let $B_i$ be the matrices $B_i = [i]$ for $i \in [1, k]$. By taking a large enough $\lambda$, the assumptions of Lemma 16 are satisfied for the pairs $A$ and $B_i$. Now, the matrices $C_i$, where $C_i$ is given by the lemma for $A$ and $B_i$, are primitive integral matrices with the same greatest eigenvalue, but incomparable nonzero spectra. Thus, their edge shifts have the same entropy by Lemma 13, but none of them factor onto another one by Lemma 14.

We still have one problem left: the edge shifts defined by the $C_i$ might not have uniform configurations. This is fixed by taking a common power of the $C_i$: Since the matrices are primitive, there exists a $p$ such that all of $C_i^p$ have nonzero matrix trace, and thus define edge shifts with uniform configurations. The $C_i^p$ are again primitive, and since $\mathrm{sp}^\times(C_i^p) = \mathrm{sp}^\times(C_i)^p$ and the eigenvalues are positive, we still have no factoring relations. Renaming the symbols of the shifts, we have proved the following lemma:

**Lemma 17.** *For all $k \in \mathbb{N}$, there exists a finite alphabet $S_k$, a symbol $a \in S_k$ and a set $\{X_1, \ldots, X_k\}$ of $k$ mixing edge shifts over $S_k$ such that whenever $i \neq j$, we have that $X_i$ does not factor onto $X_j$, $X_i \cap X_j = {}^\infty a^\infty$ and $\mathcal{B}_1(X_i) \cap \mathcal{B}_1(X_j) = a$.*

We still need some further lemmas:

**Lemma 18.** ([6]) *Let $X \subset S^{\mathbb{Z}}$ be an SFT. Then $X$ is the stable limit set of some cellular automaton if and only if $X$ is mixing and contains a uniform configuration.*

**Lemma 19.** ([5, Corollary 4.4.9]) *If $X$ is an SFT and $Y$ a transitive sofic shift with $X \subset Y$ and $h(X) = h(Y)$, then $X = Y$.*

**Lemma 20.** ([5, Proposition 4.1.9]) *If $X$ is a subshift and $\psi$ any block map, then $h(\psi(X)) \leq h(X)$.*

We are now ready to prove the main theorems of this chapter, namely the infinity of the stable and unstable hierarchies. More precisely, we prove that each level of the hierarchies is separated from the previous one.

**Theorem 21.** *For all $k$, $(k-1)-\mathrm{LIM}_s \subsetneq k-\mathrm{LIM}_s$.*

*Proof.* Let $X_i$ be given by Lemma 17, and for each $i$ let $f_i$ be the cellular automaton given by Lemma 18 having $X_i$ as its stable limit set, operating on $\mathcal{B}_1(X_i)$ instead of $S_k$. We extend each $f_i$ to the common alphabet $S_k$ by having $f_i$ consider every symbol of $S_k - \mathcal{B}_1(X_i)$ as $a$. Then clearly the system $\{f_1, \dots, f_k\}$ has $X = \bigcup_i X_i$ as its stable limit set, and thus $X \in k-\mathrm{LIM}_s$.

Now consider a hypothetical system $\mathcal{F}$ with $|\mathcal{F}| < k$ such that $X = L(\mathcal{F}) = L_n(\mathcal{F})$. Let $i$ be arbitrary, and consider a doubly transitive point $x \in X_i$. Since $x$ must have a preimage in $X$, we see that for some $f \in \mathcal{F}$ we have $X_i \subset f(X_j)$. By Lemma 20, this means that $h(X_i) \leq h(f(X_j)) \leq h(X_j) = h(X_i)$, so by Lemma 19 we have $X_i = f(X_j)$. But by the property of the $X_i$, this is only possible if $i = j$, so that $f$ maps $X_i$ onto itself.

Since $|\mathcal{F}| < k$, there must exist $f \in \mathcal{F}$ and indices $i$ and $j$ such that $f$ maps both $X_i$ and $X_j$ onto themselves. In this case, we must have $f({}^\infty a^\infty) = {}^\infty a^\infty$. Let $r$ be the radius of $f$. Let $b' \in \mathcal{B}(X_i) - a^*$ such that ${}^\infty a b' a^\infty \in X_i$. Now there exists $x' \in X_i$ with $f^n(x') = {}^\infty a b' a^\infty$. Let $x = {}^\infty a x'_{[-rn,|b|+rn]} a^\infty$. Then $f^n(x) = {}^\infty a b a^\infty$, where $b \in \mathcal{B}(X_i)$ with $b_{[m,m+|b'|-1]} = b'$ for some $m$. Similarly, we obtain $c \in \mathcal{B}(X_j) - a^*$ such that ${}^\infty a c a^\infty$ has an $a$-finite $f^n$-preimage in $X_j$. For all $N \in \mathbb{N}$, define $y_N = {}^\infty a b a^N c a^\infty \in S_k^{\mathbb{Z}}$. Now for large enough $N$, $f^{-n}(y_N) \neq \emptyset$, which is a contradiction, since $y_N \notin X$. □

For the unstable case, yet another lemma is needed:

**Lemma 22.** ([5, Theorem 8.2.19]) *If $X \subset S^{\mathbb{Z}}$ is a mixing SFT and $\phi : X \to X$ is a factor map, then $\phi$ is preinjective.*

**Theorem 23.** *For all $k$, $(k-1)-\mathrm{LIM}_u \subsetneq k-\mathrm{LIM}_u$.*

*Proof.* Let $X_i$ be given by Lemma 17, and for each $i$ let $f_i$ be the cellular automaton given by Lemma 18 having $X_i$ as its stable limit set, operating on $\mathcal{B}_1(X_i)$ instead of $S_k$. We extend each $f_i$ to the alphabet $S_k \cup \{\#\}$ by having $f_i$ consider every symbol of $S_k - \mathcal{B}_1(X_i)$ as $a$, and making $\#$ a spreading state. Denote $X' = \bigcup_i X_i$. It is clear that the family $\{f_1, \dots, f_k\}$ has $X = \overline{\{{}^\infty \# b \#^\infty \mid b \sqsubset X'\}}$ as its unstable limit set, implying $X \in k-\mathrm{LIM}_u$.

Now consider again a hypothetical system $\mathcal{F}$ with $|\mathcal{F}| < k$ having $X$ as its unstable limit set. It is easy to see that a doubly transitive point $x \in X_i$ must have a preimage in $X$ without the symbol $\#$. In particular, we again see that some $f \in \mathcal{F}$ must map $X_i$ onto itself, and find $f$, $i$ and $j$ such that $f$ maps both $X_i$ and $X_j$ onto themselves. It is clear that $f({}^\infty a^\infty) = {}^\infty a^\infty$.

Now it is not a contradiction that configurations with symbols from both $\mathcal{B}_1(X_i) - \{a\}$ and $\mathcal{B}_1(X_j) - \{a\}$ have long chains of preimages, so we need a slightly more involved argument than in Theorem 21. Let $r$ be the radius of $f$. By Lemma 3, each point in $S_k^{\mathbb{Z}}$ must locally approach $X$ when $f$ is applied to it repeatedly, so there exists an $M$ such that $\mathcal{B}_{2r+1}(f^m(S_k^{\mathbb{Z}})) \subset \mathcal{B}_{2r+1}(X)$ for all $m \geq M$.

Let $b, c \notin a^*$ be such that ${}^\infty a b a^\infty \in X_i$ and ${}^\infty a c a^\infty \in X_j$, and define

$$y = {}^\infty(b a^{2r(M+1)} c a^{2r(M+1)})^\infty.$$

Since $y$ is periodic with some period $p$, so are all its images $f^n(y)$. We claim that in all points $f^n(y)$ for $n \in \mathbb{N}$, symbols from both $\mathcal{B}_1(X_i) - \{a\}$ and $B_1(X_j) - \{a\}$ appear: First, no # can appear in the images, since all $(2r+1)$-blocks that occur will be from $\mathcal{B}_{2r+1}(X')$. Since $f$ is a surjection from $X_i$ to itself and $X_i$ is mixing, $f$ is preinjective on $X_i$ by Lemma 22, and similarly for $X_j$.

Now, when we apply $f$ to a block of the form $a^{2r}wa^{2r}$ where $w \sqsubset X_i$ is not completely over $a$, the result must contain at least one symbol of $\mathcal{B}_1(X_i) - \{a\}$, and similarly for $X_j$. Since non-$a$ symbols from both subshifts cannot appear in the same $(2r+1)$-block after $M$ steps, we then inductively see that symbols from both shifts must appear in all of the points $f^n(y)$. Since the $f^n(y)$ are periodic with period $p$, they cannot locally approach $X$, and this contradiction proves the claim. □

Note that the more involved argument of Theorem 23 can also be used in the stable case, implying that there exist subshifts

$$X \in k\text{–LIM}_s - ((k-1)\text{–LIM}_s \cup (k-1)\text{–LIM}_u),$$

and

$$Y \in k\text{–LIM}_u - ((k-1)\text{–LIM}_s \cup (k-1)\text{–LIM}_u)$$

for all $k > 1$.

The following theorem shows that the existence of several mixing components is important in the stable case:

**Theorem 24.** *If $X \in \infty\text{–LIM}_s$ is transitive, then $X \in 1\text{–LIM}_s$.*

*Proof.* Let $X$ be the stable limit set of the family $\mathcal{F}$ reached in one step. Then, since $X$ contains a doubly transitive point, we find $f \in \mathcal{F}$ that maps $X$ onto itself as in the previous proofs. But the limit set of $f$ must be contained in $X$, so $X = L(\{f\})$, and $f$ reaches $X$ in one step. □

In the unstable case, Theorem 24 might not hold as such, since the automaton $f$ could be stable. The following, however, is true.

**Theorem 25.** *If $X \in \infty\text{–LIM}_u$ is transitive, then $X \in 1\text{–LIM}_s \cup 1\text{–LIM}_u$.*

*Question 26.* Is the unstable hierarchy proper when restricted to transitive subshifts?

## 5 Boolean Operations

In this section, we study the relation between the classes $\infty\text{–LIM}_x$ and set-theoretic operations. We begin with an easy lemma.

**Lemma 27.** *If $\mathcal{F}$ and $\mathcal{F}'$ are stable families of cellular automata, then $L(\mathcal{F}) \cup L(\mathcal{F}')$ is the limit set of a stable family $\mathcal{F}''$ of automata. If both limit sets are reached in one step, then $\mathcal{F}''$ can be taken to be $\mathcal{F} \cup \mathcal{F}'$.*

*Proof.* Let $k$ be such that $L_k(\mathcal{F}) = L(\mathcal{F})$ and $L_k(\mathcal{F}') = L(\mathcal{F}')$. Then $L_1(\mathcal{F}'') = L(\mathcal{F}'') = L(\mathcal{F}) \cup L(\mathcal{F}')$, where $\mathcal{F}'' = \mathcal{F}^k \cup \mathcal{F}'^k$. □

**Corollary 28.** *The class $\infty-\mathrm{LIM}_s$ is closed under finite union.*

**Theorem 29.** *Let $\mathcal{F}$ be a finite family of CA. Then $\bigcup_{f\in\mathcal{F}} L(\{f\}) \subset L(\mathcal{F})$. If the automata in $\mathcal{F}$ commute, equality holds.*

*Proof.* The first claim is clear from the definition. Suppose then that the automata commute and $x \in L(\mathcal{F})$, so that there is a sequence $(f_i)_{i\in\mathbb{N}}$ over $\mathcal{F}$ such that for all $n$, $f_n^{-1}(\cdots f_1^{-1}(x)\cdots) \neq \emptyset$. One of the automata, say $f$, must occur infinitely many times in the sequence. Let $i \in \mathbb{N}$. Since the automata commute, we can move $i$ copies of $f$ to the beginning of the sequence, so that $f^{-i}(x) \neq \emptyset$. □

Since limit sets are always nonempty, the question whether intersections of limit sets are always limit sets themselves is trivially false. In the case of nonempty intersections, we have the following counterexamples:

*Example 30.* The class $\infty-\mathrm{LIM}_u$ is not closed under nonempty intersection.

*Proof.* Take two subshifts over the alphabet $\{0, 1, 2\}$: $X$ is a one-step SFT with the forbidden words $\{10, 11, 02, 22\}$, and $Y$ is the sofic shift with forbidden words $\{20^n1 \mid n \in \mathbb{N}\}$. By Lemma 18 $X$ is the stable limit set of some CA, and clearly $Y$ is the unstable limit set of the CA that moves 1's to the right and 2's to the left, destroying them when they collide.

Consider then the shifts $Z = X \times Y$ and $Z' = Y \times X$. Now both $Z$ and $Z'$ are unstable limit sets of the product automata. However, their intersection, which is the product with itself of the orbit closure of ${}^\infty 0120^\infty$, can't be the limit set of any family of CA by Corollary 7. □

The following example can also be found in [3].

*Example 31.* The class $\infty-\mathrm{LIM}_s$ is not closed under nonempty intersection.

*Proof.* Let $f$ be the radius $\frac{1}{2}$ CA over the alphabet $S = \{0, 1, 2\}$ whose local rule is given by the following table:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 1 | 2 | 0 | 0 |
| 2 | 2 | 0 | 0 |

The automaton $f$ marks every transition $0x$ with 1 and $x0$ with 2, where $x \in \{1, 2\}$, and otherwise produces 0. Denote by $X$ the subshift with forbidden words $\{a0^na \mid n \in \mathbb{N}, a \in \{1, 2\}\}$. It is clear that $L_1(\{f\}) \subset X$. On the other hand, if $x \in X$, we can easily construct a preimage $y \in X$ for it: Note that when we consider 1 and 2 equal, $f$ behaves like the binary XOR automaton with radius $\frac{1}{2}$. Since the XOR automaton is surjective, we obtain an $f$-preimage for $y$ by taking a suitable XOR-preimage for it, and replacing every other 1 by 2. This shows that $X \subset f(X)$, from which it follows that $L(\{f\}) = X$.

Let $g$ be the symbol-transforming automaton that maps $0 \mapsto 0$, $1 \mapsto 1$ and $2 \mapsto 0$ having the stable limit set $\{0,1\}^{\mathbb{Z}}$. Now, the intersection of the limit sets of $f$ and $g$ is the orbit closure of ${}^{\infty}010^{\infty}$, which, again by Corollary 7, is not the limit set of any family of automata. □

In fact, the previous proofs show that $1{-}\mathrm{LIM}_x$ is not closed under intersection for $x \in \{s, u\}$.

*Question 32.* Is $\infty{-}\mathrm{LIM}_u$ closed under union?

Another interesting question is whether elements of $k{-}\mathrm{LIM}_x$ can be decomposed into finite unions of elements of $m{-}\mathrm{LIM}_x$, for $m < k$. In the stable case, the following theorem proves this in the positive for $m = 2$, but the case $m = 1$ is still unknown. In the unstable case, nothing is known.

**Theorem 33.** *If $X \in \infty{-}\mathrm{LIM}_s$, then $X$ is the union of a finite number of subshifts in $2{-}\mathrm{LIM}_s$.*

*Proof.* We may assume $X$ is the limit set of a family $\mathcal{F} = \{f_1, \ldots, f_k\}$ with $X = L_1(\mathcal{F})$. Let $X_i = f_i(S^{\mathbb{Z}})$, noting that $X = \bigcup_i X_i$. Consider one of the (mixing) components $X_i$. Without loss of generality, we may assume that $X_i \subsetneq X_j$ for no $j \neq i$: Suppose that such $j$ exists. We claim that we can then remove $f_i$ from $\mathcal{F}$ without changing the limit set. Clearly, the limit set can't grow, so it suffices to show that $X_i \subset f(X)$ for some $f \in \mathcal{F} - \{f_i\}$. But this is clear, since some point of $X_j$ is doubly transitive and must have a preimage in $X$ with some $f \in \mathcal{F}$, and necessarily $f \neq f_i$.

Let $x \in X_i$ be doubly transitive, and let $f \in \mathcal{F}$ be such that $f(y) = x$, where $y \in X_{j_1}$ for some $j_1$. Now we easily see that $X_i \subset f(X_{j_1})$, so that actually $f = f_i$ and $X_i = f(X_{j_1})$. We repeat this argument for $X_{j_1}$ to obtain $j_2$ such that $f_{j_1}(X_{j_2}) = X_{j_1}$, and continue inductively to obtain a sequence $(j_n) \in [1,k]^{\mathbb{N}}$ such that $f_{j_n}(X_{j_{n+1}}) = X_{j_n}$ for all $n$. Since $[1,k]$ is finite, we have that $j_m = j_{m+p}$ for some $m$ and $p > 0$.

Now denote $Y = X_i$ and $Z = X_{j_m}$, and consider the CA

$$g = f_i \circ f_{j_1} \circ \cdots \circ f_{j_{m-1}}$$

and

$$h = f_{j_m} \circ \cdots \circ f_{j_{m+p-1}}.$$

We clearly have $L(\{g,h\}) \subset X$, since $g$ and $h$ are compositions of automata in $\mathcal{F}$. Since $h(S^{\mathbb{Z}}) = h(Z) = Z$ and $g(S^{\mathbb{Z}}) = g(Z) = Y$, we have that $Y \subset L(\{g,h\}) = Y \cup Z$. Also, the limit set is reached in one step. Now we have proved that $X_i \subset L(\{g,h\}) \subset X$ and $L(\{g,h\}) \in 2{-}\mathrm{LIM}_s$, which completes the proof since $i$ was arbitrary. □

*Question 34.* Is there a subshift $X \in 2{-}\mathrm{LIM}_s$ which is not a union of a finite number of subshifts in $1{-}\mathrm{LIM}_s$?

## 6 Conclusions

In this paper, we have studied basic properties of limit sets of finite families of cellular automata. The first question we asked was which subshifts can be realized as the limit set of a family of CA, and what is their relation to tiling systems. Here, we have shown that all stable limit sets can be realized as projective subdynamics of $\mathbb{Z}^2$ SFT's, but the unstable case is still open.

Another obvious question is what happens if one increases the number of automata. We separated the stable and unstable hierarchies and showed the triviality of the stable transitive hierarchy, but for now, not much can be said about the unstable transitive hierarchy.

Finally, we studied the relation between classes of limit sets and Boolean operations, in particular whether a limit set can be expressed as a union of simpler ones. The stable class is closed under union, but neither class is closed under intersection. We also proved that limit sets of commuting families can be decomposed into those of the individual CA, and stable limit sets into limit sets of pairs of CA. An open question is whether these results can be improved.

**Acknowledgements.** Finally, we'd like to thank Pierre Guillon for Lemma 27 and for pointing us in the direction of [7].

## References

1. Ballier, A., Guillon, P., Kari, J.: Limit sets of stable and unstable cellular automata. To Appear in Fundamenta Informaticae
2. Hedlund, G.A.: Endomorphisms and automorphisms of the shift dynamical system. Math. Systems Theory 3, 320–375 (1969)
3. Kari, J.: Properties of limit sets of cellular automata. In: Cellular Automata and Cooperative Systems (Les Houches, 1992). NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci, vol. 396, pp. 311–321. Kluwer Acad. Publ., Dordrecht (1993)
4. Kari, J.: Rice's theorem for the limit sets of cellular automata. Theoret. Comput. Sci. 127(2), 229–254 (1994), http://dx.doi.org/10.1016/0304-3975(94)90041-8
5. Lind, D., Marcus, B.: An introduction to symbolic dynamics and coding. Cambridge University Press, Cambridge (1995)
6. Maass, A.: On the sofic limit sets of cellular automata. Ergodic Theory and Dynamical Systems 15 (1995)
7. Pavlov, R., Schraudner, M.: Classification of sofic projective subdynamics of multidimensional shifts of finite type (submitted)

# Automaton Ranks of Some Self-similar Groups

Adam Woryna

Institute of Mathematics, Silesian University of Technology
Kaszubska 23, 44-100 Gliwice, Poland
adam.woryna@polsl.pl

**Abstract.** Given a group $G$ and a positive integer $d \geq 2$ we introduce the notion of an automaton rank of a group $G$ with respect to its self-similar actions on a $d$-ary tree of words as the minimal number of states in an automaton over a $d$-letter alphabet which generates this group (topologically if $G$ is closed). We construct minimal automata generating free abelian groups of finite ranks, which completely determines automaton ranks of free abelian groups. We also provide naturally defined 3-state automaton realizations for profinite groups which are infinite wreath powers $\ldots \wr H \wr H$ for some 2-generated finite perfect groups $H$. This determines the topological rank and improves the estimation for the automaton rank of these wreath powers. We show that we may take $H$ as alternating groups and projective special linear groups.

**Keywords:** Tree of Words, Self-similar Group, Automaton Group, Wreath Product.

## 1 Introduction

There are two approaches to represent groups by automata. The first one is via acceptors, also known as Rabin-Scott automata, which brings automaticity via Thurston or automaticity via Khoussainov-Nerode (see for example [5, 8, 9]). The second one is based on the concept of transducers, also known as Mealy-type automata or sequential machines, which transmit, letter by letter, an input sequence of letters from a certain alphabet into an output sequence. The language of transducers turned out to be fruitful finding examples of the so-called *self-similar groups* with extraordinary properties and investigating their geometric actions (see [6, 7, 10, 11]).

Let $X$ be a certain alphabet. In the present paper we refer to Mealy-type automata over $X$ as particular subsets with elements (called *states*) in the automorphism group of the tree $X^*$ of finite words over $X$. For a given automaton $S$ we call a subgroup generated by the set $S$ as a *group defined by the automaton* $S$. An abstract group $G$ is called an *automaton group* if $G$ is isomorphic with a group defined by an automaton over a certain alphabet $X$. In the theory of automaton groups the following question arises naturally: given integers $n, d \geq 2$ which groups are defined by $n$-state automata over a $d$-letter alphabet? Until now, the full classification was given only in case $n = d = 2$ (see [6, 7]) and some partial solution ([4]) was obtained for $n = 3$, $d = 2$. On the other hand, different automaton representations were found for many classical groups, such as free groups, some free products of finite cyclic groups, affine groups, lamplighter groups, some Baumslag-Solitar groups and others (see [1, 6, 7, 13–15]). In this context, it would be interesting for a given group $G$ and an integer $d \geq 2$ to study various automaton representations of $G$ by automata over a $d$-letter alphabet and to find among them an optimal representation, that is an automaton with the minimal number of states. This minimal number we denote by $ar(G, d)$ and call an *automaton rank* of $G$ (with respect to self-similar actions of $G$ on a $d$-ary tree of words). Referring to the ordinary rank $r(G)$, which is the minimal number of symbols needed to write any element of $G$ as a group-word on these symbols and, in a sense, measures the combinatorial complexity of this group, one can think of the automaton rank of $G$ as measuring the complexity of this group as an automaton group. Obviously, for any group $G$ and any $d \geq 2$ we have $ar(G, d) \geq r(G)$. Since finding any automaton representation of even algebraically not complicated group is rather unexpected and far from obvious, all the more difficult is to determine its automaton ranks, as it usually involves proving that the corresponding automaton is minimal. In the present paper we prove the following

**Theorem 1.1.** *The automaton rank of a free abelian group $\mathbb{Z}^k$ is equal to $k$ if and only if $k > 1$ and $d > 2$. In other cases we have $ar(\mathbb{Z}^k, d) = k + 1$.*

For the proof we construct for any $k \geq 2$ a $k$-state automaton over a 3-letter alphabet which generates a free abelian group of rank $k$. Further, by using results from [10, 11] concerning contracting self-similar actions on a binary rooted tree, we show that there is no $k$-state automaton over a 2-letter alphabet which generates a free abelian group of rank $k$. For the second result, we find automata defining some natural profinite subgroups in the automorphism group of a tree $X^*$, namely, the infinite wreath powers $\mathcal{W}(H) = \ldots \wr H \wr H$, where $H$ is some 2-generated perfect (i.e. equals its commutator subgroup) transitive permutation group on the set $X$. This construction implies the following

**Theorem 1.2.** *Let $H$ be a perfect transitive permutation group on a finite set $X$ generated by 2 elements, one of which has a fixed point and the other decomposes into disjoint cycles of the same length. Then the rank of the infinite wreath power $\mathcal{W}(H)$ is equal to 2 and for $d = |X|$ the automaton rank $ar(\mathcal{W}(H), d)$ is not greater than* 3.

Finally, we show that we may take $H$ as alternating groups $A_n$ of degree $n > 4$ as well as projective special linear groups $PSL_2(p)$ with a prime $p > 3$. In this context it is worth to note (see [2, 12]) that for any nonabelian finite simple group $H$ with its transitive action the rank of $\mathcal{W}(H)$ is equal to 2.

## 2 Self-similar Groups and Their Automaton Ranks

Let $X$ be a finite set of $d \geq 2$ symbols (the so-called alphabet). By $X^*$ we denote the tree of finite words (including the empty word $\epsilon$) over $X$. The set of all words $w \in X^*$ of a given length $i$ ($i \geq 0$) constitutes the $i$-th level of the tree $X^*$. In particular the 0-th level consists only of the empty word $\epsilon$, which is called the root of the tree $X^*$.

A faithful action $G \times X^* \to X^*$, $(g, w) \mapsto w^g$ of a given group $G$ on the tree $X^*$ is called self-similar if for every $g \in G$ and every $x \in X$ there are $y \in X$ and $h \in G$ such that $(xw)^g = yw^h$ for every $w \in X^*$. A group $G$ which has a self-similar action on a tree $X^*$ is called self-similar. The element $h \in G$ in the last equality is denoted by $g_x$ and is called the *section* of $g$ at the letter $x$. The permutation $\pi_g\colon x \mapsto y$ of the alphabet $X$ is called the *root-permutation* of $g$ or the *vertex-permutation* of $g$ at the root. We also define the section $g_w \in G$ and the vertex-permutation $\pi_{g,w} \in Sym(X)$ of every $g \in G$ at any word $w \in X^*$ inductively: $g_\epsilon = g$, $g_{xw} = (g_x)_w$, $\pi_{g,\epsilon} = \pi_g$, $\pi_{g,xw} = \pi_{g_x,w}$.

A natural example of a self-similar group is the group $Aut(X^*)$ of all automorphisms of the tree $X^*$, which consists of all permutations of the set of vertices preserving the root and the vertex-adjacency. The group $Aut(X^*)$ is an example of a profinite group equipped with a natural profinite topology. Every self-similar group $G$ is isomorphic to a subgroup of $Aut(X^*)$, with which it will be identified. For example, for a given permutation group $H < Sym(X)$, we identify the corresponding infinite wreath power $\mathcal{W}(H) = \ldots \wr H \wr H$ with the subgroup of automorphisms $g$ for which $\pi_{g,w} \in H$ for all $w \in X^*$. This is an example of a closed self-similar group. According to [3] the group $\mathcal{W}(H)$ is topologically finitely generated if and only if $H$ is perfect. For more about self-similar groups we refer to [10].

## 2.1 The Language of Wreath Products

Let $x_1, x_2, \ldots, x_d$ be some indexing of a certain alphabet $X$ and let $G$ be a group with a given self-similar action on the tree $X^*$. The concept of a vertex-permutation and a section is used to describe elements of $G$ in the language of wreath products. Namely, the relation

$$g = (g_{x_1}, g_{x_2}, \ldots, g_{x_d})\pi_g$$

is called the *wreath recursion* of an element $g \in G$ and describes the embedding of $G$ into the permutational wreath product $G \wr Sym(X)$, where the symmetric group acts on the direct power $G^d$ by permuting the factors. For example, wreath recursions for elements of the group $Aut(X^*)$ define an isomorphism $Aut(X^*) \simeq Aut(X^*) \wr Sym(X)$. We also define other embeddings into wreath products. Namely, for every $i \geq 0$ we use the restriction of the self-similar action of $G$ to the subtree $X^{\leq i} \subseteq X^*$ ending at the level $i$ and for every $g \in G$ we denote by $g|_i$ the restriction to the set $X^{\leq i}$ of the automorphism corresponding to $g$. Now, if we put $G|_i = \{g|_i\colon g \in G\}$, then we have an embedding of the group $G|_i$ into the permutational wreath product $Sym(X) \wr_{X^{i-1}} G|_{i-1}$, which is defined by the mapping

$$\Psi(g|_i) = ((\pi_{g,w_1}, \ldots, \pi_{g,w_D}), g|_{i-1}),$$

where $D = d^{i-1}$ and $w_1, w_2, \ldots, w_D$ is an ordering of the $(i-1)$-th level of the tree $X^*$ induced by indexing of the alphabet $X$.

## 2.2 Automaton Groups and Automaton Ranks

Obviously, every self-similar group $G$ has a generating set $S \subseteq G$ such that for every $x \in X$ and every $s \in S$ the section $s_x$ belongs to $S$ (suffice it to take $S = G$). We are especially interested in the case when there is a finite generating set $S \subseteq G$ with the above property.

**Definition 2.1.** *A finite subset $S \subseteq G$ of a self-similar group $G$ is called an automaton and the elements of $S$ are called states if the section $s_x$ belongs to $S$ for each $s \in S$ and each $x \in X$. A self-similar group generated (topologically if it is closed) by an automaton is called an automaton group.*

In the computational group theory an automaton $S \subseteq Aut(X^*)$ is usually described as a quadruple

$$(X, S, \varphi, \psi),$$

where the mapping $\varphi\colon S \times X \to S$, $\varphi(s,x) = s_x$ is called a transition function, and the mapping $\psi\colon S \times X \to X$, $\psi(s,x) = x^{\pi_s}$ is called an output function. Conversely, if $X$ is a certain alphabet, $S$ is a finite set, $\varphi$ and $\psi$ are the mappings of the form $\varphi\colon S \times X \to S$, $\psi\colon S \times X \to X$ and in addition for every $s \in S$ the mapping $x \mapsto \psi(s,x)$ defines a permutation of the alphabet, then the quadruple $(X, S, \varphi, \psi)$ defines an automaton in the group $Aut(X^*)$. The states of this automaton, identified with elements of the set $S$, transmit the words as follows: if $w = x_{i_1}x_{i_2}\dots x_{i_l}$ is a word, then $w^s = \psi(s_1, x_{i_1})\psi(s_2, x_{i_2})\dots\psi(s_l, x_{i_l})$, where $s_j \in S$ are defined recursively: $s_1 = s$, $s_{j+1} = \varphi(s_j, x_{i_j})$ for $1 \le j < l$. In particular, for every $s \in S$ and every letter $x \in X$ the section $s_x$ of the automorphism $s \in Aut(X^*)$ satisfies: $s_x = \varphi(s,x)$, and for the root-permutation $\pi_s$ we have: $x^{\pi_s} = \psi(s,x)$.

Another convenient way to describe an automaton is to draw its Moore diagram, which is a directed, labeled graph with the vertices identified with the states. Every vertex $s \in S$ is labeled by the corresponding root-permutation $\pi_s$, and if two vertices are of the form $s$ and $s_x$ for some $s \in S$ and $x \in X$ then we draw an arrow labeled by $x$ starting in $s$ and ending in $s_x$. To make a diagram of the automaton clear, we replace a large number of arrows connecting two given vertices and having the same direction by one multi-arrow labeled by suitable letters, and if the labelling of such a multi-arrow starting from a given vertex follows from the labelling of other arrows starting from this vertex, we will omit this labelling.

An automaton group $G$ generated by an automaton $S = \{s_1, \dots, s_k\}$ is uniquely determined by the wreath recursions of the states:

$$\begin{cases} s_1 = (q_{11}, q_{12}, \dots, q_{1d})\tau_1, \\ s_2 = (q_{21}, q_{22}, \dots, q_{2d})\tau_2, \\ \vdots \\ s_k = (q_{k1}, q_{k2}, \dots, q_{kd})\tau_k, \end{cases} \tag{1}$$

where $q_{ij} \in S$ denotes the section of $s_i$ at $x_j \in X$ and $\tau_i$ denotes the root permutation of $s_i$. Conversely, if $S = \{s_1, \dots, s_k\}$ is an arbitrary set of $k$ symbols, then formulas (1), where $\tau_i \in Sym(X)$ and $q_{ij} \in S$ $(1 \leq i \leq k,\ 1 \leq j \leq d)$, uniquely define an automaton group $G < Aut(X^*)$ with $S$ as an automaton generating $G$.

**Definition 2.2.** *The automaton rank $ar(G,d)$ of a group $G$ with respect to self-similar actions of $G$ on a $d$-ary $(d \geq 2)$ tree of words is the minimal number of states in an automaton generating $G$ (topologically if $G$ is closed). We will write $ar(G,d) = \infty$ if $G$ is not an automaton group with respect to any its self-similar action on a $d$-ary tree of words.*

It is worth to note that automaton ranks of groups can be much larger than their abstract ranks. Indeed, for any infinite collection $\{G_i : i \in I\}$ of pairwise nonisomorphic groups and any $d \geq 2$ the set $\{ar(G_i,d) : i \in I\}$ of their automaton ranks is unbounded (in particular may contain $\infty$). This follows from a simple observation that for every $d \geq 2$ and every $k \geq 1$ there is only finitely many $k$-state automata over a $d$-letter alphabet.

**Lemma 2.1.** *If $d' \geq d$, then $ar(G,d') \leq ar(G,d)$.*

*Proof.* If $G$ is generated by an automaton $S = \{s_1, \ldots, s_k\}$ over the alphabet $X = \{x_1, \ldots, x_d\}$ and the states of $S$ are defined by wreath recursions (1), then $G$ is also generated by an automaton $S' = \{s'_1, \ldots, s'_k\}$ over the alphabet $X' = \{x_1, \ldots, x_d, x_{d+1}, \ldots, x_{d'}\}$ with the states defined by wreath recursions

$$\begin{cases} s'_1 = (q_{11}, q_{12}, \ldots, q_{1d}, s'_1, \ldots, s'_1)\tau'_1, \\ \vdots \\ s'_k = (q_{k1}, q_{k2}, \ldots, q_{kd}, s'_k, \ldots, s'_k)\tau'_k, \end{cases}$$

where the permutations $\tau'_i \in Sym(X')$ $(1 \leq i \leq k)$ are defined as follows: $x^{\tau'_i} = x^{\tau_i}$ for $x \in X$ and $x^{\tau'_i} = x$ for $x \in X' \setminus X$. The claim follows. □

*Example 2.1.* An automaton rank $ar(\mathbb{Z},d)$ of the infinite cyclic group $\mathbb{Z}$ is equal to 2 for every $d \geq 2$. Indeed, $\mathbb{Z}$ can not be generated by an automaton which is a singleton $\{a\}$, as the state $a$ would be of finite order. But if a state $a$ is defined by the wreath recursion $a = (a, e, \ldots, e)\pi$, where $e$ is the so called neutral state (that is $e = Id_{X^*}$) and $\pi$ as a cycle of the length $d$, then the set $\{e, a\}$ is an automaton generating $\mathbb{Z}$. In general, for the free abelian group $\mathbb{Z}^k$ of rank $k \geq 1$ we have the estimations $k = r(\mathbb{Z}^k) \leq ar(\mathbb{Z}^k, d) \leq ar(\mathbb{Z}^k, 2) \leq k+1$. The last inequality follows from the construction of the "sausage" automaton $S = \{e, a_1, \ldots, a_k\}$ over the binary alphabet with states defined by wreath recursions: $a_1 = (e, a_n)(1,2)$, $a_i = (a_{i-1}, a_{i-1})id$ for $1 < i \leq k$. The automaton $S$ generates $\mathbb{Z}^k$ (see for example [6] or [10]).

*Example 2.2.* The combined results of [14] and [15] show that the nonabelian free group $F_k$ of rank $k \geq 3$ is generated by a $k$-state automaton over a 2-letter alphabet. This implies $ar(F_k, d) = r(F_k) = k$ for every $d \geq 2$ and every $k \geq 3$. However, the problem of finding $ar(F_2, d)$ for any $d \geq 2$ is open. For $d = 2$

we only have the estimation $4 \leq ar(F_2, 2) \leq 6$. Indeed, from the one hand, there are exactly six groups generated by 2-state automata over a 2-letter alphabet (see [7]) and $F_2$ is not among them. Furthermore, according to [4], among groups generated by 3-state automata over a 2-letter alphabet the only nonabelian free group is $F_3$. On the other hand, there is a 6-state automaton over a 2-letter alphabet generating $F_2$ (see [10]).

*Example 2.3.* We know (see [1]) that for any relatively prime integers $n, d > 1$ the Baumslag-Solitar group $BS(1, n) = \langle a, t\colon tat^{-1} = a^n \rangle$ is generated by an $n$-state automaton over a $d$-letter alphabet. In particular, $ar(BS(1, n), d) \leq n$ for odd $n > 1$ and all $d \geq 2$. On the other hand, for different $n$ the groups $BS(1, n)$ are not isomorphic. Thus for every $d \geq 2$ and every $m$ there is $n$ such that the automaton rank $ar(BS(1, n), d)$ is finite and not smaller than $m$. This contrasts with the fact that each $BS(1, n)$ is 2-generated.

*Example 2.4.* Let $H$ be a perfect transitive permutation group on a $d$-element set. Let $\{\alpha_1, \ldots, \alpha_k\}$ be an arbitrary generating set. According to [3] if the action of $H$ is not free, then the infinite wreath power $\mathcal{W}(H)$ is topologically generated by the automaton $\{e, a_1, \ldots, a_k, b_1, \ldots, b_k\}$ with the states $a_i$, $b_i$ described by wreath recursions: $a_i = (a_i, b_i, e, \ldots, e)id$, $b_i = (e, \ldots, e)\alpha_i$. In particular, the automaton rank $ar(\mathcal{W}(H), d)$ is not greater than $2k + 1$.

## 3 Automaton Ranks of Free Abelian Groups

We start with the following

**Proposition 3.1.** *There is no $k$-state ($k \geq 1$) automaton over a 2-letter alphabet which generates a free abelian group of rank $k$.*

*Proof.* Suppose that the statement is not true, and let $\{a_1, \ldots, a_k\}$ be an automaton over the alphabet $\{1, 2\}$ which generates $\mathbb{Z}^k$. The wreath recursions for the states $a_i$ ($1 \leq i \leq k$) are of the form $a_i = (a_{i'}, a_{i''})\pi_i$, where $i', i'' \in \{1, \ldots, k\}$ and $\pi_i \in \{id, (1, 2)\}$. Directly from these recursions we see that every section of $a_i^\eta$ ($1 \leq i \leq k$, $\eta \in \{-1, 1\}$) is equal to $a_{i'}^\eta$ for some $i' \in \{1, \ldots, k\}$. In general, let $g = a_1^{s_1} \ldots a_k^{s_k} \in \mathbb{Z}^k$ be arbitrary, and let $s = s_1 + \ldots + s_k$. By easy induction on $|s_1| + \ldots + |s_k|$ we show that every section of $g$ is of the form $a_1^{s_1'} a_2^{s_2'} \ldots a_k^{s_k'}$, where the integers $s_i'$ are such that $s_1' + \ldots + s_k' = s$. Thus, since $\mathbb{Z}^k$ is generated freely by the states $a_i$ ($1 \leq i \leq k$), we see that $s \neq s'$ implies $(a_1^s)_w \neq (a_1^{s'})_{w'}$ for all $w, w' \in \{1, 2\}^*$. But Theorem 5.2 in [11] together with Theorem 2.12.1 in [10] implies that the self-similar action of a free abelian group generated by an automaton over a 2-letter alphabet is contracting. This means that there is a finite set $N \subseteq \mathbb{Z}^k$ such that for every $g \in \mathbb{Z}^k$ there is $i > 0$ such that $g_w \in N$ for every word $w$ from the $i$-th level of $X^*$. In particular, for every integer $s$ there is $w_s \in X^*$ such that $(a_1^s)_{w_s} \in N$. But from the above inequality we obtain: $(a_1^{s'})_{w_{s'}} \neq (a_1^s)_{w_s}$ for $s' \neq s$, which contradicts with the fact that the set $N$ is finite. The claim follows. $\square$

It turns out that the situation changes over the alphabet $X$ with at least 3 letters. To show this let us consider for every $k > 1$ the automaton $\mathcal{A} = \{a_1, \ldots, a_k\}$ over the alphabet $X = \{1, 2, 3\}$ with the states defined by following wreath recursions

$$a_1 = (a_k, a_1, a_1)(1, 2), \quad a_i = (a_{i-1}, a_{i-1}, a_{i-1})id, \quad 1 < i \leq k. \tag{2}$$

The Moore diagram of the automaton $\mathcal{A}$ is presented in Figure 1.

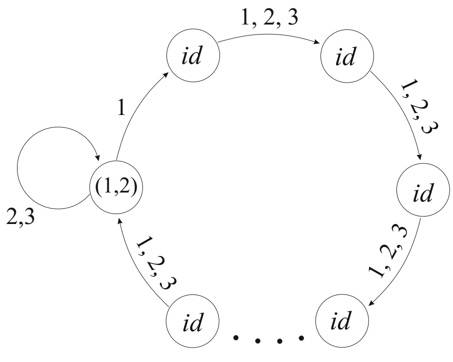


**Fig. 1.** The minimal automaton generating $\mathbb{Z}^k$

**Lemma 3.1.** *For all* $1 \leq i, j \leq k$ *we have* $a_i a_j = a_j a_i$.

*Proof.* By induction on the length $|w|$ of a word $w$ we show that $w^{a_i a_j} = w^{a_j a_i}$. For $|w| = 0$ it is obvious. Directly from the wreath recursions (2) we obtain the following wreath recursions for $a_i a_j$:

$$\begin{aligned} a_i a_j &= (a_{i-1}a_{j-1}, a_{i-1}a_{j-1}, a_{i-1}a_{j-1})id, \quad 1 < i, j \leq k, \\ a_1 a_i &= (a_k a_{i-1}, a_1 a_{i-1}, a_1 a_{i-1})(1, 2), \quad 1 < i \leq k, \\ a_i a_1 &= (a_{i-1} a_k, a_{i-1} a_1, a_{i-1} a_1)(1, 2), \quad 1 < i \leq k. \end{aligned}$$

In particular we have $\pi_{a_i a_j} = \pi_{a_j a_i}$ for all $1 \leq i, j \leq k$. Consequently, if $|w| = 1$, then we have $w^{a_i a_j} = w^{\pi_{a_i a_j}} = w^{\pi_{a_j a_i}} = w^{a_j a_i}$. By the above wreath recursions for $a_i a_j$ we see that for all $1 \leq i, j \leq k$ and every letter $x \in X$ the section of $a_i a_j$ at $x$ is equal to $a_{i'} a_{j'}$ for some $1 \leq i', j' \leq k$. Moreover, if $(a_i a_j)_x = a_{i'} a_{j'}$, then $(a_j a_i)_x = a_{j'} a_{i'}$. Now, let $w \in X^*$ be nonempty and let us assume the statement for all words of the length smaller than $|w|$. Then $w = xv$ for some $x \in X$ and $v \in X^*$. Let $1 \leq i, j \leq k$ and let us denote $x' = x^{\pi_{a_i a_j}} = x^{\pi_{a_j a_i}}$. By the above observation, there are $1 \leq i', j' \leq k$ such that $(a_i a_j)_x = a_{i'} a_{j'}$ and $(a_j a_i)_x = a_{j'} a_{i'}$. Since $|v| < |w|$, we obtain:

$$w^{a_i a_j} = x' v^{(a_i a_j)_x} = x' v^{a_{i'} a_{j'}} = x' v^{a_{j'} a_{i'}} = x' v^{(a_j a_i)_x} = w^{a_j a_i}.$$

The claim follows. □

**Lemma 3.2.** *The element $a_1$ is of infinite order.*

*Proof.* For any even number $s$ we obtain from the wreath recursion for $a_1$ the following wreath recursion for $a_1^s$:

$$a_1^s = (a_1^{s/2}a_k^{s/2}, a_1^{s/2}a_k^{s/2}, a_1^s)id. \tag{3}$$

From the wreath recursions $a_i = (a_{i-1}, a_{i-1}, a_{i-1})id$ $(1 < i \le k)$ we obtain for any $s$ the following wreath recursions:

$$a_i^s = (a_{i-1}^s, a_{i-1}^s, a_{i-1}^s)id, \quad 1 < i \le k. \tag{4}$$

Suppose that $a_1^s$ is an identity for some $s > 0$. Since the root-permutation of $a_1$ is the transposition, the number $s$ must be even. Thus by (3) we obtain: $a_k^{s/2} = a_1^{s/2}$. Since the root-permutation of $a_k$ is the identity, the number $s/2$ must be even. By (4) the section of $a_k^{s/2}$ at the word $3^{k-2}1$ is equal to $a_1^{s/2}$. By (3) the section of $a_1^{s/2}$ at the word $3^{k-2}1$ is equal to $a_1^{s/4}a_k^{s/4}$. Consequently $a_1^{s/2} = a_1^{s/4}a_k^{s/4}$ and thus $a_k^{s/4} = a_1^{s/4}$. By repeating we obtain $a_k^2 = a_1^2$, and further $a_1^2 = (a_k^2)_{3^{k-2}1} = (a_1^2)_{3^{k-2}1} = a_1a_k$. Consequently $a_k = a_1$, which is false. □

**Proposition 3.2.** *The automaton $\mathcal{A} = \{a_1, \ldots, a_k\}$ defined by recursions (2) generates a free abelian group of rank $k$.*

*Proof.* Suppose not, then there is a minimal positive integer $S$ which is a sum $|s_1| + \ldots + |s_k|$ of absolute values of exponents in a product $a_1^{s_1}a_2^{s_2}\ldots a_k^{s_k}$ defining $Id_{X^*}$. Let us denote $s = s_1 + \ldots + s_k$. By wreath recursions (2) we see that the section of the above product at the word $3^{k-1}$ is equal to $a_1^s$, and thus $s = 0$ by Lemma 3.2. If $s_1 \cdot s_i \ge 0$ for all $1 \le i \le k$, then $S = |s| = 0$, which is a contradiction with $S > 0$. Thus there is a minimal number $j > 1$ such that $s_1 \cdot s_j < 0$. The section of $a_1^{s_1}a_2^{s_2}\ldots a_k^{s_k}$ at the word $3^{j-2}$ is equal to $a_1^{s'}a_2^{s_j}\ldots a_{k-j+2}^{s_k}$, where $s' = s_1 + \ldots + s_{j-1}$. Since $(1,2)^{s'}$ is a root-permutation of the last product, the number $s'$ must be even. By the wreath recursion $a_1^{s'} = (a_1^{s'/2}a_k^{s'/2}, a_1^{s'/2}a_k^{s'/2}, a_1^{s'})id$ and by Lemma 3.1 we see that the section of $a_1^{s'}a_2^{s_j}\ldots a_{k-j+2}^{s_k}$ at the letter 1 is equal to the product $a_1^{s'/2+s_j}a_2^{s_{j+1}}\ldots a_{k-j+1}^{s_k}a_k^{s'/2}$. The above product defines $Id_{X^*}$ and the sum of absolute values of exponents in this product is equal to $S' = |s'/2 + s_j| + |s'/2| + |s_{j+1}| + \ldots + |s_k|$. Since $s's_j < 0$, we have $|s'/2 + s_j| < |s'/2| + |s_j|$, and thus $S' < S$. Since all $s_i$ $(1 \le i < j)$ in the sum $s' = s_1 + \ldots + s_{j-1}$ are of the same sign, we obtain: $|s'/2 + s_j| + |s'/2| = 0$ if and only if $s_i = 0$ for $1 \le i \le j$. Consequently $S' = 0$ if and only if $s_i = 0$ for $1 \le i \le k$. Thus $0 < S' < S$, which contradicts the definition of $S$. □

*Proof (of Theorem 1.1).* For $k = 1$ the statement follows by Example 2.1. For $k > 1$ and $d = 2$ the statement follows by Proposition 3.1 and the construction of a 'sausage' automaton from Example 2.1. For $k > 1$ and $d > 2$ the statement follows by Lemma 2.1 and Proposition 3.2. □

## 4 The Groups $\ldots \wr H \wr H$ as Automaton Groups

Let $H$ be a perfect transitive permutation group on a finite set $X$ generated by 2 elements $\alpha$, $\beta$ such that: (i) $\alpha$ has a fixed point, i.e. there is $x_0 \in X$ such that $x_0^\alpha = x_0$, (ii) $\beta$ decomposes into disjoint cycles of the same length. Then there are $m, r \geq 1$ such that $\beta$ decomposes into $r$ cycles each of the length $m$: $\beta = (x_{11}, \ldots, x_{1m})(x_{21}, \ldots, x_{2m}) \ldots (x_{r1}, \ldots, x_{rm})$, where $x_{rm} = x_0$. Let $X = \{x_{11}, x_{21}, \ldots, x_{r1}, \ldots, x_0\}$ be an indexing of the alphabet such that the letters $x_{i1}$ $(1 \leq i \leq r)$ are in the first $r$ positions and the letter $x_{rm} = x_0$ is in the last position. Let $\mathcal{A} = \{e, a, b\}$ be a 3-state automaton over $X$ with the states defined by the following wreath recursions:

$$a = (e, \ldots, e, a)\alpha, \quad b = (b, \ldots, b, e, \ldots, e)\beta, \tag{5}$$

where all $b$'s on the right-hand side of the second recursion are in the first $r$ positions (i.e. positions corresponding to the letters $x_{i,1}$, $1 \leq i \leq r$). The Moore diagram of the automaton $\mathcal{A}$ is depicted in Figure 2.

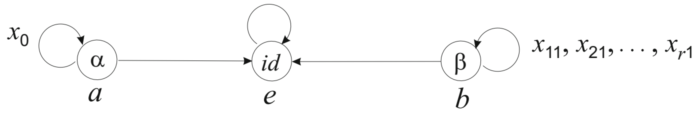


**Fig. 2.** An automaton generating $\ldots \wr H \wr H$

**Proposition 4.1.** *The infinite wreath power $\mathcal{W}(H) = \ldots \wr H \wr H$ is topologically generated by the automaton $\mathcal{A}$ defined by wreath recursions (5).*

*Proof.* Let $G$ be a group generated by the automaton $\mathcal{A}$, and for $i \geq 1$ let $G|_i$ be a group generated by restrictions $a_i = a|_i$ and $b_i = b|_i$ of the states $a$, $b$ to the subtree $X^{\leq i}$. We must show that $G|_i = \mathcal{W}(H)|_i$ for every $i \geq 1$. Since the elements $\alpha$, $\beta$ generate $H$ and $a_1 = \alpha$, $b_1 = \beta$, we obtain: $G|_1 = H = \mathcal{W}(H)|_1$. Let us assume that $G|_{i-1} = \mathcal{W}(H)|_{i-1}$ for some $i > 1$. Let $D = |X|^{i-1}$ and let

$$w_1 = x_{11} \ldots x_{11}, \ \ldots \ , w_D = x_0 \ldots x_0$$

be an ordering of $X^{i-1}$ induced by the above indexing of $X$. The embedding $\Psi\colon G|_i \to H \wr_{X^{i-1}} G|_{i-1}$ is induced by

$$\Psi(a_i) = ((\pi_{a,w_1}, \ldots, \pi_{a,w_D}), a_{i-1}), \quad \Psi(b_i) = ((\pi_{b,w_1}, \ldots, \pi_{b,w_D}), b_{i-1}).$$

By induction assumption we have $G|_{i-1} = \mathcal{W}(H)|_{i-1}$. Thus, since the elements $a_{i-1}$, $b_{i-1}$ generate $G|_{i-1}$, we see that $G|_i$ projects under $\Psi$ onto the whole $\mathcal{W}(H)|_{i-1}$. Thus to prove our proposition it is enough to show that $\Psi(G|_i)$ contains $H^D \times \{Id\}$, where $Id = Id_{X^{\leq i-1}}$. We show at first that for every

$\gamma \in H$ there is $k \in G|_i$ such that $\Psi(k) = ((id, \dots, id, \gamma), Id)$. For the powers $b^{m^j}$ $(j \geq 1)$ the following wreath recursions hold $b^{m^j} = (b^{m^{j-1}}, \dots, b^{m^{j-1}})id$. This implies: $(b_{i-1})^{m^{i-1}} = Id$ and $\pi_{b^{m^{i-1}}, w_j} = \beta$ for $1 \leq j \leq D$. From the wreath recursion for $a$ we obtain $\pi_{a,w_j} = id$ for $1 \leq j < D$ and $\pi_{a,w_D} = \alpha$, as well as $a_{i-1}$ stabilizes $w_D$. In particular, we have

$$\Psi(a_i) = ((id, \dots, id, \alpha), a_{i-1}), \quad \Psi(b_i^{m^{i-1}}) = ((\beta, \dots, \beta), Id). \tag{6}$$

Since $H$ is a perfect group generated by $\alpha$ and $\beta$, we see that every $\gamma \in H$ is represented by a group word $W(\alpha, \beta)$ on $\alpha$ and $\beta$ in which the sums of exponents on generators are both equal to zero. Then from the fact that $a_{i-1}$ stabilizes $w_D$ and from equalities (6), we obtain the equality $\Psi(k) = ((id, \dots, id, \gamma), Id)$, where the element $k = W(a_i, b_i^{m^{i-1}})$ arises from $W(\alpha, \beta)$ by the substitution of any $\alpha$ for $a_i$ and any $\beta$ for $b_i^{m^{i-1}}$. Obviously, the group $G|_{i-1} = \mathcal{W}(H)|_{i-1}$ acts transitively on $X^{i-1}$. Thus for every $1 \leq j \leq D$ there is $g_j \in G|_{i-1}$ such that $w_j^{g_j} = w_D$. For $1 \leq j \leq D$ let $\gamma_j \in H$ be arbitrary, and let $h_j, k_j \in G|_i$ be such that $\Psi(h_j) = ((\pi_{h_j,w_1}, \dots, \pi_{h_j,w_D}), g_j)$ and $\Psi(k_j) = ((id, \dots, id, \delta_j), Id)$, where $\delta_j = \pi_{h_j,w_j} \gamma_j \pi_{h_j,w_j}^{-1}$. Then we have: $\Psi(h_j^{-1} k_j h_j) = ((id, \dots, id, \gamma_j, id, \dots, id), Id)$, where $\gamma_j$ on the right-hand side of the above recursion is in the $j$-th position. Finally, for the product $g = (h_1^{-1} k_1 h_1)(h_2^{-1} k_2 h_2) \dots (h_D^{-1} k_D h_D)$ we obtain $\Psi(g) = ((\gamma_1, \dots, \gamma_D), Id)$. The claim follows. □

As a direct consequence of Proposition 4.1 we obtain Theorem 1.2.

## 4.1 The Case of Alternating Groups $A_n$, $n > 4$

For every $n > 4$ the alternating group $A_n$ is a transitive group of permutations on the set $X = \{1, 2, \dots, n\}$. This is a perfect group (as a simple nonabelian group). The next Lemma shows that the group $A_n$ is generated by two elements $\alpha$, $\beta$ which satisfy the conditions (i)-(ii).

**Lemma 4.1.** *The alternating group $A_n$ $(n > 4)$ is generated by the set $\{\alpha, \beta\}$, where $\alpha = (1, 2, 3)$ and $\beta = (1, 2, \dots, n)$ or $\beta = (1, 3, \dots, n-1)(2, 4, \dots, n)$ depending on the parity of $n$.*

*Proof.* The group $A_n$ is generated by 3-cycles $(i, j, k)$ with $i, j, k \geq 1$, all distinct. By equalities: $(i, j, k) = (1, l, i)(1, j, k)(1, i, l)$, $i, j, k, l \geq 1$, all distinct, $(1, i, j) = (1, 2, j)(1, 2, i)(1, 2, j)^{-1}$, $i \neq j$, $i, j \geq 3$, we have to show that each 3-cycle $(1, 2, i)$ with $i \geq 3$ is a product of $\alpha$, $\beta$ or their inverses. Indeed, in case $n$ even we define: $\alpha_0 = \alpha$, $\alpha_i = \beta^{-1} \alpha_{i-1} \beta$ for $1 \leq i \leq (n-2)/2$. By easy calculation we obtain: $\alpha_i = (2i+1, 2i+2, 2i+3)$ for $0 \leq i < (n-2)/2$ and $\alpha_{(n-2)/2} = (n-1, n, 1)$. Then, for the elements $\gamma_i$ $(1 \leq i \leq n-2)$ defined as follows: $\gamma_1 = \alpha$ and $\gamma_i = \alpha_{i'}^{-1} \gamma_{i-1} \alpha_{i'}$ for $i > 1$, where $i' = [i/2]$, we easily verify: $\gamma_i = (1, 2, i+2)$. In case $n$ is odd we define similarly: $\alpha_0 = \alpha$, $\alpha_i = \beta^{-1} \alpha_{i-1} \beta$ for $1 \leq i \leq n$. Then for the elements $\gamma_i$ $(1 \leq i \leq n-2)$ defined as follows: $\gamma_1 = \alpha$ and $\gamma_i = \alpha_{i-1}^{-1} \gamma_{i-1} \alpha_{i-1}$ for $i > 1$ we obtain $\gamma_i = (1, 2, i+2)$. The claim follows. □

## 4.2 The Case $H = PSL_2(p)$, $p > 3$

Let $p > 3$ be a prime and let $X = \mathbb{F}_p \cup \{\infty\}$ be the projective line over the field $\mathbb{F}_p = \{0, 1, \ldots, p-1\}$. The projective special linear group $PSL_2(p)$ is the group of linear-fractional transformations $x \mapsto (ax+b)/(cx+d)$ of the set $X$, where $a, b, c, d \in \mathbb{F}_p$ and $ad - bc = 1$ (we make standard conventions about $\infty$, for example $a/0 = \infty$ for $a \neq 0$ and $(a\infty + b)/(c\infty + d) = a/c$). The above transformations determine $PSL_2(p)$ as a perfect and transitive group of permutations of $X$. We show that there is a 2-element generating set in $PSL_2(p)$ which satisfies (i)-(ii).

It is known that $PSL_2(p)$ is generated by the following two transformations: $\alpha\colon x \mapsto x+1$, $\beta\colon x \mapsto -1/x$. Since $\beta$ is an involution and $\infty$ is a fixed point of $\alpha$, we see that $\alpha$, $\beta$ satisfy (i)-(ii) if and only if $-1$ is not a square in $\mathbb{F}_p$, or equivalently, $p \equiv 3 \pmod 4$. Now, we show how we may skip the above restriction for $p$. To this end for every $r \in \mathbb{F}_p$ we choose in the group $PSL_2(p)$ the transformation $\beta_r\colon x \mapsto -1/x + r$. We see by the equality $\beta = \beta_r\alpha^{-r}$ and the fact that the permutations $\alpha$, $\beta$ generate $PSL_2(p)$ that for every $r \in \mathbb{F}_p$ the transformations $\alpha$, $\beta_r$ also generate $PSL_2(p)$. Now, let us consider the sequence $(u_n)_{n\geq 0}$ defined recursively: $u_0 = 0$, $u_1 = 1$ and $u_{n+2} = ru_{n+1} - u_n$ for $n \geq 0$.

**Lemma 4.2.** *We have $\beta_r^n\colon x \mapsto (u_{n+1}x - u_n)/(u_nx - u_{n-1})$, where $\beta_r^n$ denotes the $n$-th power of $\beta_r$.*

*Proof.* We have $\beta_r^1(x) = (rx-1)/x = (u_2x - u_1)/(u_1x - u_0)$ for every $x \in X$. Let us assume the statement for some $n \geq 1$. Then for every $x \in X$ we have:

$$\beta_r^{n+1}(x) = \beta_r(\beta_r^n(x)) = -\frac{1}{\beta_r^n(x)} + r = \frac{u_{n-1} - u_nx}{u_{n+1}x - u_n} + r =$$
$$= \frac{(ru_{n+1} - u_n)x - (ru_n - u_{n-1})}{u_{n+1}x - u_n} = \frac{u_{n+2}x - u_{n+1}}{u_{n+1}x - u_n}.$$

The claim follows. □

**Lemma 4.3.** *There is $r \in \mathbb{F}_p$ such that $\beta_r$ defines a nontrivial permutation of the set $X$, which decomposes into disjoint cycles of the same length.*

*Proof.* Since $p > 3$ is a prime, the sequence $4 \cdot 1, 4 \cdot 2, \ldots, 4 \cdot (p-1)$ contains all nonzero elements of the field $\mathbb{F}_p$. Since $\mathbb{F}_p \setminus \{0\}$ contains both squares and non-squares, there is $i \in \mathbb{F}_p \setminus \{0, p-1\}$ such that the set $\{4i, 4(i+1)\}$ contains both a square and a non-square. Let us denote $r = 4i+2$. Since the product of a nonzero square and a non-square is a non-square, the element $r^2 - 4 = 4i \cdot 4(i+1)$ is a non-square. The permutation $\beta_r$ is nontrivial as $\beta_r(\infty) = r$. Let $m$ be the length of the shortest cycle in the decomposition of $\beta_r$ into disjoint cycles. We show that all cycles in this decomposition are of the length $m$. To this end we have to show that $\beta_r^m$ is the identity permutation. Since $\infty$ is not a fixed point of $\beta_r$, the shortest cycle contains an element different than $\infty$. Thus $\beta_r^m(x_0) = x_0$ for some $x_0 \in \mathbb{F}_p$. By Lemma 4.2 we have $(u_{m+1}x_0 - u_m)/(u_mx_0 - u_{m-1}) = x_0$, or equivalently: $u_mx_0^2 - x_0(u_{m+1} + u_{m-1}) + u_m = 0$. By definition of the sequence $(u_i)$ we have

$u_{m+1}+u_{m-1} = ru_m$, and thus $u_m(x_0^2 - rx_0 + 1) = 0$. Since $r^2 - 4$ is a non-square, we have $x_0^2 - rx_0 + 1 \neq 0$. Consequently $u_m = 0$ and $u_{m+1} = -u_{m-1}$. Thus for every $x \in X$ we have $\beta_r^m(x) = (u_{m+1}x - u_m)/(u_m x - u_{m-1}) = -u_{m+1}x/u_{m-1} = x$, that is $\beta_r^m = Id_X$. The claim follows. □

**Corollary 4.1.** *Let $H$ be an alternating group $A_n$ of degree $n > 4$ or a projective special linear group $PSL_2(p)$ with a prime $p > 3$. Then the infinite wreath power $\mathcal{W}(H) = \ldots \wr H \wr H$ has a 3-state automaton realization. In particular, the automaton rank $ar(\mathcal{W}(A_n), n)$ as well as the automaton rank $ar(\mathcal{W}(PSL_2(p)), p+1)$ is not greater than 3.*

## References

1. Bartholdi, L., Sunik, Z.: Some solvable automaton groups. Contemporary Mathematics 394, 11–29 (2006)
2. Bhattacharjee, M.: The probability of generating certain profinite groups by two elemets. Israel Journal of Mathematics 86, 311–329 (1994)
3. Bondarenko, I.: Finite generation of iterated wreath products. Archiv der Mathematik 95(4), 301–308 (2010)
4. Bondarenko, I., Grigorchuk, R., Kravchenko, R., Muntyan, Y., Nekrashevych, V., Sunic, Z., Savchuk, D.: On classification of groups generated by 3-state automata over a 2-letter alphabet. Algebra and Discrete Mathematics 1, 1–163 (2008)
5. Epstein, D.B.A., Cannon, J.W., Holt, D.F., Levy, S.V., Paterson, M.S., Thurston, W.P.: Word processing in groups. Jones and Bartlett Publishers, Boston (1992)
6. Grigorchuk, R., Nekrashevych, V., Sushchanskyy, V.: Automata, dynamical systems and groups. Proceedings of Steklov Institute of Mathematics 231, 128–203 (2000)
7. Grigorchuk, R., Zuk, A.: The lamplighter group as a group generated by a 2-state automaton and its spectrum. Geometriae Dedicata 87, 209–244 (2001)
8. Kharlampovich, O., Khoussainov, B., Miasnikov, A.: From automatic structures to automatic groups, 35 pages (2011), arXiv:1107.3645v2 [math.GR]
9. Khoussainov, B., Nerode, A.: Automatic Presentations of Structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995)
10. Nekrashevych, V.: Self-similar groups. Mathematical Surveys and Monographs, vol. 117. Amer. Math. Soc., Providence (2005)
11. Nekrashevych, V., Sidki, S.: Automorphisms of the binary tree: state-closed subgroups and dynamics of 1/2-endomorphisms. In: Groups: Topological, Combinatorial and Arithmetic Aspects. LMS Lecture Notes Series, vol. 311, pp. 375–404 (2004)
12. Quick, M.: Probabilistic generation of wreath products of non-abelian finite simple groups. Commun. Algebra 32(12), 4753–4768 (2004)
13. Silva, P., Steinberg, B.: On a class of automata groups generalizing lamplighter groups. Internat. J. Algebra Comput. 15(5-6), 1213–1234 (2005)
14. Steinberg, B., Vorobets, M., Vorobets, Y.: Automata over a binary alphabet generating free groups of even rank. Internat. J. Algebra Comput. 21(1-2), 329–354 (2011)
15. Vorobets, M., Vorobets, Y.: On a series of finite automata defining free transformation groups. Groups Geom. Dyn. 4(2), 377–405 (2010)

# One-Way Reversible and Quantum Finite Automata with Advice

Tomoyuki Yamakami

Department of Information Science, University of Fukui
3-9-1 Bunkyo, Fukui 910-8507, Japan

**Abstract.** We examine characteristic features of reversible and quantum computations in the presence of supplementary information, known as advice. In particular, we present a simple, algebraic characterization of languages recognized by one-way reversible finite automata with advice. With a further elaborate argument, a similar but slightly weaker result for bounded-error one-way quantum finite automata is also proven. As an immediate application of those features, we demonstrate certain containments and separations among various standard language families that are suitably assisted by advice.

**Keywords:** Reversible Finite Automaton, Quantum Finite Automaton, Advice, Regular Language, Context-free Language.

## 1 Background, Motivations, and Challenges

From theoretical and practical interests, we wish to promote our basic understandings of the exotic behaviors of reversible and quantum computations by examining machine models of those computations, in particular, reversible finite automata and quantum finite automata. Of various types of such automata (e.g., [10]), in order to make our argument clear and transparent, we initiate our study by limiting our focus within one of the simplest automaton models: *one-way deterministic reversible finite automata* (or 1rfa's, in short) and *one-way measure-many quantum finite automata* (or 1qfa's, thereafter). Our 1qfa scans each input-tape cell by moving a single tape head only in one direction (without stopping) and performs a *(projection) measurement* immediately after every head move, until the tape head scans the right endmarker. From a theoretical perspective, the 1qfa's with more than 7/9 success probability are essentially as powerful as 1rfa's [1], and therefore it is possible to view 1rfa's as a special case of 1qfa's. Notice that, for bounded-error 1qfa's, it is not always possible to make a sufficient amplification of success probability. This is one of many features that make the analysis of 1qfa's quite different from that of polynomial-time quantum Turing machines. These intriguing features of 1qfa's, on the contrary, have kept stimulating our research since their introduction in late 1990s. Back in an early period of study, numerous unconventional features have been revealed. For instance, as Ambainis and Freivalds [1] demonstrated, certain quantum finite automata can be built more state-efficiently than deterministic finite automata.

However, as Kondacs and Watrous [4] proved, a certain regular language cannot be recognized by any 1qfa with bounded-error probability. Moreover, by Brodsky and Pippenger [2], no bounded-error 1qfa recognizes languages accepted by minimal finite automata that lack a so-called *partial order condition.* The latter two facts suggest that the language-recognition power of 1qfa's is hampered by their own inability to generate useful quantum states from input information. To overcome such drawbacks, a simple, straightforward way is to appeal to an outside information source.

In a wide range of literature, various notions of classical machines equipped with supplemental information have been extensively studied. Because of its simplicity, we consider only Karp and Lipton's [3] style of information, known as *(deterministic) advice*, a piece of which encodes additional data, given in parallel with a standard input, into a single string (called an *advice string*), which depends only on the size of the input. A series of recent studies [6–9] on classical one-way finite automata with advice have unearthed advice's delicate roles. These advised automaton models have immediate connections to other fields, including one-way communication, random access coding, and two-player zero-sum games. Two central questions concerning the advice are: how can we encode necessary information into a piece of advice before a computation starts and, as a computation proceeds step by step, how can we decode and utilize such information stored inside the advice?

For the polynomial-time quantum Turing machines, there is a rich literature on the power and limitation of advice (see, for instance, [5]); disappointingly, little is known for the roles of advice when it is given to finite automata, in particular, 1rfa's and 1qfa's. For bounded-error 1qfa's, for instance, an immediate advantage of taking such advice is the elimination of both endmarkers placed on an input tape. Beyond such an obvious advantage, however, there are numerous challenges in the study of advice. The presence of advice tends to make an analysis of underlying computations quite difficult and it often demands quite different kinds of proof techniques. As a quick example, a standard *pumping lemma*—a typical proof technique that showcases the non-regularity of a given language—is not quite serviceable to advised computations; therefore, we need to develop other tools (e.g., a swapping lemma [7]) for them. On a similar light, certain advised 1qfa's violate the aforementioned criterion of the partial order condition, and this fact makes a proof technique of [4] inapplicable to, for example, a class separation between regular languages and languages accepted by bounded-error advised 1qfa's.

To aim at analyzing the behaviors of 1qfa's as well as 1rfa's, in this paper, we first need to lay out a ground work to (1) capture the fundamental features of those automata when advice is given to boost their language-recognition power and (2) develop methodology necessary to lead to collapses and separations of advised language families. In particular, the aforementioned difficulties surrounding the advice for 1qfa's motivate us to seek different kinds of proof techniques.

**Major Contributions.** In this paper, we will prove two main theorems. As the first main theorem (Theorem 1), with an elaborate argument, we will show a machine-independent necessary and sufficient conditions for languages to be recognized by 1rfa's taking advice. Moreover, for bounded-error 1qfa's with advice, we will give a machine-independent sufficient condition as the second theorem (Theorem 6). These two conditions exhibit certain behavioral characteristics of 1rfa's and 1qfa's when appropriate advice is provided. Applying these theorems further, we can prove several class separations among advised language families. These separations indicate, to some extent, inherent strengths and weaknesses of reversible and quantum computations even in the presence of advice.

## 2 Basic Terminology

We briefly explain fundamental notions and notations used in this paper. Let $\mathbb{N}$ be the set of all *nonnegative integers*. For any pair $m,n \in \mathbb{N}$ with $m \leq n$, the *integer interval* $[m,n]_{\mathbb{Z}}$ denotes the set $\{m, m+1, m+2, \ldots, n\}$ and $[n]$ is the shorthand for $[1,n]_{\mathbb{Z}}$. An *alphabet* $\Sigma$ is a finite nonempty set and a *string* over $\Sigma$ is a series of symbols taken from $\Sigma$. In particular, the *empty string* is always denoted $\lambda$ and we set $\Sigma^+ = \Sigma^* - \{\lambda\}$. The *length* $|x|$ of a string $x$ is the total number of symbols in $x$. For any string $x$ and any number $n \in \mathbb{N}$, $Pref_n(x)$ expresses the string consisting of the first $n$ symbols of $x$ whenever $|x| \geq n$. In particular, $Pref_0(x) = \lambda$. A *language* over $\Sigma^*$ is a subset of $\Sigma^*$. We conveniently treat a language $L$ as a "function" defined as $L(x) = 1$ (resp. $L(x) = 0$) if $x \in L$ (resp. $x \notin L$). Given an alphabet $\Gamma$, a *probability ensemble* over $\Gamma^*$ means an infinite series $\{D_n\}_{n\in\mathbb{N}}$ of probability distributions, in which each $D_n$ maps $\Gamma^n$ to the unit real interval $[0,1]$. For ease of our later analysis, we always assume that (1) every finite automaton is equipped with a single read-only input tape, on which each input string is initially surrounded by two endmarkers (the left endmarker ¢ and the right endmarker \$) and (2) every automaton moves its tape head rightward without stopping. Set $\check{\Sigma} = \Sigma \cup \{¢, \$\}$. For brevity, we abbreviate as *1dfa* a one-way deterministic finite automaton. Let REG, CFL, and DCFL denote, respectively, the families of *regular languages*, of *context-free languages*, and of *deterministic context-free languages*.

To introduce a notion of *(deterministic) advice* that is fed to finite automata beside input strings, we adopt the "track" notation of [6]. For two symbols $\sigma \in \Sigma$ and $\tau \in \Gamma$, where $\Sigma$ and $\Gamma$ are two alphabets, the notation $[\,{}^{\sigma}_{\tau}\,]$ expresses a new symbol made up of $\sigma$ and $\tau$. On the input tape, this new symbol is written in a single tape cell, which is split into two tracks, whose upper track contains $\sigma$ and the lower one contains $\tau$. Notice that an automaton's tape head scans two track symbols $\sigma$ and $\tau$ in $[\,{}^{\sigma}_{\tau}\,]$ at once. For two strings $x$ and $y$ of the same length $n$, $[\,{}^{x}_{y}\,]$ denotes a concatenated string $[\,{}^{x_1}_{y_1}\,][\,{}^{x_2}_{y_2}\,]\cdots[\,{}^{x_n}_{y_n}\,]$, provided that $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_n$. An *advice function* is a function mapping $\mathbb{N}$ to $\Gamma^*$, where $\Gamma$ is an alphabet, called an *advice alphabet*. The advised language family REG/$n$ of Tadaki et al. [6] is the collection of all languages $L$ over certain alphabets $\Sigma$ such that there exist a 1dfa $M$, an advice alphabet $\Gamma$, and an advice function $h : \mathbb{N} \to \Gamma^*$ for which (i) for every length $n \in \mathbb{N}$, $|h(n)| = n$ and (ii) for every string $x \in \Sigma^*$, $x \in L$ iff $M$ accepts $[\,{}^{x}_{h(|x|)}\,]$. Similarly, CFL/$n$ is defined in [7, 9].

## 3 Reversible Finite Automata with Advice

Since its introduction, the usefulness of advice has been demonstrated for various models of underlying computations. Following this line of study, we begin with examining characteristic features of a reversible language family that is naturally assisted by advice. A *one-way (deterministic) reversible finite automaton* (or 1rfa, in short) is a 1dfa $M = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ that satisfies the following "reversibility condition": for every inner state $q \in Q$ and every symbol $\sigma \in \Sigma$, there exists at most one inner state $q' \in Q$ for which $\delta(q', \sigma) = q$. Formally, the advised language family 1RFA/$n$ is composed of all languages $L$ over alphabets $\Sigma$ such that there exist a 1rfa $M$ and an advice function $h$ satisfying (i) $|h(n)| = n$ for any length $n \in \mathbb{N}$ and (ii) $M([\,{}^{x}_{h(|x|)}]) = L(x)$ for every string $x \in \Sigma^*$. Similar to 1RFA $\subseteq$ REG, the containment 1RFA/$n$ $\subseteq$ REG/$n$ also holds.

To understand the reversible behaviors of 1rfa's in the presence of advice, we wish to seek a machine-independent, algebraic characterization of every language in 1RFA/$n$, which turns out to be a useful tool in studying the computational complexity of languages in 1RFA/$n$. Here, we describe the first main theorem, Theorem 1, of this paper.

**Theorem 1.** *Let $S$ be any language over an alphabet $\Sigma$. The following two statements are equivalent. Let $\Delta = \{(x, n) \mid x \in \Sigma^*, n \in \mathbb{N}, |x| \le n\}$.*

1. *$S$ is in* 1RFA/$n$.
2. *There is an equivalence relation $\equiv_S$ over $\Delta$ such that*
   *(i) the set $\Delta/\equiv_S$ is finite, and*
   *(ii) for any length parameter $n \in \mathbb{N}$, any symbol $\sigma \in \Sigma$, and any two strings $x, y \in \Sigma^*$ with $|x| = |y| \le n$, the following holds:*
      *(a) whenever $|x\sigma| \le n$, $(x\sigma, n) \equiv_S (y\sigma, n)$ iff $(x, n) \equiv_S (y, n)$, and*
      *(b) if $(x, n) \equiv_S (y, n)$, then $S(xz) = S(yz)$ for all strings $z$ with $|xz| = n$.*

Condition $(a)$ in this theorem concerns the reversibility of automata. Hereafter, we want to give the proof of Theorem 1.

**Proof of Theorem 1.** (1 $\Rightarrow$ 2) Assuming $S \in$ 1RFA/$n$, we take a 1rfa $M = (Q, \Sigma, \delta, q_0, Q_{acc}, Q_{rej})$ and an advice function $h$ such that $M([\,{}^{x}_{h(|x|)}]) = S(x)$ for all $x \in \Sigma^*$. Now, we set: $(x, n) \equiv_S (y, m)$ iff there exists a state $q \in Q$ for which $M$ enters $q$ just after reading $[\,{}^{x}_{w}]$ as well as $[\,{}^{y}_{w'}]$, where $w = Pref_{|x|}(h(n))$ and $w' = Pref_{|y|}(h(m))$. It is easy to check that $\equiv_S$ satisfies Conditions (i)-(ii).

(2 $\Rightarrow$ 1) To make our proof simple, we ignore the empty string and consider only the set $S \cap \Sigma^+$. Assume that we have an equivalence relation $\equiv_S$ that satisfies Conditions (i)-(ii). We want to show that $S \in$ 1RFA/$n$. Let $d = |\Delta/\equiv_S|$ for a certain constant $d > 0$ and assume that $\Delta/\equiv_S = \{A_1, A_2, \ldots, A_d\}$. Note that any two sets in $\Delta/\equiv_S$ are disjoint. The following four properties hold.

**Claim 1.**

1. *For any $x$ ($|x| \leq n$), there is a unique $q \in [d]$ such that $(x, n) \in A_q$.*
2. *For any $x, y, \sigma, q, n$ with $\sigma \in \Sigma$ and $|x| = |y| < n$, if $(x, n), (y, n) \in A_q$ then there exists an index $q' \in [d]$ such that $(x\sigma, n), (y\sigma, n) \in A_{q'}$.*
3. *For each fixed length $n \geq 1$, there exist two disjoint sets of indices in $[d]$, say, $C^{(n)}_{acc} = \{q^{(n)}_{acc,j}\}_j$ and $C^{(n)}_{rej} = \{q^{(n)}_{rej,j}\}_j$ such that $\{(x, n) \mid |x| = n, x \in S\} \subseteq \bigcup_j A_{q^{(n)}_{acc,j}}$ and $\{(x, n) \mid |x| = n, x \notin S\} \subseteq \bigcup_j A_{q^{(n)}_{rej,j}}$.*
4. *For every $x, y, \sigma, q'$ satisfying $(x\sigma, n), (y\sigma, n) \in A_{q'}$, there exists a unique index $q \in [d]$ for which $(x, n), (y, n) \in A_q$.*

It is tedious but straightforward to prove this claim; thus, we omit it entirely. Next, for any $n \in \mathbb{N}^+$ and $i \in [n]$, we define finite functions $h_{n,i} : [d] \times \Sigma \to [d] \cup \{\#\} \cup ([d] \times \{C_{acc}\} \times \{C_{rej}\})$ as follows. Let $q, q' \in [d]$ and $\sigma \in \Sigma$.

(i) Let $h_{n,1}(q, \sigma) = q'$ iff $(\sigma, n) \in A_{q'}$.
(ii) For any $i \in [2, n-1]_{\mathbb{Z}}$, $h_{n,i}(q, \sigma) = q'$ iff there exists a string $x$ with $|x| = i - 1$ such that $(x, n) \in A_q$ and $(x\sigma, n) \in A_{q'}$.
(iii) Let $h_{n,n}(q, \sigma) = (q', C^{(n)}_{acc}, C^{(n)}_{rej})$ iff there exists a string $x$ with $|x| = n - 1$ such that $(x, n) \in A_q$ and $(x\sigma, n) \in A_{q'}$.
(iv) In the above definitions, to make $h_{n,i}$ a total function, when $(x, n) \notin A_q$ for any string $x$ with $|x| = i - 1$, we need to define $h_{n,i}(q, \sigma) = \#$.

We treat each function $h_{n,i}$ as a new "symbol" and set $\Gamma = \{h_{n,i} \mid n \geq 1, i \in [n]\}$ so that $\Gamma$ becomes a finite set. Our advice string $h_n$ of length $n$ is defined to be $h_{n,1}h_{n,2}\cdots h_{n,n}$. By Claim 1(2), it is not difficult to show that $h_{n,i}$ is indeed a function.

We then define a finite automaton $M$ with its transition function $\delta$ as follows. Let $\delta(q_0, ¢) = q_0$ and $\delta(q, \$) = q$. For any $n$ and any $i \in [n-1]$, let $\delta(q, [\begin{smallmatrix}\sigma \\ h_{n,i}\end{smallmatrix}]) = h_{n,i}(q, \sigma)$. Assuming that $h_{n,n}(q, \sigma) = (q', C^{(n)}_{acc}, C^{(n)}_{rej})$, let $\delta(q, [\begin{smallmatrix}\sigma \\ h_{n,n}\end{smallmatrix}]) = (acc, q')$ (resp., $= (rej, q')$) if $q' \in C^{(n)}_{acc}$ (resp., $q' \in C^{(n)}_{rej}$). For any other remaining cases (if any) with $q \neq \#$, let $\delta(q, [\begin{smallmatrix}\sigma \\ \tau\end{smallmatrix}]) = \#$ and $\delta(\#, [\begin{smallmatrix}\sigma \\ \tau\end{smallmatrix}]) = q_0$. Note that, for any input string $x \in \Sigma^n$, with the correct advice function $h$, the automaton $M$ never reaches any other remaining cases; hence, we may define the values $\delta(q, [\begin{smallmatrix}\sigma \\ \tau\end{smallmatrix}])$ arbitrarily so that $\delta$ is reversible. Using Claim 1(1)&(4), it is possible to prove by induction on $i$ (for $h_{n,i}$) that $M$ is reversible.

Finally, we want to show that $S = \{x \mid M \text{ accepts } [\begin{smallmatrix}x \\ h_n\end{smallmatrix}]\}$. Assume that $x = \sigma_1\sigma_2\cdots\sigma_n$ and $(\lambda, n) \in A_{q_0}$, $(\sigma_1, n) \in A_{q_1}$, $(\sigma_1\sigma_2, n) \in A_{q_2}$, ..., $(x, n) \in A_{q_n}$. First, we consider the case where $x \in S$. We can prove by induction that $q_i = \hat{\delta}(q_0, [\begin{smallmatrix}\sigma_1\cdots\sigma_i \\ h_{n,1}\cdots h_{n,i}\end{smallmatrix}])$ for every $i \in [0, n]_{\mathbb{Z}}$, where $\hat{\delta}$ is the *extended transition function* induced from $\delta$. From the induction hypothesis on $i$, it immediately follows that $\hat{\delta}(q_0, [\begin{smallmatrix}\sigma_1\cdots\sigma_{i+1} \\ h_{n,1}\cdots h_{n,i+1}\end{smallmatrix}]) = h_{n,i+1}(\hat{\delta}(q_0, [\begin{smallmatrix}\sigma_1\cdots\sigma_i \\ h_{n,1}\cdots h_{n,i}\end{smallmatrix}]), \sigma_{i+1}) = h_{n,i+1}(q_i, \sigma_{i+1}) = q_{i+1}$. Since $x \in S$, by Claim 1(3), we obtain $q_n = (acc, q^{(n)}_{acc,j})$ for a certain $j$. Thus, it follows that $\hat{\delta}(q_0, [\begin{smallmatrix}x \\ h_n\end{smallmatrix}]) = (acc, q^{(n)}_{acc,j})$. Since $\hat{\delta}(q_0, [\begin{smallmatrix}x \\ h_n\end{smallmatrix}]\$) = q_{acc,j}$, $M$ accepts $[\begin{smallmatrix}x \\ h_n\end{smallmatrix}]$. Next, we assume that $x \notin S$. This case is similar to the previous case, since the only difference is the final step. This completes the proof of Theorem 1. □

For comparison, let us introduce another advised family 1RFA/$Rn$ that utilizes *random advice*, where randomized advice[1] is a probabilistic extension of deterministic advice. More precisely, 1RFA/$Rn$ denotes the family of all languages $L$ satisfying the following condition: there exist a 1rfa $M$, an error bound $\varepsilon \in [0, 1/2)$, and a probability ensemble $\{D_n\}_{n\in\mathbb{N}}$ such that, for every string $x \in \Sigma^*$, $\mathrm{Prob}_{D_{|x|}}[M([\,{}^{x}_{D_{|x|}}\,]) = L(x)] \geq 1 - \varepsilon$, where the probability is taken according to the distribution $D_{|x|}$. When 1dfa's are used in place of 1rfa's, we also obtain the family REG/$Rn$, and it was proven in [9] that REG/$Rn \nsubseteq$ CFL/$n$. In what follows, we show a class separation between REG/$n$ and DCFL$\cap$1RFA/$Rn$.

**Proposition 2.** *(1)* DCFL $\cap$ 1RFA/$Rn \nsubseteq$ REG/$n$*. (2)* 1RFA/$Rn \nsubseteq$ CFL/$n$*.*

**Proof.** (1) In this proof, we will sharpen the proof of [9, Proposition 16]. For our purpose, we use a "marked" version of $Pal$, the set of even-length *palindromes.* Consider the deterministic context-free language $Pal_{\#} = \{w\#w^R \mid w \in \{0, 1\}^*\}$ over the ternary alphabet $\Sigma = \{0, 1, \#\}$. Similar to the separation $Pal \notin$ REG/$n$ [7], we can prove that $Pal_{\#} \notin$ REG/$n$ by employing a so-called *swapping lemma* given in [7].

Next, we will show that $Pal_{\#} \in$ 1RFA/$Rn$. Assuming that its input tape has no endmarkers, we first define a *one-tape probabilistic finite automaton* (or a *1pfa*) $M$ with $Q = \{q_0, q_1, q_2, q_3\}$ and $Q_{acc} = \{q_0, q_2\}$. Let $\Gamma = \{0, 1, \#\}$. If $n = 2m$, we define $D_n$ to generate $y\#y^R$ with probability $2^{-m}$. Otherwise, we let $D_n$ generate $\#^n$ with probability 1. The transition function $\delta$ of $M$ is defined as follows. For every $\sigma, \tau \in \{0, 1\}$ and any index $i \in \{0, 1\}$, $\delta(q_i, [\,{}^{\sigma}_{\tau}\,]) = q_{\sigma\tau+i \bmod 2}$ and $\delta(q_i, a) = q_{i+1 \bmod 2}$, where $a = [\,{}^{\#}_{\#}\,]$. For any other state-symbol pair $(q, \sigma)$, we make two new transitions from $(q, \sigma)$ to both $q_2$ and $q_3$ with probability exactly 1/2. It is not quite difficult to translate $M$ into a reversible automaton.

On input of the form $x\#x'$, if $x' = x^R$, then we enter an accepting state with probability 1 for each fixed advice string $y$ produced by $D_n$. On the contrary, if $x' \neq x^R$, then over all possible $y$'s, we enter an accepting state with probability exactly 1/2. To reduce its error probability to 1/4, we need to make two runs of the above procedure in parallel.

(2) Similarly, we can show that $Dup = \{ww \mid w \in \{0, 1\}^*\}$ over the binary alphabet $\{0, 1\}$ is in 1RFA/$Rn$. It is known that $Dup \notin$ CFL/$n$ [7]. □

As an immediate consequence of Proposition 2, we obtain:

**Corollary 3.** 1RFA/$n \neq$ 1RFA/$Rn$*.*

## 4 Properties of Advice for Quantum Computation

As the second topic, we will examine *one-way measure-many quantum finite automata* (or 1qfa's, in short) that takes deterministic advice with bounded-error probability, where every 1qfa permits only one-way head moves and performs a (projection) measurement, at each step, to see if the machine enters any halting state. Formally, a 1qfa $M$ is defined as a sextuple $(Q, \Sigma, \{U_\sigma\}_{\sigma\in\check{\Sigma}}, q_0, Q_{acc}, Q_{rej})$,

[1] Randomized advice in the context of 1dfa's was discussed in [9].

where each *time-evolution operator* $U_\sigma$ is a unitary operator acting on the Hilbert space $E_Q = span\{|q\rangle \mid q \in Q\}$ of dimension $|Q|$. The series $\{U_\sigma\}_{\sigma\in\check{\Sigma}}$ describe the *time evolution* of $M$. Let $P_{acc}$, $P_{rej}$, and $P_{non}$ be respectively the projections of $E_Q$ onto the subspaces $E_{acc} = span\{|q\rangle \mid q \in Q_{acc}\}$, $E_{rej} = span\{|q\rangle \mid q \in Q_{rej}\}$, and $E_{non} = span\{|q\rangle \mid q \in Q_{non}\}$. For any symbol $\sigma \in \check{\Sigma}$, we define a *transition operator* $T_\sigma$ as $T_\sigma = P_{non}U_\sigma$. For each fixed string $x = \sigma_1\sigma_2\cdots\sigma_n$ in $\check{\Sigma}^*$, we set $T_x = T_{\sigma_n}T_{\sigma_{n-1}}\cdots T_{\sigma_2}T_{\sigma_1}$.

To describe precisely the *time-evolution* of $M$, let us consider a new Hilbert space $\mathcal{S}$ spanned by the basis vectors in $E_Q \times \mathbb{R} \times \mathbb{R}$. We then define a *norm*[2] of an element $\psi = (|\phi\rangle, \gamma_1, \gamma_2)$ in $\mathcal{S}$ to be $||\psi|| = (|||\phi\rangle||^2 + |\gamma_1| + |\gamma_2|)^{1/2}$. With the space $\mathcal{S}$, we extend the aforementioned transition operator $T_\sigma$ to $\hat{T}_\sigma$ by defining $\hat{T}_\sigma(|\phi\rangle, \gamma_1, \gamma_2) = (T_\sigma|\phi\rangle, \gamma_1 + ||P_{acc}U_\sigma|\phi\rangle||^2, \gamma_2 + ||P_{rej}U_\sigma|\phi\rangle||^2)$. For an arbitrary string $x = \sigma_1\sigma_2\cdots\sigma_n$ in $\check{\Sigma}^*$, we further define $\hat{T}_x$ as $\hat{T}_{\sigma_n}\hat{T}_{\sigma_{n-1}}\cdots\hat{T}_{\sigma_1}$. Notice that this extended operator $\hat{T}_x$ may not be a linear operator in general; however, it satisfies the following useful properties, which will play a key role in the proof of Theorem 6.

**Lemma 4.** [key lemma] *Each of the following statements holds.*

1. *For any two quantum states $|\phi\rangle, |\phi'\rangle \in E_{non}$ and any string $x \in \check{\Sigma}^*$, $|||\phi\rangle - |\phi'\rangle||^2 - ||T_x(|\phi\rangle - |\phi'\rangle)||^2 \leq \frac{3}{2}[(|||\phi\rangle||^2 - ||T_x|\phi\rangle||^2) + (|||\phi'\rangle||^2 - ||T_x|\phi'\rangle||^2)]$.*
2. *For any two elements $\psi, \psi' \in \mathcal{S}$, it holds that $||\psi + \psi'|| \leq ||\psi|| + ||\psi'||$.*
3. *For any two elements $\psi, \psi' \in \mathcal{S}$ and any string $x \in \check{\Sigma}^*$, $||\hat{T}_x\psi - \hat{T}_x\psi'|| \leq \sqrt{2}||\psi - \psi'||$.*
4. *For any two elements $\psi, \psi' \in \mathcal{S}$ and any string $x \in \check{\Sigma}^*$, let $\psi = (|\phi\rangle, \gamma_1, \gamma_2)$ and $\psi' = (|\phi'\rangle, \gamma_1', \gamma_2')$. Then, $||\hat{T}_x\psi - \hat{T}_x\psi'||^2 \geq ||\psi - \psi'||^2 - 3(|||\phi\rangle - |\phi'\rangle||^2 - ||T_x(|\phi\rangle - |\phi'\rangle)||^2)$.*

**Proof Sketch.** Here, we give only an outline of the proof of (1). Assume that $x = x_1x_2\cdots x_n$ and fix $i \in [1,n]_{\mathbb{Z}}$ arbitrarily. Moreover, let $|\phi_i\rangle = T_{x_1x_2\cdots x_{i-1}}|\phi\rangle$ and $|\phi_i'\rangle = T_{x_1x_2\cdots x_{i-1}}|\phi'\rangle$. It then follows that

$$\begin{aligned}|||\phi\rangle - |\phi'\rangle||^2 - ||T_x(|\phi\rangle - |\phi'\rangle)||^2 \;=\; & (|||\phi\rangle||^2 - ||T_x|\phi\rangle||^2) \\ & + (|||\phi'\rangle||^2 - ||T_x|\phi'\rangle||^2) + (\langle\phi|T_x^\dagger T_x|\phi'\rangle + \langle\phi'|T_x^\dagger T_x|\phi\rangle - \langle\phi|\phi'\rangle - \langle\phi'|\phi\rangle) \\ \leq\; & (|||\phi\rangle||^2 - ||T_x|\phi\rangle||^2) + (|||\phi'\rangle||^2 - ||T_x|\phi'\rangle||^2) \\ & + |\langle\phi|T_x^\dagger T_x|\phi'\rangle - \langle\phi|\phi'\rangle| + |\langle\phi'|T_x^\dagger T_x|\phi\rangle - \langle\phi'|\phi\rangle|.\end{aligned}$$

A vigorous calculation leads to the following inequalities:

$$\begin{aligned}|\langle\phi|\phi'\rangle - \langle\phi|T_x^\dagger T_x|\phi'\rangle| &\leq \frac{1}{2}\sum_{i=1}^{n}[(|||\phi_i\rangle||^2 - ||T_{x_i}|\phi_i\rangle||^2) + (|||\phi_i'\rangle||^2 - ||T_{x_i}|\phi_i'\rangle||^2)] \\ &\leq \frac{1}{2}[(|||\phi_1\rangle||^2 - |||\phi_{n+1}\rangle||^2) + (|||\phi_1'\rangle||^2 - |||\phi_{n+1}'\rangle||^2)].\end{aligned}$$

Since $|\phi_{n+1}\rangle = T_x|\phi\rangle$ and $|\phi_{n+1}'\rangle = T_x|\phi'\rangle$, the lemma follows. □

Recall that any input to a 1qfa must be of the form $¢x\$ = \sigma_1\sigma_2\cdots\sigma_{n+2}$, where $\sigma_1 = ¢$, $\sigma_{n+2} = \$$, and $x \in \Sigma^n$. The *acceptance probability* of $M$ on $x$ at step

[2] Our definition of "norm" is slightly different in its form from the norm in [4].

$i$ $(1 \leq i \leq n+2)$, denoted $p_{acc}(x,i)$, is $||P_{acc}U_{\sigma_i}|\phi_{i-1}\rangle||^2$, where $|\phi_0\rangle = |q_0\rangle$ and $|\phi_i\rangle = T_{\sigma_i}|\phi_{i-1}\rangle$. The *acceptance probability* of $M$ on $x$, denoted $p_{acc}(x)$, is $\sum_{i=1}^{n+2} p_{acc}(x,i)$. Similarly, we define the *rejection probabilities* $p_{rej}(x,i)$ and $p_{rej}(x)$ using $P_{rej}$ instead of $P_{acc}$. In the end of a computation, using those notations, we obtain $\hat{T}_{\cent x\$}(|q_0\rangle,0,0) = (|\phi_{n+2}\rangle, p_{acc}(x), p_{rej}(x))$. Finally, let 1QFA denote the collection of all languages recognized by 1qfa's with *bounded error probability* (i.e., the error probability is upper-bounded by an absolute constant in $[0,1/2)$).

Similar to 1RFA/$n$, the notation 1QFA/$n$ indicates the collection of all languages $L$ over alphabets $\Sigma$ that satisfy the following condition: there exist a 1qfa $M$, an error bound $\varepsilon \in [0,1/2)$, an advice alphabet $\Gamma$, and an advice function $h : \mathbb{N} \to \Gamma^*$ such that (i) $|h(n)| = n$ for each length $n \in \mathbb{N}$ and (ii) for every $x \in \Sigma^*$, $M$ on input $[\,{}^{x}_{h(|x|)}]$ outputs $L(x)$ with probability at least $1-\varepsilon$ (abbreviated as $\mathrm{Prob}_M[M([\,{}^{x}_{h(|x|)}]) = L(x)] \geq 1-\varepsilon$). Similar to a known inclusion $1\mathrm{QFA} \subseteq \mathrm{REG}$ [4], $1\mathrm{QFA}/n \subseteq \mathrm{REG}/n$ holds.

An immediate benefit of using advice for 1qfa's is the elimination of endmarkers on their input tapes. Earlier, Brodsky and Pippenger [2] demonstrated that we can eliminate the left endmarker ¢. The use of advice further enables us to eliminate the right endmarker \$ as well. This is done by marking the end of an input string by a piece of advice. We omit the proof of Lemma 5.

**Lemma 5.** [endmarker lemma] *For any language $L \in 1\mathrm{QFA}/n$, there exist a 1qfa $M$, a constant $\varepsilon \in [0,1/2)$, an advice alphabet $\Gamma$, and an advice function $h$ such that (i) $M$'s input tape has no endmarkers, (ii) $|h(n)| = n$ for any length $n \in \mathbb{N}$, and (iii) for any string $x \in \Sigma^*$, $\mathrm{Prob}_M[M([\,{}^{x}_{h(|x|)}]) = L(x)] \geq 1-\varepsilon$.*

Next, we will give a precise description of our main theorem. Following a standard convention, for any partial order $\leq$ defined on a finite set, we use the notation $x = y$ whenever $x \leq y$ and $y \leq x$; moreover, we write $x < y$ if $x \leq y$ and $x \neq y$. A finite sequence $(s_1, s_2, \ldots, s_m)$ is called a *strictly descending chain* of length $m$ (with respect to $\leq$) if $s_{i+1} < s_i$ for any index $i \in [m-1]$. For our convenience, we call a reflexive, symmetric, binary relation a *closeness relation*. For any closeness relation $\cong$, an $\cong$-*discrepancy set* is a set $S$ satisfying that, for any two elements $x, y \in S$, if $x$ and $y$ are "different," then $x \not\cong y$.

**Theorem 6.** *Let $S$ be any language over an alphabet $\Sigma$. Let $\Delta = \{(x,n) \in \Sigma^* \times \mathbb{N} \mid |x| \leq n\}$. If $S \in 1\mathrm{QFA}/n$, then there exist two constants $c, d > 0$, an equivalence relation $\equiv_S$ over $\Delta$, a partial order $\leq_S$ over $\Delta$, and a closeness relation $\cong$ over $\Delta$ that satisfy the following seven conditions. Let $(x,n),(y,n) \in \Delta$, $z \in \Sigma^*$, and $\sigma \in \Sigma$ with $|x| = |y|$.*

1. *The set $\Delta/\!\equiv_S$ is finite.*
2. *If $(x,n) \cong (y,n)$, then $(x,n) \equiv_S (y,n)$.*
3. *If $|x\sigma| \leq n$, then $(x\sigma,n) \leq_S (x,n)$.*
4. *If $|xz| \leq n$, $(x,n) =_S (xz,n)$, $(y,n) =_S (yz,n)$, and $(xz,n) \cong (yz,n)$, then $(x,n) \equiv_S (y,n)$.*
5. *$(x,n) \equiv_S (y,n)$ iff $S(xz) = S(yz)$ for all $z \in \Sigma^*$ with $|xz| = n$.*

6. *Any strictly descending chain (w.r.t. $\leq_S$) in $\Delta$ has length at most $c$.*
7. *Any $\cong$-discrepancy subset of $\Delta$ has cardinality at most $d$.*

The meanings of the above relations $\simeq$, $\leq_S$, and $\equiv_S$ will be soon explained in the proof of Theorem 6. Our proof of the theorem heavily depends on Lemma 4, and thus it requires only basic properties of the norm in Hilbert spaces.

**Proof of Theorem 6.** Let $S \in 1\mathrm{QFA}/n$ be any language over an alphabet $\Sigma$. For this language $S$, there are an alphabet $\Gamma$, an error bound $\varepsilon \in [0,1/2)$, a 1qfa $M$, and an advice function $h : \mathbb{N} \to \Gamma^*$ such that, for every string $x \in \Sigma^*$, $\mathrm{Prob}_M[M([\,{}^{x}_{h(|x|)}]) = S(x)] \geq 1-\varepsilon$. W.l.o.g, we assume that $\varepsilon > 0$.

Recall that $\Delta = \{(x,n) \mid x \in \Sigma^*, |x| \leq n\}$. For the advice alphabet $\Gamma$, let $\check{\Gamma} = \{[\,{}^{\sigma}_{\tau}] \mid \sigma \in \Sigma, \tau \in \Gamma\}$ denote its extended alphabet and set $e = |\check{\Gamma}|$. For simplicity, write $\psi_0$ for the triplet $(|q_0\rangle, 0, 0)$. For each element $(x,n) \in \Delta$, we assume that $\hat{T}_{¢[\,{}^{x}_{w}]}\psi_0 = (|\phi_x\rangle, \gamma_{x,1}, \gamma_{x,2})$, where $w = Pref_{|x|}(h(n))$.

To satisfy Condition 5, it suffices to define $(x,n) \equiv_S (y,n)$ whenever $S(xz) = S(yz)$ for all strings $z$ with $|xz| = n$. The following claim is trivial.

**Claim 2.** *The set $\Delta/\equiv_S$ is finite.*

Next, we define a closeness relation $\cong$ over $\Delta$. Choose a constant $\mu$ satisfying $0 < \mu < (1-2\varepsilon)/10$ and define $(x,n) \cong (y,m)$ if $||\hat{T}_{¢[\,{}^{x}_{w}]}\psi_0 - \hat{T}_{¢[\,{}^{y}_{w'}]}\psi_0||^2 < \mu$, where $w = Pref_{|x|}(h(n))$ and $w' = Pref_{|y|}(h(m))$. For any $\cong$-discrepancy subset $G$ of $\Delta$, it is obvious that $G$ is a finite set and thus its cardinality $|G|$ is upper-bounded by a certain absolute constant. Now, we define $d = \max_G\{|G|\}$. Hence, Condition 7 is also satisfied.

To show Condition 2, we first claim the following statement.

**Claim 3.** *If $||\hat{T}_{¢[\,{}^{x}_{w}]}\psi_0 - \hat{T}_{¢[\,{}^{y}_{w}]}\psi_0||^2 < 1-2\varepsilon$, then $(x,n) \equiv_S (y,n)$.*

Condition 2 follows from Claim 3, since $(x,n) \cong (y,n)$ implies $||\hat{T}_{¢[\,{}^{x}_{w}]}\psi_0 - \hat{T}_{¢[\,{}^{y}_{w}]}\psi_0||^2 < \mu < 1-2\varepsilon$. To prove Claim 3, it suffices to show Claims 4 and 5.

**Claim 4.** *For any two elements $(x,n),(y,n) \in \Delta$ and any string $z \in \Sigma^*$ with $|x| = |y|$ and $|xz| = n$, it holds that $2||\hat{T}_{¢[\,{}^{x}_{w}]}\psi_0 - \hat{T}_{¢[\,{}^{y}_{w}]}\psi_0||^2 \geq |p_{acc}(xz) - p_{acc}(yz)| + |p_{rej}(xz) - p_{rej}(yz)|$.*

**Claim 5.** *If $|x| = |y| \leq n$ and $||\hat{T}_{¢[\,{}^{x}_{w}]}\psi_0 - \hat{T}_{¢[\,{}^{y}_{w}]}\psi_0||^2 < 1-2\varepsilon$, then $S(xz) = S(yz)$ for any string $z$ with $|xz| = n$.*

**Proof.** Assume to the contrary that $||\hat{T}_{¢[\,{}^{x}_{w}]}\psi_0 - \hat{T}_{¢[\,{}^{y}_{w}]}\psi_0||^2 < 1-2\varepsilon$ but $S(xz) \neq S(yz)$ for a certain $z$ with $|xz| = n$. This implies that $|p_{acc}(xz) - p_{acc}(yz)| \geq 1-2\varepsilon$ and $|p_{rej}(xz) - p_{rej}(yz)| \geq 1-2\varepsilon$. Thus, by Claim 4, we have

$$2||\hat{T}_{¢[\,{}^{x}_{w}]}\psi_0 - \hat{T}_{¢[\,{}^{y}_{w}]}\psi_0||^2 \geq |p_{acc}(xz) - p_{acc}(yz)| + |p_{rej}(xz) - p_{rej}(yz)|,$$

which is further lower-bounded by $2(1-2\varepsilon)$. This contradicts our assumption that $||\hat{T}_{¢[\substack{x\\w}]}\psi_0 - \hat{T}_{¢[\substack{y\\w}]}\psi_0||^2 < 1-2\varepsilon$. Therefore, $S(xz) = S(yz)$ should hold. □

Next, we define a partial order $\leq_S$ on $\Delta$ as follows: $(x,n) \leq_S (y,m)$ iff there exist two numbers $s,s' \in \mathbb{N}$ such that (i) $0 \leq s \leq s' \leq \lceil 1/\mu \rceil$, (ii) $(s-1)\mu < |||\phi_x\rangle||^2 \leq s\mu$, and (iii) $(s'-1)\mu < |||\phi_y\rangle||^2 \leq s'\mu$. Since $|||\phi_{x\sigma}\rangle|| \leq |||\phi_x\rangle||$ for any $x$ and $\sigma$, Condition 3 immediately follows. Note that $(x,n) =_S (y,m)$ implies $||||\phi_y\rangle||^2 - |||\phi_x\rangle||^2| < \mu$.

The desired constant $c$ is set to be $\lceil 1/\mu \rceil + 1$. Since $|||\phi_\lambda\rangle|| = 1$ and $|||\phi_x\rangle|| \leq \varepsilon$ for all $x$ with $|x| = n$, the relation $<_S$ must appear at most $c$ times in any series of the form $\{(\lambda,n),(\sigma_1,n),\ldots,(\sigma_1\cdots\sigma_n,n)\}$. Condition 6 thus follows.

Condition 4 requires Claim 6, which follows from Lemmas 4(1)&(4).

**Claim 6.** *Assume that* $|x| = |y|$ *and* $|xz| \leq n$. *If* $||\hat{T}_{¢[\substack{xz\\wu}]}\psi_0 - \hat{T}_{¢[\substack{yz\\wu}]}\psi_0||^2 < \gamma$, $|||\phi_x\rangle||^2 - |||\phi_{xz}\rangle||^2 < \alpha$, *and* $|||\phi_y\rangle||^2 - |||\phi_{yz}\rangle||^2 < \alpha$, *then* $||\hat{T}_{¢[\substack{x\\w}]}\psi_0 - \hat{T}_{¢[\substack{y\\w}]}\psi_0||^2 < \gamma + 9\alpha$, *where* $wu = Pref_{|xz|}(h(n))$, $|w| = |x|$, *and* $|u| = |z|$.

To show Condition 4, assume that $(x,n) \cong (y,n)$, $(x\sigma,n) =_S (x,n)$, and $(yz,n) =_S (y,n)$. In other words, $||\hat{T}_{¢[\substack{x\\w}]}\psi_0 - \hat{T}_{¢[\substack{y\\w}]}\psi_0||^2 < \mu$, $|||\phi_{xz}\rangle||^2 - |||\phi_x\rangle||^2 < \mu$, and $|||\phi_{yz}\rangle||^2 - |||\phi_y\rangle||^2 < \mu$. Claim 6 implies that $||\hat{T}_{¢[\substack{x\\w}]}\psi_0 - \hat{T}_{¢[\substack{y\\w}]}\psi_0||^2 < 10\mu$. Since $10\mu < 1-2\varepsilon$, Claim 3 yields $(x,n) \equiv_S (y,n)$, as requested. This completes the proof of Theorem 6. □

We have so far proven two main theorems, Theorems 1–6, which give unique features of advised 1rfa's and 1qfa's. Hereafter, we will state two direct consequences of those features. As the first consequence, we will show below that 1QFA is not included in 1RFA/$n$. This result can be viewed as a strength of bounded-error quantum computation over error-free advised quantum computation.

**Corollary 7.** 1QFA $\nsubseteq$ 1RFA/$n$, *and thus* 1RFA/$n \neq$ 1QFA/$n$.

**Proof.** Consider the language $L = \{0^m1^n \mid m,n \in \mathbb{N}\}$. Ambainis and Freivalds [1] showed how to recognize this language $L$ on a certain 1qfa with success probability at least 0.68. It thus suffices to show that $L \notin$ 1RFA/$n$. Note that $L$ was shown to be outside of 1RFA [1]. Our result therefor extends this result.

Assume that $L \in$ 1RFA/$n$. By Theorem 1, there exists an equivalence relation $\equiv_L$ over $\Delta$ that satisfies the theorem's conditions. Let $k$ be the cardinality $|\Delta/\equiv_L|$. Fix any number $n$ satisfying that $n > k+1$. Let us consider the set $S = \{0^i1^{n-i} \mid i \in [1,n-1]_{\mathbb{Z}}\}$. Since $|S| = n-1 > k$, there are two indices $i,j \in [1,n-1]_{\mathbb{Z}}$ with $i < j$ such that $(0^i1^{n-i},n) \equiv_L (0^j1^{n-j},n)$. By Condition $(a)$ of the theorem, we have $(0^i1^{j-i},n) \equiv_L (0^j,n)$ since $i < j$. By Condition $(b)$, if we choose $z = 0^{n-j}$, then $L(0^i1^{j-i}z) = L(0^jz)$. Recall that $i < j < n$. However, since $L(0^i1^{j-i}z) = L(0^i1^{j-i}0^{n-j}) = 0$ and $L(0^jz) = L(0^n) = 1$, we obtain a contradiction. Therefore, $L$ cannot belong to 1RFA/$n$. □

Next, we seek the second consequence. Concerning the language family 1QFA, Kondacs and Watrous [4] first proved that REG $\nsubseteq$ 1QFA. The following class separation between REG and 1QFA$/n$ indicates that 1qfa's are still not as powerful as 1dfa's even with a great help of advice.

**Corollary 8.** REG $\nsubseteq$ 1QFA$/n$, *and thus* 1QFA$/n \neq$ REG$/n$.

**Proof.** Our example language $S$ over the binary alphabet $\Sigma = \{a, b\}$ can be expressed using a regular expression as $(aa + ab + ba)^*$.

First, it is not difficult to show that $S$ is regular. Next, we intend to show that $S$ is not in 1QFA$/n$. Assume otherwise; that is, $S \in$ 1QFA$/n$. Let $\Delta = \{(x, n) \in \Sigma^* \times \mathbb{N} \mid |x| \leq n\}$. By Theorem 6, there exist two constants $c, d > 0$, an equivalence relation $\equiv_S$, a partial order $\leq_S$, and a closeness relation $\cong$ that satisfy Conditions 1-7 given in the theorem. Let $e$ be the total number of equivalence classes in $\Delta/\equiv_S$ and let $k = \max\{c, d, e\}$. Let $n$ denote the minimal *even* integer such that $n \geq (k + 1)(k + 5)$.

To draw a contradiction, we want to construct a special string $x$ of length at most $n$. Inductively, we build a series of strings $x_1, x_2, \ldots, x_m$ such that each $x_i$ has length at most $2(\lceil \log k \rceil + 1)$ and the total length $|x_1 \cdots x_m|$ should be maximized without exceeding $n$. For our convenience, set $x_0 = \lambda$. Assuming that $x_0, x_1, x_2, \ldots, x_i$ are already defined, we want to define $x_{i+1}$ as follows. Let us denote by $x'_i$ the concatenated string $x_1 x_2 \cdots x_i$ and denote by $z_{i,w}$ the string $x'_i w$ for each string $w \in ((a + b)a)^*$ with $|z_{i,w}| \leq n$. Our key claim is stated as follows.

**Claim 7.** *There exists a nonempty string* $w \in ((a + b)a)^*$ *such that* $|w| \leq 2(\lceil \log k \rceil + 1)$ *and* $(z_{i,w}, n) <_S (x'_i, n)$.

Now, we choose the (lexicographically) first nonempty string $w \in ((a + b)a)^*$ that satisfies $|w| \leq 2(\lceil \log k \rceil + 1)$ and $(z_{i,w}, n) <_S (x'_i, n)$. The desired string $x_{i+1}$ is defined to be this string $w$. Obviously, it holds that $|x_{i+1}| \leq 2(\lceil \log k \rceil + 1)$. The construction of $x_i$'s implies that $(x'_m, n) <_S (x'_{m-1}, n) <_S \cdots <_S (x'_1, n)$. From Condition 6, we have $m \leq c \leq k$. Hence, we conclude that $|x_1 x_2 \cdots x_m| > n - 2(\lceil \log k \rceil + 1)$ because, otherwise, we can define $x_{m+1}$ to satisfy that $|x_{m+1}| \leq 2(\lceil \log k \rceil + 1)$ and $(x'_{m+1}, n) <_S (x'_m, n)$, contradicting the maximality of $|x_1 x_2 \cdots x_m|$. Thus, we obtain $n < (k + 1)(k + 5)$. However, since $n \geq (k + 1)(k + 5)$, we obtain a contradiction. Therefore, $S \notin$ 1QFA$/n$.

To complete the proof of the corollary, it still remains to prove Claim 7. This claim can be proven by a way of contradiction with a careful use of Conditions 4, 5, and 7 of Theorem 6. Let us assume that $x'_i$ is already defined. Toward a contradiction, we now suppose that the claim fails; that is, for any nonempty string $w \in ((a + b)a)^*$ with $|w| \leq 2(\lceil \log k \rceil + 1)$, we have $(z_{i,w}, n) =_S (x'_i, n)$. Under this assumption, the following statement holds.

**Claim 8.** *For any two distinct pair* $w, w'$ *in* $(aa + ab + ba)^*$ *with* $|w| = |w'| \leq n - 2$, *it holds that* $(wa, n) \not\equiv_S (w'b, n)$.

Assuming that Claim 8 holds, let us consider all strings in $((a+b)a)^{\lceil \log k \rceil + 1}$. Note that the total number of such strings is $2^{\lceil \log k \rceil + 1} \geq 2k$. We define $G_n$ to be the set of all elements $(z_{i,w}, n) \in \Delta_n$ for any $w \in ((a+b)a)^{\lceil \log k \rceil + 1}$. Now, we want to show that $G_n$ is a $\cong$-discrepancy set. Assume otherwise; that is, $(z_{i,w}, n) \cong (z_{i,w'}, n)$ for certain two *distinct* strings $w, w' \in ((a+b)a)^{\lceil \log k \rceil + 1}$. For such strings, there are (possibly empty) strings $y, y', z$ for which $w = yaaz$ and $w' = y'baz$. By applying Claim 8 to the two strings $x'_i y$ and $x'_i y'$, we conclude that $(x'_i ya, n) \not\equiv_S (x'_i y'b, n)$. Since $(z_{i,w}, n) =_S (x'_i, n) =_S (z_{i,w'}, n)$ by our assumption, Condition 4 of Theorem 6 implies that $(x'_i ya, n) \equiv_S (x'_i y'b, n)$, a contradiction. This implies that $G_n$ is indeed a $\cong$-discrepancy subset of $\Delta$. By the definition of $G_n$, it follows that $|G_n| \geq 2k$. However, this contradicts Condition 7 of Theorem 6, which says $|G_n| \leq d \leq k$. Therefore, Claim 7 should hold.

At the end, let us prove Claim 8 by induction on the length $|w|$. Consider the case where $|w| = 0$. Assume that $(a, n) \equiv_S (b, n)$. By the definition of $S$, there is a string $z$ such that $|az| = n$ and $S(az) \neq S(bz)$. For instance, when $n = 2$, we have $S(ab) \neq S(bb)$. However, Condition 5 implies that $S(az) = S(bz)$, leading to a contradiction. Thus, it follows that $(a, n) \not\equiv_S (b, n)$. Next, consider the case where $0 < |w| \leq n - 2$. Since $w, w' \in (aa + ab + ba)^*$, there exists a string $z$ such that $|wabz| = n$ and $S(wabz) \neq S(w'bbz)$. Hence, by a similar reasoning to the initial case, we obtain that $(wa, n) \not\equiv_S (w'b, n)$. □

## References

1. Ambainis, A., Freivalds, R.: 1-way quantum finite automata: strengths, weaknesses, and generalizations. In: FOCS 1998, pp. 332–342 (1998)
2. Brodsky, A., Pippenger, N.: Characterizations of 1-way quantum finite automata. SIAM J. Comput. 31, 1456–1478 (2002)
3. Karp, R., Lipton, R.: Turing machines that take advice. L'Enseignement Mathématique 28, 191–209 (1982)
4. Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In: FOCS 1997, pp. 66–75 (1997)
5. Nishimura, H., Yamakami, T.: Polynomial-time quantum computation with advice. Inf. Process. Lett. 90, 195–204 (2004)
6. Tadaki, K., Yamakami, T., Lin, J.: Theory of one tape linear time Turing machines. Theor. Comput. Sci. 411, 22–43 (2010)
7. Yamakami, T.: Swapping lemmas for regular and context-free languages (2008), arXiv:0808.4122
8. Yamakami, T.: Immunity and pseudorandomness of context-free languages. Theor. Comput. Sci. 412, 6432–6450 (2011)
9. Yamakami, T.: The roles of advice to one-tape linear-time Turing machines and finite automata. Int. J. Found. Comput. Sci. 21, 941–962 (2011)
10. Yamasaki, T., Kobayashi, H., Imai, H.: Quantum versus deterministic counter automata. Theor. Comput. Sci. 334, 275–297 (2005)

# Integration of the Dual Approaches in the Distributional Learning of Context-Free Grammars

Ryo Yoshinaka[⋆]

Kyoto University, Japan
ry@i.kyoto-u.ac.jp

**Abstract.** Recently several "distributional learning algorithms" have been proposed and have made great success in learning different subclasses of context-free grammars. The distributional learning models and exploits the relation between strings and contexts that form grammatical sentences in the language of the learning target. There are two main approaches. One, which we call *primal*, constructs nonterminals whose language is supposed to be characterized by strings. The other, which we call *dual*, uses contexts to characterize the language of each nonterminal of the conjecture grammar. This paper shows how those opposite approaches are integrated into single learning algorithms that learn quite rich classes of context-free grammars.

## 1 Introduction

Recent studies on grammatical inference have demonstrated how powerful the idea of "distributional learning" is for learning context-free grammars (CFGs). Distributional learning algorithms exploit information on combinations of strings and contexts that form grammatical sentences in the concerned language $L$. Clark [5] has classified existing distributional learning algorithms into two approaches, which are called *primal* and *dual*, respectively, and Yoshinaka [11] has discussed the neat symmetry of those two. For example, Shirakawa and Yokomori's learning algorithm for *c-deterministic* CFG*s* [9] and Clark's one for learning CFGs with the $q$-FCP [4] take the dual approach. As their respective primal counterparts, one can understand Clark's learning algorithm for *congruential* CFG*s* [3] and Yoshinaka's for CFGs with the $p$-FKP [11]. The primal approach assumes that the learning target is generated by a CFG whose nonterminal symbols generate a language "characterizable" by a finite string set. A learner uses sets of strings obtained from given data as nonterminal symbols of its hypothesis grammar, whereas sets of contexts are used to reject ineligible rules constructed from those nonterminals. In the dual approach, the roles played by string sets and context sets are switched. An important and interesting property of the hypothesis grammar is the monotonic relation between the generated language and

[⋆] This work was supported in part by MEXT KAKENHI (B-23700156).

respective sets of strings and contexts. In the primal approach, the bigger the string set gets, the bigger the conjectured language becomes, while the bigger the context set becomes, the smaller the language will be. The dual approach shows just the opposite monotonicity. Based on this monotonicity, a learner is able to adjust the hypothesis language and finally achieves a right grammar for the target language.

Although those two approaches might appear incompatible due to the opposite monotonicity, actually it is possible to integrate those two into a single algorithm. This paper presents algorithms that take advantage of both primal and dual approaches, which enables us to learn richer classes of languages more efficiently than taking solely either primal or dual approach. One of our algorithms can be seen as an integration of Shirakawa and Yokomori's algorithm for c-deterministic CFGs [9] and Clark's for congruential CFGs [3]. The other is an integration of Yoshinaka's algorithms for CFGs with the $p$-FKP and with the $q$-FCP [11], respectively, where the latter is a simplification of Clark's [4].

## 2 Preliminaries

Let $\Sigma$ be a nonempty finite set of letters. We denote the empty string by $\lambda$. Any element of $\Sigma^* \times \Sigma^*$ is called a *context*. The empty context $\langle \lambda, \lambda \rangle$ is denoted by $\Lambda$. For a string $v \in \Sigma^*$ and a context $\langle u_1, u_2 \rangle \in \Sigma^* \times \Sigma^*$, the *composition* of them is $\langle u_1, u_2 \rangle \odot v = u_1 v u_2 \in \Sigma^*$. The composition operation is naturally generalized to be applied to sets $W \subseteq \Sigma^* \times \Sigma^*$ and $V \subseteq \Sigma^*$ as $W \odot V = \{\, w \odot v \mid w \in W,\ v \in V \,\}$. For a language $L \subseteq \Sigma^*$, we let

$$\begin{aligned}\mathrm{Sub}(L) &= \{\, v \in \Sigma^* \mid w \odot v \in L \text{ for some } w \in \Sigma^* \times \Sigma^* \,\},\\ \mathrm{Con}(L) &= \{\, w \in \Sigma^* \times \Sigma^* \mid w \odot v \in L \text{ for some } v \in \Sigma^* \,\}.\end{aligned}$$

We also have an operation dual to $\odot$. We denote the set of contexts that admit every string in a set $V \subseteq \Sigma^*$ with respect to a language $L \subseteq \Sigma^*$ by

$$L \oslash V = \{\, w \in \Sigma^* \times \Sigma^* \mid w \odot v \in L \text{ for all } v \in V \,\}.$$

Similarly, the set of strings that every context in $W \subseteq \Sigma^* \times \Sigma^*$ accepts is

$$L \oslash W = \{\, v \in \Sigma^* \mid w \odot v \in L \text{ for all } w \in W \,\}.$$

Note that $L \oslash \{\Lambda\} = L$. By definition, $W \odot V \subseteq L$ iff $W \subseteq L \oslash V$ iff $V \subseteq L \oslash W$. When $L$ is understood, particularly when our learning target is $L$, we denote $L \oslash W$ for $W \subseteq \Sigma^* \times \Sigma^*$ by $W^\dagger$ and $L \oslash V$ for $V \subseteq \Sigma^*$ by $V^\ddagger$. It is easy to see that $V \subseteq (V^\ddagger)^\dagger$, $W \subseteq (W^\dagger)^\ddagger$, $V^\ddagger = ((V^\ddagger)^\dagger)^\ddagger$ and $W^\dagger = ((W^\dagger)^\ddagger)^\dagger$. Moreover, we define $W^\ddagger = (W^\dagger)^\ddagger$ and $V^\dagger = (V^\ddagger)^\dagger$. It is clear by definition that $W \odot V \subseteq L$ iff $W^\ddagger \odot V^\dagger \subseteq L$. In both cases where $X \subseteq \Sigma^*$ and where $X \subseteq \Sigma^* \times \Sigma^*$, we have $X^\dagger \subseteq \Sigma^*$ and $X^\ddagger \subseteq \Sigma^* \times \Sigma^*$. For $X_1, X_2 \in 2^{\Sigma^*} \cup 2^{\Sigma^* \times \Sigma^*}$, we write $X_1 \preccurlyeq_L X_2$

iff $X_1^\dagger \subseteq X_2^\dagger$, which is equivalent to $X_1^\ddagger \supseteq X_2^\ddagger$. If $X_1 \preccurlyeq_L X_2$ and $X_2 \preccurlyeq_L X_1$, we write $X_1 \approx_L X_2$.[1] The subscript $L$ of $\approx_L$ is dropped when understood.

**Lemma 1.** *For $X_1, X_2 \in 2^{\Sigma^*} \cup 2^{\Sigma^* \times \Sigma^*}$, $X_1 \preccurlyeq X_2$ iff $X_2^\ddagger \odot X_1^\dagger \subseteq L$.*

*We have $w \odot V_1 \subseteq L \Leftrightarrow w \odot V_2 \subseteq L$ for every $w \in \Sigma^* \times \Sigma^*$ if and only if $V_1 \approx V_2$.*

*We have $W_1 \odot v \subseteq L \Leftrightarrow W_2 \odot v \subseteq L$ for every $v \in \Sigma^*$ if and only if $W_1 \approx W_2$.*

A CFG is a tuple $G = \langle \Sigma, N, P, S \rangle$, where $\Sigma$ is the set of terminal symbols, $N$ is the set of nonterminal symbols, $P \subseteq N \times (N \cup \Sigma)^*$ is the set of rules, and $S \in N$ is the start symbol. The derivation relation of $G$ is denoted by $\Rightarrow_G^*$. The language generated by each nonterminal symbol $A$ is denoted by $\mathcal{L}(G, A) = \{\, v \in \Sigma^* \mid A \Rightarrow_G^* v \,\}$ and the language of $G$ is $\mathcal{L}(G) = \mathcal{L}(G, S)$. The context set of a nonterminal $A$ is defined by $\mathcal{C}(G, A) = \{\, \langle u_1, u_2 \rangle \in \Sigma^* \times \Sigma^* \mid S \Rightarrow_G^* u_1 A u_2 \,\}$. This paper assumes that $\mathcal{L}(G, A) \neq \varnothing$ and $\mathcal{C}(G, A) \neq \varnothing$ for all $A \in N$. The description size of $G$ is defined to be $\|G\| = \sum_{A \to \alpha \in P} |A\alpha|$.

## 3 Learning Targets

Clark [3] has introduced a class of grammars, called *congruential* CFG*s* and presented a learning algorithm for them. We say that a CFG is congruential if for every nonterminal $A$ of $G$ and every $v_1, v_2 \in \mathcal{L}(G, A)$, we have $\{v_1\} \approx \{v_2\}$.

Quite a symmetric result has been obtained by Shirakawa and Yokomori [9]. They have presented a learning algorithm for *c-deterministic* CFG*s* in Chomsky normal form (CNF), where we say that a CFG is c-deterministic if for any context $w \in \mathcal{C}(G, A)$ of any nonterminal $A$, it holds that $\mathcal{L}(G) \oslash \{w\} = \mathcal{L}(G, A)$.

This paper targets a class of CFGs that is a superclass of Clark's and Shirakawa and Yokomori's learning targets.

**Definition 1.** By $\mathbb{G}(r)$ we denote the class of CFGs $G$ such that each rule has at most $r$ nonterminals on the right-hand side and each nonterminal $A$ satisfies either

- $\{v\} \approx \mathcal{L}(G, A)$ for any $v \in \mathcal{L}(G, A)$, or
- $\{w\} \approx \mathcal{L}(G, A)$ for any $w \in \mathcal{C}(G, A)$.

The class of languages generated by grammars in $\mathbb{G}(r)$ is denoted by $\mathbb{L}(r)$.

Since the generative power of congruential CFGs and that of c-deterministic CFGs are incomparable, our new class $\mathbb{G}(2)$ defines a properly richer class of languages than those existing classes. An easy example is the disjoint union of a language $L_1 \subseteq \Sigma_1^*$ that is congruential and not c-deterministic and another $L_2 \subseteq \Sigma_2^*$ that is c-deterministic and not congruential, where $\Sigma_1 \cap \Sigma_2 = \varnothing$.

[1] Pairs $\langle X^\dagger, X^\ddagger \rangle$ with the order $\preccurlyeq$ form a lattice structure, called a *syntactic concept lattice* proposed by Clark [2], and indeed technical discussions of this paper would be better understood with that background. This paper does not go into the details of them due to the space limit, but is technically self-contained.

Clark [4] has introduced a hierarchy of language classes that are more general than c-deterministic CFGs and Yoshinaka [11] has discussed the "primal" counterpart of them, which is a generalization of the property introduced by Clark et al. [6]. An integration should be defined as follows.

**Definition 2.** By $\mathbb{G}(p,q,r)$ we denote the class of CFGs $G$ such that each rule has at most $r$ nonterminals on the right-hand side and each nonterminal $A$ admits either

– a finite string set $V_A \subseteq \Sigma^*$ such that $|V_A| \leq p$ and $V_A \approx \mathcal{L}(G,A)$, or
– a finite context set $W_A \subseteq \Sigma^* \times \Sigma^*$ such that $|W_A| \leq q$ and $W_A \approx \mathcal{L}(G,A)$.

The class of languages generated by grammars in $\mathbb{G}(p,q,r)$ is denoted as $\mathbb{L}(p,q,r)$.

We call such $V_A$ or $W_A$ a *characterizing set* for a nonterminal $A$. We note that $V_A \subseteq \mathrm{Sub}(\mathcal{L}(G))$ and $W_A \subseteq \mathrm{Con}(\mathcal{L}(G))$. If $G \in \mathbb{G}(r)$, for each $A$, either every $v \in \mathcal{L}(G,A)$ or every $w \in \mathcal{C}(G,A)$ forms a characterizing singleton set for $A$. In fact, $\mathbb{L}(r)$ is a proper subclass of $\mathbb{L}(1,1,r)$. An example in the difference is $\{\, a^m b^n \mid m < n \,\}$.

*Example 2.* For $n \geq 3$, let

$$L_n = \{\, a_1^{k_1} a_2^{k_2} \dots a_n^{k_n} \mid k_1,\dots,k_n \in \mathbb{N} \text{ and } k_i = k_j \text{ for some } i \neq j \,\}\,.$$

One can show that $L_n \in \mathbb{L}(p,q,1)$ if $p+q > n$. The author has not found a grammar in $\mathbb{G}(p,q,r)$ with $p+q \leq n$ that generates $L_n$ except when $\{p,q\} = \{0,n\}$ or $\{p,q\} = \{1,n-1\}$.

We remark that although all the preceding algorithms [3,4,9,11] that base our algorithms consider only CFGs in CNF, this paper lifts this assumption. In fact none of the properties except the congruentiality and 1-FKP are shown to be closed under Chomsky-normalization. For example, every regular language is generated by a c-deterministic grammar [9], but no c-deterministic grammar in CNF generates a finite language $\{ac, ad, bc\}$. Suppose otherwise, if a c-deterministic grammar in CNF generates $\{ac, ad, bc\}$, we must have a derivation $S \Rightarrow_G AC \Rightarrow_G^* ac$ for some nonterminals $A, C$. The fact $\langle \lambda, c\rangle \in \mathcal{C}(G,A)$ implies that $\{\langle \lambda, c\rangle\}^\dagger = \{a,b\} \approx \mathcal{L}(G,A)$, which implies $\mathcal{L}(G,A) = \{a,b\}$. Similarly we have $\mathcal{L}(G,C) = \{c,d\}$. This contradicts the fact $bd \notin \mathcal{L}(G)$. Another such example is the language of palindromes: $\{\, v \in \Sigma^* \mid v = v^{\mathrm{R}} \,\}$ where $v^{\mathrm{R}}$ denotes the reverse of $v$. The palindrome language is generated by a c-deterministic CFG, but it cannot be in CNF.

We work under different learning schemes for $\mathbb{G}(r)$ and $\mathbb{G}(p,q,r)$ in accordance with the preceding work. Our learner for $\mathbb{G}(r)$ works with a minimally adequate teacher (MAT) following [3,9], and another identifies $\mathbb{G}(p,q,r)$ in the limit from positive data and membership queries like [4,11]. Yet both types of our learners construct a conjecture in the same manner.

## 4 Hypotheses

### 4.1 Construction

Hereafter we arbitrarily fix three natural numbers $p, q, r \geq 1$ and a target language $L_* \in \mathbb{L}(p, q, r)$ to be learnt. When learning $\mathbb{G}(r)$, assume $p = 1$ and $q = 1$. We assume that our learner has an access to an oracle who answers *membership queries* (MQs), which ask whether an arbitrary string $u$ belongs to the learning target $L_*$. Our conjecture $\hat{G} = \mathcal{G}_{p,q,r}(J, K, E, F)$ is constructed from two finite sets of strings $J, K \subseteq \Sigma^*$ and two finite sets of contexts $E, F \subseteq \Sigma^* \times \Sigma^*$ with the aid of the oracle. The nonterminal set $\hat{N}$ of $\hat{G}$ is constituted from two parts:

$$\begin{aligned} \hat{N} &= \hat{N}_K \cup \hat{N}_F \text{ where} \\ \hat{N}_K &= \{\, [\![V]\!] \mid V \subseteq K \text{ with } |V| \leq p \,\}, \\ \hat{N}_F &= \{\, [\![W]\!] \mid W \subseteq F \text{ with } |W| \leq q \,\}, \end{aligned}$$

where $[\![X]\!]$ simply means a symbol indexed with $X$. For legibility we write $[\![x]\!]$ for $[\![\{x\}]\!]$. We would like each nonterminal $[\![X]\!] \in \hat{N}$ to generate $X^\dagger$. Accordingly if we have a rule of the form

$$[\![X_0]\!] \to u_0[\![X_1]\!]u_1[\![X_2]\!]u_2 \dots [\![X_k]\!]u_k \,,$$

by the nature of the context-free derivation, it should hold that $u_0X_1^\dagger u_1X_2^\dagger u_2 \dots X_k^\dagger u_k \subseteq X_0^\dagger$. Equivalently,

$$X_0^\ddagger \odot u_0X_1^\dagger u_1X_2^\dagger u_2 \dots X_k^\dagger u_k \subseteq L_*. \qquad (\star)$$

Yet in general the inclusion relation ($\star$) is not computable. We substitute some computable finite sets for $X_0^\ddagger$ and $X_i^\dagger$ to "approximate" the inclusion relation ($\star$). By the aid of finitely many MQs, the approximation will be decidable. For $[\![X_i]\!] \in \hat{N}_K$ with $i \geq 1$, we have $X_i \approx X_i^\dagger$, thus $X_i$ can be substituted for $X_i^\dagger$ in ($\star$). For $[\![X_i]\!] \in \hat{N}_F$, we substitute $X_i^\dagger \cap J$ for $X_i^\dagger$, which is also computable by the aid of MQs, though it is not necessarily the case that $X_i^\dagger \cap J \approx X_i^\dagger$. On the other hand, if $[\![X_0]\!] \in \hat{N}_F$, we have $X_0 \approx X_0^\ddagger$, thus $X_0$ can be substituted for $X_0^\ddagger$ in ($\star$), but if $[\![X_0]\!] \in \hat{N}_K$, we use $X_0^\ddagger \cap E$ for $X_0^\ddagger$. For a uniform treatment of nonterminals in $\hat{N}_K$ and $\hat{N}_F$, we introduce the following notation. The string interpretation of $[\![X]\!] \in \hat{N}$ is

$$[\![X]\!]^{(J)} = \begin{cases} X & \text{if } [\![X]\!] \in \hat{N}_K; \\ X^\dagger \cap J = \{\, v \in J \mid X \odot v \subseteq L_* \,\} & \text{if } [\![X]\!] \in \hat{N}_F. \end{cases}$$

and the context interpretation of $[\![X]\!] \in \hat{N}$ is

$$[\![X]\!]^{(E)} = \begin{cases} X^\ddagger \cap E = \{\, w \in E \mid w \odot X \subseteq L_* \,\} & \text{if } [\![X]\!] \in \hat{N}_K; \\ X & \text{if } [\![X]\!] \in \hat{N}_F. \end{cases}$$

Our grammar $\mathcal{G}_{p,q,r}(J, K, E, F)$ has rules of the form

$$[\![X_0]\!] \to u_0[\![X_1]\!]u_1[\![X_2]\!]u_2 \dots [\![X_k]\!]u_k$$

if $0 \le k \le r$, $u_0, \dots, u_k \in K$ and[2]

$$X_0^{(E)} \odot u_0 X_1^{(J)} u_1 \dots X_k^{(J)} u_k \subseteq L_*.$$

The start symbol is $[\![\Lambda]\!] \in \hat{N}_F$.

**Lemma 3.** *One can construct $\mathcal{G}_{p,q,r}(J, K, E, F) = \langle \Sigma, \hat{N}, \hat{P}, [\![\Lambda]\!] \rangle$ in polynomial time with the aid of* MQ*s, where the degree of the polynomial depends on $p, q, r$. We have $|\hat{N}_K| \le |K|^p$, $|\hat{N}_F| \le |F|^q$ and $|\hat{P}| \le |\hat{N}|^{r+1}|K|^{r+1}$.*

## 4.2 Properties of Hypotheses

**Lemma 4.** *Let $\hat{G} = \mathcal{G}_{p,q,r}(J, K, E, F)$ and $\hat{G}' = \mathcal{G}_{p,q,r}(J', K', E', F')$.*

1. *If $J \subseteq J'$, $K = K'$, $E = E'$ and $F = F'$, then $\mathcal{L}(\hat{G}) \supseteq \mathcal{L}(\hat{G}')$.*
2. *If $J = J'$, $K \subseteq K'$, $E = E'$ and $F = F'$, then $\mathcal{L}(\hat{G}) \subseteq \mathcal{L}(\hat{G}')$.*
3. *If $J = J'$, $K = K'$, $E \subseteq E'$ and $F = F'$, then $\mathcal{L}(\hat{G}) \supseteq \mathcal{L}(\hat{G}')$.*
4. *If $J = J'$, $K = K'$, $E = E'$ and $F \subseteq F'$, then $\mathcal{L}(\hat{G}) \subseteq \mathcal{L}(\hat{G}')$.*

*Proof.* (1,3) By definition, every rule of $\hat{G}'$ is also a rule of $\hat{G}$.
(2,4) Every rule of $\hat{G}$ is also a rule of $\hat{G}'$. □

We say that a rule of the form $[\![X_0]\!] \to u_0[\![X_1]\!]u_1 \dots [\![X_k]\!]u_k$ is *correct* if ($\star$) holds. If a rule is not correct, it is *incorrect*. It is clear that the inclusion relation ($\star$) implies $X_0^{(E)} \odot u_0 X_1^{(J)} u_1 \dots X_k^{(J)} u_k \subseteq L_*$.

**Lemma 5.** *Every pair $\langle K, F \rangle$ admits a pair $\langle E, J \rangle$ of sets of a polynomial cardinality such that $\hat{G} = \mathcal{G}_{p,q,r}(J, K, E, F)$ has no incorrect rules.*

*Proof.* If a rule $[\![X_0]\!] \to u_0[\![X_1]\!]u_1 \dots [\![X_k]\!]u_k$ is incorrect, there are $w \in X_0^{\ddagger}$ and $v_i \in X_i^{\dagger}$ for $i = 1, \dots, k$ such that $w \odot u_0 v_1 u_1 \dots v_k u_k \notin L_*$. If $[\![X_0]\!] \in \hat{N}_F$, we may assume that $w \in X_0$ without loss of generality. For $i \in \{1, \dots, k\}$, if $[\![X_i]\!] \in \hat{N}_K$, we may assume that $v_i \in X_i$. If $[\![X_0]\!] \in \hat{N}_K$, put $w$ into $E$ and if $[\![X_i]\!] \in \hat{N}_F$, put $v_i$ into $J$. This way we need at most one context and at most $k$ strings to suppressed the incorrect rule. On the other hand, there are at most $(|\hat{N}||K|)^{r+1}$ rules. Hence, the lemma holds. □

We note that if $[\![X_0]\!] \in \hat{N}_F$ and $[\![X_i]\!] \in \hat{N}_K$ for all $i = 1, \dots, k$, the rule is always correct. We say that a pair $\langle E, J \rangle$ is *fiducial on $\langle K, F \rangle$ with respect to $L_*$* if $\mathcal{G}_{p,q,r}(J, K, E, F)$ has no incorrect rules.

**Lemma 6.** *If $\hat{G} = \mathcal{G}_{p,q,r}(J, K, E, F)$ has no incorrect rules, then $\mathcal{L}(\hat{G}) \subseteq L_*$.*

[2] A little more reasonable construction is to limit $u_0, \dots, u_k$ to have $v_1, \dots, v_k$ such that $u_0 v_1 u_1 \dots v_k u_k \in K$, but we ignore such technical detailed modifications.

*Proof.* We show by induction on the length of derivation that $v \in X^\dagger$ if $[\![X]\!] \Rightarrow^*_{\hat{G}} v$ with $v \in \Sigma^*$, which implies particularly for the start symbol $[\![X]\!] = [\![\Lambda]\!]$, we have $v \in X^\dagger = L_*$.

Suppose that $[\![X_0]\!] \to u_0[\![X_1]\!]u_1 \dots [\![X_k]\!]u_k$ is a correct rule of $\hat{G}$ and $[\![X_0]\!]$ derives $u_0v_1u_1 \dots v_ku_k$ where $[\![X_i]\!]$ derives $v_i$ for $i = 1, \dots, k$. By the induction hypothesis we have $v_i \in X_i^\dagger$. Since the rule is correct, we have

$$X_0^\ddagger \odot u_0v_1u_1 \dots v_ku_k \subseteq X_0^\ddagger \odot u_0X_1^\dagger u_1 \dots X_k^\dagger u_k \subseteq L_*,$$

which means $u_0v_1u_1 \dots v_ku_k \in X_0^\dagger$. □

**Lemma 7.** *Let $G_* = \langle \Sigma, N_*, P_*, S_* \rangle \in \mathbb{G}(p,q,r)$ generate $L_*$ and $X_A$ a characterizing set for each $A \in N_*$. We define a homomorphism $h$ by $h(A) = [\![X_A]\!]$ and $h(a) = a$ for $a \in \Sigma$. For any rule $A \to \alpha$ of $G_*$, the rule $h(A) \to h(\alpha)$ is correct. Moreover if $A$ is the start symbol of $G_*$, $[\![\Lambda]\!] \to h(\alpha)$ is a correct rule.*

*Proof.* For any rule $B_0 \to u_0B_1u_1 \dots B_ku_k$, by the nature of context-free derivation, we have $u_0L_1u_1 \dots L_ku_k \subseteq L_0$ where $L_i = \mathcal{L}(G_*, B_i)$ for $i = 0, \dots, k$. For any characterizing set $X_i$ of $B_i$, we have

$$X_0^\ddagger \odot u_0X_1^\dagger u_1 \dots X_k^\dagger u_k \approx L_0^\ddagger \odot u_0L_1u_1 \dots L_ku_k \subseteq L_*$$

and hence the rule $[\![X_0]\!] \to u_0[\![X_1]\!]u_1 \dots [\![X_k]\!]u_k$ is correct by Lemma 1. Moreover if $B_0$ is the start symbol, we have $u_0L_1u_1 \dots L_ku_k \subseteq L_*$, which implies that $\Lambda \odot u_0X_1^\ddagger u_1 \dots X_k^\ddagger u_k \approx u_0L_1u_1 \dots L_ku_k \subseteq L_*$. Hence the rule $[\![\Lambda]\!] \to u_0[\![X_1]\!]u_1 \dots [\![X_k]\!]u_k$ is correct. □

**Corollary 8.** *Suppose that $K \cup F$ includes a characterizing set for each nonterminal $A$ of a* CFG *$G_*$ with $L_* = \mathcal{L}(G_*)$ and $K$ includes $u_0, \dots, u_k$ for each rule $A \to u_0B_1u_1 \dots B_ku_k$ of $G_*$. Then $L_* \subseteq \mathcal{L}(\mathcal{G}_{p,q,r}(J,K,E,F))$.*

Therefore, by Lemma 6 and Corollary 8, if $K$ and $F$ are big enough to cover a characterizing set of every nonterminal of the learning target $G_*$, and $E$ and $J$ are big enough with respect to $K$ and $F$ to suppress incorrect rules, then our conjecture $\mathcal{G}_{p,q,r}(J,K,E,F)$ generates exactly the target language $\mathcal{L}(G_*)$.

## 5 Learning of $\mathbb{G}(p,q,r)$

### 5.1 Identification in the Limit from Positive Data and Membership Queries

Our learning paradigm is *identification in the limit from positive data and membership queries.* A *positive presentation* of a language $L_*$ over $\Sigma$ is an infinite sequence $u_1, u_2, \dots \in \Sigma^*$ such that $L_* = \{\, u_i \mid i \geq 1 \,\}$. A learner is given a positive presentation of the language $L_* = \mathcal{L}(G_*)$ of the target grammar $G_*$ and each time a new example $u_i$ is given, it outputs a grammar $G_i$ computed from $u_1, \dots, u_i$ with the aid of an oracle who answers MQs. We say that a learning algorithm *identifies* $G_*$ *in the limit from positive data and membership queries* if for any positive presentation $u_1, u_2, \dots$ of $\mathcal{L}(G_*)$, there is an integer $n$ such that $G_n = G_m$ for all $m \geq n$ and $\mathcal{L}(G_n) = \mathcal{L}(G_*)$. We say that a learning algorithm *identifies a class* $\mathbb{G}$ *of grammars in the limit from positive data and membership queries* iff it identifies all $G \in \mathbb{G}$.

## 5.2 Learning Algorithm for $\mathbb{G}(p, q, r)$

Our learner is shown as Algorithm 1. When the learner observes that some positive example is not generated by the current conjecture, it expands $K$ and $F$ so that they shall cover a characterizing set of some nonterminal of a target grammar $G_*$. On the other hand, to get rid of incorrect rules, it keeps expanding $E$ and $J$.

**Algorithm 1.** Learn $\mathbb{G}(p, q, r)$

```
Data: A sequence of strings $u_1, u_2, \dots \in L_*$; membership oracle $\mathcal{O}$ for $L_*$;
Result: A sequence of CFGs $G_1, G_2, \dots$
let $D := \varnothing$; $J, K, E, F := \varnothing$; $\hat{G} := \mathcal{G}_{p,q,r}(J, K, E, F)$;
for $n = 1, 2, \dots$ do
    let $D := D \cup \{u_n\}$; $J := \mathrm{Sub}(D)$; $E := \mathrm{Con}(D)$;
    if $D \not\subseteq \mathcal{L}(\hat{G})$ then
        let $K := J$; $F := E$;
    end if
    output $\hat{G} = \mathcal{G}_{p,q,r}(J, K, E, F)$ as $G_n$;
end for
```

**Lemma 9.** *If the current conjecture $\hat{G}$ does not precisely generate $L_*$, then the learner will discard $\hat{G}$ at some point.*

*Proof.* If $L_* \not\subseteq \mathcal{L}(\hat{G})$, the learner will get some element $u \in L_* - \mathcal{L}(\hat{G})$ at some point and discard the current conjecture $\hat{G}$, where the new conjecture has a new rule $[\![\Lambda]\!] \to u$ at least.

If $\mathcal{L}(\hat{G}) \not\subseteq L_*$, Lemma 6 implies that $\langle E, J\rangle$ is not fiducial on $\langle K, F\rangle$. All incorrect rules will be removed at some point by Lemma 5. □

**Theorem 10.** *Algoritm 1 identifies $\mathbb{L}(p, q, r)$ in the limit.*

*Proof.* By Lemma 9, the learner never converges to a wrong hypothesis. It is impossible that the sets $K, F$ are changed infinitely many times, because $K, F$ are monotonically expanded and sometime $K, F$ will contain a characterizing set of every nonterminal of a target grammar $G_*$, in which case the learner never updates $K, F$ any more by Corollary 8. Then sometime $\langle E, J\rangle$ will be fiducial on the converged $\langle K, F\rangle$ by Lemma 5, where $\hat{G}$ generates the target language by Lemma 6. Thereafter no rules will be added to or removed from $\hat{G}$ any more. □

The literature has established no consensus on the definition of polynomial-time identification in the limit. Here we remark nontrivial properties on the efficiency of our algorithm. First, Lemma 3 implies that Algorithm 1 updates its conjecture in polynomial time in the size of the given positive examples. Second, there is a set $D \subseteq L_*$ with $|D| \leq \max\{p,q\}|N_*| + r|P_*|$ such that $K = \mathrm{Sub}(D)$ and $F = \mathrm{Con}(D)$ satisfy the condition of Corollary 8. Together with Lemma 5, we claim that the amount of data needed for convergence is not too big.

## 6 Learning of $\mathbb{G}(r)$

### 6.1 Minimally Adequate Teacher

Our learner for $\mathbb{G}(r)$ works under Angluin's MAT learning model [1]. A learner has an oracle who answers *equivalence queries (*EQ*s)* in addition to MQs. An instance of an EQ is a grammar $\hat{G}$ and the oracle answers "YES" if $\mathcal{L}(\hat{G}) = L_*$ and otherwise returns a counterexample $u \in (L_* - \mathcal{L}(\hat{G})) \cup (\mathcal{L}(\hat{G}) - L_*)$. A counterexample is called *positive* if $u \in L_* - \mathcal{L}(\hat{G})$ and it is *negative* if $u \in \mathcal{L}(\hat{G}) - L_*$. The oracle is supposed to answer every query in constant time. The learning process finishes when the oracle answers "YES" to an EQ.

In this learning scheme, at any point in the run, we allow a leaner to use time bounded polynomially in $\|G_*\|$ and $\ell$ where $G_*$ is a smallest grammar such that $\mathcal{L}(G_*) = L_*$ and $\ell$ is the length of the longest counterexample given by the oracle.

### 6.2 Learning Algorithm for $\mathbb{G}(r)$

Algorithm 2 is our learner for $\mathbb{G}(r)$. We show that the learner makes at most polynomially many EQs and that it takes polynomial time to raise another EQ after receiving an answer to an EQ. We first explain the case when a positive counterexample is given to the learner and later discuss the other case where the learner gets a negative counterexample. We let $G_* = \langle \Sigma, N_*, P_*, S_* \rangle \in \mathbb{G}(r)$ be our learning target.

When Algorithm 2 gets a positive counterexample. obviously it updates its conjecture in polynomial time. We show that the number of positive counterexamples that the learner receives is bounded by the number of rules in $P_*$ of the target grammar $G_*$.

We divide the set $N_*$ of nonterminals of $G_*$ into two sets $N_1$ and $N_2$ so that any $v \in \mathcal{L}(G_*, B)$ characterizes $B \in N_1$ and any $w \in \mathcal{C}(G_*, A)$ characterizes $A \in N_2$. We let

$$\begin{aligned}
\Delta_0(K) &= \{\, A \to u_0 B_1 u_1 \dots B_k u_k \in P_* \mid u_0, \dots, u_k \in K \,\},\\
\Delta_1(K) &= \{\, B \in N_1 \mid K \cap \mathcal{L}(G_*, B) \neq \varnothing \,\},\\
\Delta_2(F) &= \{\, A \in N_2 \mid F \cap \mathcal{C}(G_*, A) \neq \varnothing \,\}.
\end{aligned}$$

If $\Delta_0(K) = P_*$, $\Delta_1(K) = N_1$ and $\Delta_2(F) = N_2$, Corollary 8 ensures that $\mathcal{L}(G_*) \subseteq \mathcal{L}(\mathcal{G}(J, K, E, F))$.

**Algorithm 2.** Learn $\mathbb{G}(r)$

```
let $J := K := E := F := \varnothing$; $\hat{G} = \mathcal{G}_{1,1,r}(J, K, E, F)$;
while the oracle does not answer "YES" to the EQ on $\hat{G}$ do
    let $u$ be the counterexample from the oracle;
    if $u \in L_* - \mathcal{L}(\hat{G})$ then
        let $K := K \cup \mathrm{Sub}(\{u\})$ and $F := F \cup \mathrm{Con}(\{u\})$;
    else
        let $J := J \cup \textsc{Witness}_1(\hat{G}, u)$ and $E := E \cup \textsc{Witness}_2(\hat{G}, u)$;
    end if
    let $\hat{G} = \mathcal{G}_{1,1,r}(J, K, E, F)$;
end while
output $\hat{G}$;
```

**Lemma 11.** *Each time the learner gets a positive counterexample, at least one of the sets $\Delta_0(K)$, $\Delta_1(K)$ and $\Delta_2(F)$ is properly expanded.*

*Proof.* Obviously none of $\Delta_0(K)$, $\Delta_1(K)$ and $\Delta_2(F)$ gets shrunk. Suppose that the learner has got a positive counterexample $u \in \mathcal{L}(G_*) - \mathcal{L}(\hat{G})$, in which case, a derivation of $u$ by $G_*$ involves either a rule $\pi$ with $\pi \notin \Delta_0(K)$, a nonterminal $B \in N_1$ with $B \notin \Delta_1(K)$, or a nonterminal $A \in N_2$ with $A \notin \Delta_2(F)$. In the respective cases, we have $\pi \in \Delta_0(K \cup \mathrm{Sub}(u))$, $B \in \Delta_1(K \cup \mathrm{Sub}(u))$, and $A \in \Delta_2(F \cup \mathrm{Con}(u))$. The lemma holds. □

**Corollary 12.** *The algorithm receives a positive counterexample at most $|P_*|$ times. Hence $|K|, |F| \in \mathrm{O}(|P_*|\ell^2)$, where $\ell$ is the length of a longest positive counterexample given so far. The number of nonterminals of a conjecture is bounded by $|\hat{N}| \leq |K| + |F| \in \mathrm{O}(|P_*|\ell^2)$ and the number of rules is at most $|\hat{N}|^{r+1}|K|^{r+1} \in \mathrm{O}((|P_*|\ell^2)^{2(r+1)})$.*

We next suppose that a negative counterexample $u \in \mathcal{L}(\hat{G}) - L_*$ is given. The proof of Lemma 6 implies that our grammar $\hat{G}$ uses an incorrect rule to derive $u$. In order to find an incorrect rule, we first parse $u$ with $\hat{G}$ and (implicitly) get a derivation tree $t$ for $u$. We then search $t$ in a topdown manner for an incorrect rule that has contributed to the overgeneralization. If a nonterminal $[\![x]\!]$ of our conjecture $\hat{G}$ derives $v$ such that $v \notin \{x\}^\dagger$, an incorrect rule must be used in the derivation. In the case of $[\![x]\!] \in \hat{N}_F$, $v \notin \{x\}^\dagger$ means $x \odot v \notin L_*$. On the other hand, if $[\![x]\!] \in \hat{N}_K$, $v \notin \{x\}^\dagger$ means that there is $w \in \Sigma^* \times \Sigma^*$ such that $w \odot x \in L_*$ and $w \odot v \notin L_*$. Hence in the recursive procedure, we maintain such a context $w \in \{x\}^\ddagger - \{v\}^\ddagger$ to evidence that the derivation $[\![x]\!] \Rightarrow^*_{\hat{G}} v$ involves an incorrect rule. Actually what the following procedure outputs is not an incorrect rule, but witnesses to be added to $J$ and/or $E$ to get rid of the rule. The recursive procedure Witness takes a triple $\langle x, v, w\rangle$ such that $[\![x]\!] \Rightarrow^*_{\hat{G}} v$ and $w \in \{x\}^\ddagger - \{v\}^\ddagger$, where $w = x$ if $[\![x]\!] \in \hat{N}_F$. We define $\textsc{Witness}_1(\hat{G}, u) = \textsc{Witness}(\Lambda, u, \Lambda) \cap \Sigma^*$ and $\textsc{Witness}_2(\hat{G}, u) = \textsc{Witness}(\Lambda, u, \Lambda) \cap (\Sigma^* \times \Sigma^*)$.

Suppose that WITNESS$(x, v, w)$ is called. That is, $[\![x]\!] \Rightarrow^*_{\hat{G}} v$ and $w \in \{x\}^\ddagger - \{v\}^\ddagger$. Let the problematic derivation be

$$[\![x]\!] \underset{\hat{G}}{\Rightarrow} u_0[\![x_1]\!]u_1 \dots [\![x_k]\!]u_k \overset{*}{\underset{\hat{G}}{\Rightarrow}} u_0 v_1 u_1 \dots v_k u_k = v\,, \quad \text{with } [\![x_i]\!] \overset{*}{\underset{\hat{G}}{\Rightarrow}} v_i\,.$$

We partition $\{1, \dots, k\}$ into $I_K = \{\, i \mid x_i \in K \,\}$ and $I_F = \{\, i \mid x_i \in F \,\}$.

If there is $i \in I_F$ such that $x_i \odot v_i \notin L_*$, we recursively call WITNESS$(x_i, v_i, x_i)$.

Suppose that $x_i \odot v_i \subseteq L_*$ for all $i \in I_F$. Let

$$\alpha_j = u_0 v_1 u_1 \dots v_j u_j \beta_{j+1} u_{j+1} \dots \beta_k u_k$$
$$\text{where } \beta_i = \begin{cases} v_i & \text{for } i \in I_F\,; \\ x_i & \text{for } i \in I_K\,, \end{cases}$$

for $j = 0, \dots, k$. If $w \odot \alpha_0 \notin L_*$, by

$$w \odot \alpha_0 \in \{x\}^\ddagger \odot u_0\{x_1\}^\dagger u_1 \dots \{x_k\}^\dagger u_k,$$

the rule $[\![x]\!] \to u_0[\![x_1]\!]u_1 \dots [\![x_k]\!]u_k$ is incorrect. Return the witnesses $w \in \{x\}^\ddagger$ if $[\![x]\!] \in \hat{N}_K$ and $v_i \in \{x_i\}^\dagger$ for all $i \in I_F$, and halt the recursive procedure.

We now suppose $w \odot \alpha_0 \in L_*$. On the other hand we know that $w \odot \alpha_k \notin L_*$. Hence one can find $j$ such that

$$\begin{aligned} w \odot \alpha_{j-1} &= w \odot u_0 v_1 u_1 \dots v_{j-1} u_{j-1} \beta_j u_j \beta_{j+1} u_{j+1} \dots \beta_k u_k \in L_*\,, \\ w \odot \alpha_j &= w \odot u_0 v_1 u_1 \dots v_{j-1} u_{j-1} v_j u_j \beta_{j+1} u_{j+1} \dots \beta_k u_k \notin L_*\,. \end{aligned}$$

Here we have $\beta_j \neq v_j$, i.e., $\beta_j = x_j \in K$. Apply WITNESS to the triple

$$\langle x_j, v_j, \langle w_1 u_0 v_1 u_1 \dots v_{j-1} u_{j-1}, u_j \beta_{j+1} u_{j+1} \dots \beta_k u_k w_2 \rangle\rangle \tag{$\star\star$}$$

where $w = \langle w_1, w_2 \rangle$.

**Lemma 13.** *Let $\hat{G} = \mathcal{G}_{1,1,r}(J, K, E, F)$ and $u \in \mathcal{L}(\hat{G}) - L_*$.* WITNESS$(\Lambda, u, \Lambda)$ *runs in polynomial time in $|u|$ and $\|\hat{G}\|$. Moreover $\mathcal{G}_{1,1,r}(J', K, E', F)$ has strictly less incorrect rules than $\hat{G}$ where $E' = E \cup$* WITNESS$_1(\hat{G}, u)$ *and $J' = J \cup$* WITNESS$_2(\hat{G}, u)$.

*Proof.* Clearly the number of recursive calls of the procedure is bounded by the height $h$ of the derivation tree for $u$, where we may assume without loss of generality that $h \leq |u||\hat{N}|$ by prohibiting the derivation tree from containing a vacuous circular derivation of the form $A \Rightarrow^+_{\hat{G}} A$. Clearly in each recursion, to call itself with another argument or to halt takes polynomial time in the size $|x| + |v| + |w|$ of the argument $\langle x, v, w \rangle$. Since $[\![x]\!] \in \hat{N}$, it is enough to give a polynomial bound on $|v| + |w|$. Suppose that the last case of the recursion happens, which is the only nontrivial case. Observing ($\star\star$), one sees that $|v| + |w|$ can increase by

$$\begin{aligned} (|v_i| + |w_1 u_0 v_1 u_1 \dots v_{j-1} u_{j-1}| + |u_j \beta_{j+1} u_{j+1} \dots \beta_k u_k w_2|) - (|v| + |w_1| + |w_2|) \\ \leq |\beta_{j+1} \dots \beta_k| \leq rs \end{aligned}$$

for $s = \max\{\, |x| \mid x \in K\,\}$. All in all, the computation of Witness$(\Lambda, u, \Lambda)$ is done in polynomial time in $|u|$ and $\|\hat{G}\|$.

The latter claim of the lemma is easy to see by the construction. □

**Lemma 14.** *The number of times that the algorithm receives a negative counterexample and the cardinality of $E, J$ are both polynomially bounded by $|P_*|$ and $\ell$ where $\ell$ is the length of a longest counterexample given so far.*

*Proof.* Each time the learner receives a negative counterexample, at most $r$ elements are added to $J$, at most one element is added to $E$ and at least one incorrect rule is removed. Hence the number of negative counterexamples is no more than the number $|\hat{P}|$ of rules constructed from $K$ and $F$, i.e., $|\hat{P}| \leq \sum_{k \leq r} |\hat{N}|^{k+1}|K|^{k+1} \in \mathrm{O}((|P_*|\ell^2)^{2r+2})$ by Corollary 12 and $|E \cup J| \in \mathrm{O}((r+1)(|P_*|\ell^2)^{2r+2})$. □

**Theorem 15.** *Algorithm 2 learns $\mathbb{G}(r)$ with a* MAT *in polynomial time.*

*Proof.* By Lemmas 3, 13 and 14 and Corollary 12.

## 7 Concluding Remarks

The language $L_n$ of Example 2 is in $\mathbb{L}(n, 0) \cap \mathbb{L}(0, n)$, which can be learnt by any of the existing algorithms by Clark [4] and by Yoshinaka [11]. Yet our algorithm almost halves the degree of the polynomial of the updating time by Lemma 3.

It is known that the idea of the distributional learning is applied to several extensions of CFGs [7,10,12]. Our techniques are easily applied to those extensions as well.

## References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
2. Clark, A.: A Learnable Representation for Syntax Using Residuated Lattices. In: de Groote, P., Egg, M., Kallmeyer, L. (eds.) FG 2009. LNCS, vol. 5591, pp. 183–198. Springer, Heidelberg (2011)
3. Clark, A.: Distributional learning of some context-free languages with a minimally adequate teacher. In: Sempere, García (eds.) [8], pp. 24–37
4. Clark, A.: Learning context free grammars with the syntactic concept lattice. In: Sempere, García (eds.) [8], pp. 38–51
5. Clark, A.: Towards General Algorithms for Grammatical Inference. In: Hutter, M., Stephan, F., Vovk, V., Zeugmann, T. (eds.) ALT 2010. LNCS, vol. 6331, pp. 11–30. Springer, Heidelberg (2010)
6. Clark, A., Eyraud, R., Habrard, A.: A note on contextual binary feature grammars. In: EACL 2009 Workshop on Computational Linguistic Aspects of Grammatical Inference, pp. 33–40 (2009)
7. Kasprzik, A., Yoshinaka, R.: Distributional Learning of Simple Context-Free Tree Grammars. In: Kivinen, J., Szepesvári, C., Ukkonen, E., Zeugmann, T. (eds.) ALT 2011. LNCS, vol. 6925, pp. 398–412. Springer, Heidelberg (2011)

8. Sempere, J.M., García, P. (eds.): ICGI 2010. LNCS, vol. 6339. Springer, Heidelberg (2010)
9. Shirakawa, H., Yokomori, T.: Polynomial-time MAT learning of c-deterministic context-free grammars. Transaction of Information Processing Society of Japan 34, 380–390 (1993)
10. Yoshinaka, R.: Efficient learning of multiple context-free languages with multidimensional substitutability from positive data. Theoretical Computer Science 412(19), 1821–1831 (2011)
11. Yoshinaka, R.: Towards Dual Approaches for Learning Context-Free Grammars Based on Syntactic Concept Lattices. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 429–440. Springer, Heidelberg (2011)
12. Yoshinaka, R., Kanazawa, M.: Distributional Learning of Abstract Categorial Grammars. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS, vol. 6736, pp. 251–266. Springer, Heidelberg (2011)

# Author Index