Yuan Gao
Hanlin Lu
Shinnosuke Seki
Sheng Yu (Eds.)

LNCS 6224

Developments in Language Theory

14th International Conference, DLT 2010
London, ON, Canada, August 2010
Proceedings

Springer

Yuan Gao Hanlin Lu
Shinnosuke Seki Sheng Yu (Eds.)

Developments in Language Theory

14th International Conference, DLT 2010
London, ON, Canada, August 17-20, 2010
Proceedings

Springer

Volume Editors

Yuan Gao
Hanlin Lu
Shinnosuke Seki
Sheng Yu
University of Western Ontario
Department of Computer Science
London, ON, N6A 5B7, Canada
E-mail: {ygao72,hlu47,sseki,syu}@csd.uwo.ca

Library of Congress Control Number: 2010930708

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

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

ISSN	0302-9743
ISBN-10	3-642-14454-3 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-14454-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

The 14th International Conference on Developments in Language Theory (DLT 2010) was held in London, Ontario, Canada on the beautiful campus of the University of Western Ontario. It was a four-day conference starting August 17 and ending August 20, 2010.

The DLT conference series is one of the major international conference series in language theory. It started in Turku, Finland in 1993. Initially, it was held once every two years. Since 2001, it has been held every year, odd years in Europe and even years in other continents.

The papers submitted to DLT 2010 were from 27 countries all over the world, which include Australia, Austria, Belgium, Canada, Czech Republic, Denmark, Finland, France, Germany, Hungary, Iceland, India, Italy, Japan, Latvia, Moldova, The Netherlands, Poland, Portugal, Russian Federation, Slovakia, South Korea, Spain, Sweden, Tunisia, UK, and the USA. Each paper was reviewed by three referees and discussed by the members of the Program Committee. Finally, 32 regular papers were selected by the Program Committee for presentation at the conference. There were six invited talks given at the conference. They were given by (in alphabetic order) Dora Giammarresi (Rome), Markus Holzer (Giessen), Oscar Ibarra (Santa Barbara), Lila Kari (London, Ontario), Michel Rigo (Liege), and Grzegorz Rozenber (Leiden). In addition, there were six posters on display at the conference. This volume includes all the 32 contributed papers, the papers or abstracts from the 6 invited speakers, and a 2-page abstract for each of the 6 poster papers.

We warmly thank all the invited speakers and all the authors of the submitted papers. Their efforts were the bases of the success of the conference.

We would like to thank all the members of the Program Committee and the external referees. Their work in evaluating the papers and comments during the discussions were essential to the decisions on the contributed papers. We would also like to thank all the members of the DLT Steering Committee, especially its former Chair Grzegorz Rosenberg, for their ideas and efforts in forming the Program Committee and selecting the invited speakers.

We wish to thank the conference sponsors: The University of Western Ontario, the Fields Institute, European Association for Theoretical Computer Science, and Academia Europaea.

We would also like to thank the staff of the Computer Science Editorial at Springer, for their help in making this volume available before the conference. Their timely instructions were very helpful to our preparation for this volume.

August 2010 Sheng Yu

Organization

DLT 2010 was organized by the Department of Computer Science, University of Western Ontario.

Program Committee

Janusz A. Brzozowski	Waterloo, Canada
Véronique Bruyère	Mons, Belgium
Cristian S. Calude	Auckland, New Zealand
Maxime Crochemore	Marne-la-Vallée, France
Clelia De Felice	Salerno, Italy
Aldo de Luca	Naples, Italy
Volker Diekert	Stuttgart, Germany
Michael Domaratzki	Winnipeg, Canada
Paul Gastin	Paris, France
Juraj Hromkovic	Zürich, Switzerland
Lucian Ilie	London, Canada
Masami Ito	Kyoto, Japan
Hendrik Jan Hoogeboom	Leiden, The Netherlands
Natasha Jonoska	Tampa, USA
Juhani Karhumäki	Turku, Finland
Werner Kuich	Wien, Austria
Giancarlo Mauri	Milan, Italy
Igor Potapov	Liverpool, UK
Kai Salomaa	Kingston, Canada
Mikhail Volkov	Yekaterinburg, Russia
Takashi Yokomori	Tokyo, Japan
Sheng Yu	London, Canada, Chair

Organizing Committee

Yuan Gao	London, Ontario, Canada
Hanlin Lu	London, Ontario, Canada
Cheryl McGrath	London, Ontario, Canada
Angie Muir	London, Ontario, Canada
Shinnosuke Seki	London, Ontario, Canada
Sheng Yu	London, Ontario, Canada, Chair

External Referees

Angela Angeleska
Golnaz Badkobeh
Paul Bell
Alberto Bertoni
Luc Boasson
Hans-Joachim Boeckenhauer
Benedikt Bollig
Bernd Borchert
Véronique Bruyère
Michelangelo Bucci
Cezar Campeanu
Felice Cardone
Arturo Carpi
Olivier Carton
Flavio D'Alessandro
Karma Dajani
Mark Daley
Alessandro De Luca
Alberto Dennunzio
Egor Dolzhenko
Pal Domosi
Laurent Doyen
Manfred Droste
Christiane Frougny
Kaoru Fujioka
Zoltan Fulop
Julia Gamzova
Blaise Genest
Daniela Genova
Amy Glen
Vesa Halava
Yo-Sub Han
Tero Harju
Juha Honkala
Mari Huova
Rachaël Jungers
Tomasz Jurdzinski
Eija Jurvanen
Jarkko Kari
Tomi Karki
Dennis Komm
Manfred Kudlek
Manfred Kufleitner
Dietrich Kuske
Ernst Leiss
Alberto Leporati
Gerhard Lischke
Anna Lubiw
Andreas Maletti
Sabrina Mantaci
Radu Mardare
Leonardo Mariani
Simone Martini
Tobias Moemke
Frantisek Mraz
Benedek Nagy
Cyril Nicaud
Joachim Niehren
Igor Nitto
Damian Niwinski
Alexander Okhotin
Fumiya Okubo
Mihai Oltean
Paritosh Pandya
Xiaoxue Piao
Joni Pirnot
Martin Platek
Antonio E. Porreca
Narad Rampersad
Michael Rao
Klaus Reinhardt
Romero-Jiménez Álvaro
Wojciech Rytter
Aleksi Saarela
Kalle Saari
Pierluigi San Pietro
Sylvain Schmitz
Marinella Sciortino
Jeffrey Shallit
Arseny Shur
Pedro V. Silva
Cristina Sirangelo
William F. Smyth
Andreas Sprock
Benjamin Stenberg
Wolfgan Steiner
Monika Steinova
German Tischler
Sophie Tison
Pascal Weil
Philipp Weis
Marc Zeitoun
Rosalba Zizza

Sponsoring Institutions

The University of Western Ontario
Research Western
The Fields Institute
European Association for Theoretical Computer Science
Academia Europaea

Table of Contents

Reaction Systems: A Model of Computation Inspired by Biochemistry

Andrzej Ehrenfeucht[1] and Grzegorz Rozenberg[1,2]

[1] Department of Computer Science, University of Colorado at Boulder
430 UCB, Boulder, CO 80309, USA
[2] Leiden Institute of Advanced Computer Science, Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands

Natural Computing (see, e.g., [5] or [6]) is concerned with human-designed computing inspired by nature as well as with computation taking place in nature. In other words, natural computing investigates models and computational techniques inspired by nature as well as it investigates, in terms of information processing, processes taking place in nature.

Both strands of research are very active. The first strand is perhaps more established. Well-known research areas here are cellular automata, neural computing, evolutionary computing, molecular computing, quantum computing, membrane computing, and swarm intelligence. Representative areas for the second research strand are computational nature self-assembly, computational nature of brain processes, computational nature of developmental processes, computational nature of biochemical reactions, and systems biology approach to bionetworks. Research in natural computing is gununinely interdisciplinary, and it (especially research from the second strand) underscores the important (but often forgotten in general public) fact that computer science is also the fundamental science of information processing. As such, it is also a basic science for other scientific disciplines, e.g., biology.

One of the fascinating goals of natural computing is to understand, in terms of information processing, the functioning of a living cell. An important step in this direction is understanding of interactions between biochemical reactions. Clearly, on a suitable (high) level of abstraction, the functioning of a living cell is determined by interactions of a huge number of biochemical reactions that take place in living cells.

An important initial observation is that interactions between biochemical reactions as well as the functioning of an individual reaction is determined by two mechanisms: facilitation and inhibition.

This leads to the formal definition of a reaction as a triplet of finite nonempty sets $b = (R, I, P)$, where R is the set of reactions that facilitate (are needed for) b to take place, I (disjoint from R) is the set of inhibitors each of which forbids b from taking place, and P is the set of products produced by b if/when it takes place. For b to take place in a given state T (a set of biochemical entities), R must be included in it, while I must be disjoint with it – we say then that b is enabled in/by T. Then, a finite set of reactions A is enabled in T if any of its reactions is enabled, and the result of A taking place in T is the union of the products of all reactions from A that are enabled in T.

A reaction system (see, e.g., [1]) is an ordered pair $\mathcal{A} = (S, A)$, where A is a finite set of reactions and S is a finite set of entities including all entities involved in reactions from A. The most straightforward view of $\mathcal{A}$ is that it creates a successor state from a given state (where a state of $\mathcal{A}$ is a subset of S): for $U, V \subseteq S$, V is a successor of U in $\mathcal{A}$ if $res_{\mathcal{A}}(U) \subseteq V$, where $res_{\mathcal{A}}(U)$ is the result of applying to U all reactions from A. This transition relation becomes a function (of two variables) if we consider also the notion of context: for $U, V, C \subseteq S$, V is the C-successor of U in $\mathcal{A}$ if $V = res_{\mathcal{A}}(U) \cup C$; we say then that C is the context part of V, and $res_{\mathcal{A}}(U)$ is the result part of V. A possible interpretation of the context part of the state is that it represents the influence of "the rest of the system," hence it is assumed that $\mathcal{A}$ is a part of a bigger (reaction) system. Given a sequence $\gamma = C_0, C_1, \cdots, C_n$ with $n \geq 1$ of contexts, and assuming that $W_0 = C_0$ is the initial state of $\mathcal{A}$, one gets deterministically the state sequence (determined by γ) $\tau = W_0, W_1, \cdots, W_n$, where for all $i \in \{1, \cdots, n\}$, $W_i = res_{\mathcal{A}}(W_{i-1}) \cup C_i$.

Thus the state sequence τ is determined by the interactive process (γ, δ), where δ is the sequence $res_{\mathcal{A}}(W_0), res_{\mathcal{A}}(W_1), \cdots, res_{\mathcal{A}}(W_n)$ – this interactive process formalizes the interaction between $\mathcal{A}$ and the rest of the system represented by γ. The notion of an interactive process is a central notion in investigating the behavior of reaction systems.

There are many research lines pursued in the framework of reaction systems. Here are some examples.

1. Investigation of the relationship between reaction systems and other models of computation, see, e.g., [1].
2. Investigation of problems intrinsic to biochemical systems, such as formation of compounds or evolution, see, e.g., [2].
3. Investigation of fundamental notions such as the notion of time in biochemical systems, see, e.g., [3].
4. Investigation of (transition) functions definable by reaction systems, see, e.g., [4].

In our lecture we will review some of these research lines. The lecture is of an expository character and it is self-contained.

References

1. Ehrenfeucht, A., Rozenberg, G.: Reaction Systems. Fundamenta Informaticae 75, 263–280 (2007)
2. Ehrenfeucht, A., Rozenberg, G.: Events and Modules in Reaction Systems. Theoretical Computer Science 376, 3–16 (2007)

3. Ehrenfeucht, A., Rozenberg, G.: Introducting Time in Reaction Systems. Theoretical Computer Science 410, 310–322 (2009)
4. Ehrenfeucht, A., Main, M., Rozenberg, G.: Combinatorics of Life and Death in Reaction Systems. International J. of Foundations of Computer Science 21 (2009) (to appear)
5. Kari, L., Rozenberg, G.: The Many Facets of Natural Computing. Communications of the ACM 51, 72–83 (2008)
6. Rozenberg, G.: Computer Science, Informatics, and Natural Computing – Personal Reflections. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) New Computational Paradigms – Changing Conceptions of What Is Computable. Springer, Heidelberg (2007)

A Brief Excursion Inside the Class of Tiling Recognizable Two-Dimensional Languages*

Dora Giammarresi

Dipartimento di Matematica, Università di Roma “Tor Vergata”
via della Ricerca Scientifica, 00133 Roma, Italy
giammarr@mat.uniroma2.it

Abstract. Tiling recognizable two-dimensional languages, also known as REC, generalize recognizable string languages to two dimensions and share with them several theoretical properties. Nevertheless REC is not closed under complementation and this implies that it is intrinsically non-deterministic. As result, all subclasses corresponding to different notion of unambiguity and determinism define a hierarchy inside REC. Moreover we show that some definitions of unambiguity are equivalent to corresponding notions of determinism and therefore correspond decidable classes closed under complementation and linear parsing algorithms.

Keywords: Automata and Formal Languages. Two-dimensional languages. Tiling systems. Unambiguity, Determinism.

1 Introduction

Two-dimensional languages (or picture languages) are sets of two-dimensional arrays of symbols chosen in a finite alphabet (called pictures): they generalize to two dimensions (2D for short) the classical string languages. The study of picture languages is highly motivated by pattern recognition and image processing problems as well as by all the applications of tiling problems in math, topology, physics and biology. The first generalization of finite-state automata to two dimensions can be attributed to M. Blum and C. Hewitt who in 1967 introduced the notion of a four-way automaton moving on a two-dimensional tape as the natural extension of a one-dimensional two-way finite automaton (see [5]). The model was not successful since it does not satisfies important properties. Since then, many approaches have been presented in the literature in order to find in 2D a counterpart of what regular languages are in one dimension: finite automata, grammars, logics and regular expressions (see for example [6,10,21,14,23]). In this paper we focus on the family REC of *tiling recognizable picture languages* (see [10,11] that have been widely investigated and that it is considered as a valid candidate to represent a counter part to 2D of regular string languages. The definition of REC takes as starting point a characterization of recognizable string languages in terms of local languages and projections (cf. [9]): the pair of a local picture language and a projection is called *tiling system*.

* This work was partially supported by MIUR Project *“Aspetti matematici e applicazioni emergenti degli automi e dei linguaggi formali”* (2007), by ESF Project *“AutoMathA”* (2005-2010).

The definition of REC in terms of tiling systems turns out to be very robust: in [11,13] it is shown that the family REC has a characterization in terms of logical formulas (a generalization of Büchi's theorem for strings to 2D). Moreover in [15], it is proved that REC have a counterpart as machine models in the *two-dimensional on-line tessellation acceptor (OTA)* introduced by K. Inoue and A. Nakamura in [14]. Other models of automata for REC are proposed in [2,6,20]. Tiling systems can be also simulated by domino systems [15] and Wang tiles [8] and grammars [7]. A crucial difference lies in the fact that the definition of recognizability by tiling systems is intrinsically non-deterministic. Deterministic machine models to recognize two-dimensional languages have been considered in the literature: they always accept classes of languages smaller than the corresponding non-deterministic ones (see for example, [5,14,22]). This seems to be unavoidable when jumping from one to two dimensions. Further REC family is not closed under complementation and therefore the definition of any constraint to force determinism in tiling systems should necessary result in a class smaller than REC. Strictly connected with this problems are the negative results on parsing in REC. The non-determinism implies that when recognizing a picture of m row and n columns, once reached a given position one may eventually backtrack on all positions already visited, that is on $O(mn)$ steps. Moreover in [17] it is proved that the parsing problem for REC languages is NP-complete.

In formal language theory, an intermediate notion between determinism and non-determinism is the notion of unambiguity. In an unambiguous model, we require that each accepted object admits only one successful computation. Both determinism and unambiguity correspond to the existence of a *unique* process of computation, but while determinism is a "local" notion, unambiguity is a fully "global" one. *Unambiguous tiling recognizable* two-dimensional languages have been introduced in [10], and their family is referred to as UREC. Informally, a picture language belongs to UREC when it admits an unambiguous tiling system, that is if every picture has a unique pre-image in its corresponding local language. In [3], the proper inclusion of UREC in REC is proved but it is also proved that it is undecidable whether a given tiling system is unambiguous. From a computational side, this implies that the complexity of the parsing is as in REC i.e. at each step of the computation it can be necessary to backtrack on all already visited positions. As direct consequence UREC cannot be equivalent to *any* deterministic model for REC.

The main goal is then to find subclasses for REC that inherit important properties but also allow feasible computations. Moreover it could be interesting to prove that some notion of unambiguity and determinism coincide also in the 2D settings.

Remark that another difference between unambiguity and determinism is that determinism is always related to a scanning strategy to read the input. In the string case the scanning is implicitly assumed to be left-to right and in fact deterministic automata are defined related to this direction. Moreover since deterministic, non-ambiguous and non-deterministic models are all equivalent there is no need to consider determinism from right-to-left (referred to as co-determinism). Nevertheless it is worthy to remark that not all regular string languages admits automata that are both deterministic and co-deterministic. In the two-dimensional case we have to consider all the scanning directions from left, right, top and bottom sides.

By exploiting the different possibilities of scanning for a two-dimensional array in [1] there are introduced different notions of unambiguity. We consider tiling systems whose parsing computations can have at each position a backtracking on at most $m + n$ steps. Such definitions lie between those of unambiguity and determinism (as long as we consider that a deterministic computation has zero backtracking steps at each position) while they all coincide with determinism when pictures degenerate in strings (i.e. when considering one-row pictures).

The informal definitions are very simple and natural. A tiling system is *column-unambiguous* if, when used to recognize a picture by reading it along a left-to-right or right-to left direction, once computed a local column, there is only one possible next local column. As consequence in a parsing by a column-unambiguous tiling system of a picture with m rows, the backtracking at each step is at most of m steps. Similarly there are defined *row-unambiguous* and *diagonal-unambiguous* tiling systems corresponding to computations that proceed by rows or by diagonals, respectively. The corresponding families of languages are denoted by *Col-UREC*, *Row-UREC* and *Diag-UREC*. In [1] there are proved necessary conditions for a language to be in Col-UREC and in Row-UREC. Using such conditions one can show that families Col-UREC and Row-UREC are strictly contained in UREC. In a different set-up it is also shown that Diag-UREC is strictly included both in Col-UREC and Row-UREC.

Very interestingly we can prove that diagonal-unambiguous tiling systems are equivalent to some deterministic tiling systems where the uniqueness of computation is guaranteed by certain conditions on the set of local tiles: the corresponding language family is denoted by DREC ([1]). Similar results hold for classes Col-UREC and Row-UREC whose union turns to be equivalent to another "deterministic" class named Snake-DREC [19]. All those classes are closed under complementation [1,19]. As result, when we consider this sort of "line" unambiguity we can prove equivalence with deterministic models and therefore we guarantee a parsing linear in the size (i.e. number of rows times number of columns) of the input.

2 Basic Definitions and Notations

We recall some definitions about two-dimensional languages. The notations used and more details can be mainly found in [11].

A *picture* (or a *two-dimensional string*) over a finite alphabet Σ is a two-dimensional rectangular array of elements of Σ. Given a picture $p \in \Sigma^{**}$, let $p_{(i,j)}$ denote the symbol in p with coordinates (i, j), where position $(1, 1)$ corresponds to top-left corner. A sub-picture of p is a rectangular sub-array of p.

Given a picture p, we let $|p|_{row}$ and $|p|_{col}$ denote the number of rows and columns of p, respectively, and let $|p| = (|p|_{row}, |p|_{col})$ denote the picture size. The set of all pictures over Σ of size (m, n) is denoted by $\Sigma^{m,n}$. The set of all pictures over Σ is denoted by Σ^{**}. We remark that set Σ^{**} includes also all the empty pictures, i.e. pictures of sizes $(m, 0)$ or $(0, n)$ for all $m, n \geq 0$. In this paper we will not be interested in empty pictures and therefore we will always consider set Σ^{++}.

A *two-dimensional (picture) language* over an alphabet Σ is a subset of Σ^{++}.

Between pictures and picture languages, there can be defined some kind of concatenation operations. Let p and q be two pictures over an alphabet Σ.

The *column concatenation* of p and q (denoted by $p \obar q$) and the *row concatenation* of p and q (denoted by $p \ominus q$) are partial operations, defined only if $|p|_{row} = |q|_{row}$ and $|p|_{col} = |q|_{col}$, respectively, and are given by:

$$p \obar q = \begin{array}{|c|c|}\hline \; p \; & \; q \; \\ \hline\end{array} \qquad\qquad p \ominus q = \begin{array}{|c|}\hline p \\ \hline q \\ \hline\end{array}.$$

These definitions can be extended in the usual way to picture languages. Given two picture languages L_1, L_2 over an alphabet Σ we can define $L_1 \obar L_2$ and $L_1 \ominus L_2$. Furthermore, by iterating the concatenation operations, we obtain the column and row *closure* or *star*. The *column closure* of a picture language$L$ (denoted by $L^{+\obar}$) and the *row closure* of L (denoted by $L^{+\ominus}$) are defined respectively as

$$L^{+\obar} = \bigcup_i L^{i\,\obar} \qquad \text{and} \qquad L^{+\ominus} = \bigcup_i L^{i\,\ominus}$$

where $L^{1\,\obar} = L$, $L^{n\obar} = L^{(n-1)\obar} \obar L$ and $L^{1\,\ominus} = L$, $L^{n\ominus} = L^{(n-1)\ominus} \ominus L$.

To define a picture language it is often necessary to identify the symbols on the boundary of the pictures. For any picture p with m rows and n columns, we consider the *bordered picture* $\widehat{p}$ obtained by surrounding p with a special *boundary symbol* $\# \notin \Sigma$: positions of $\widehat{p}$ will be indexed in $\{0, 1, \cdots, m+1\} \times \{0, 1, \cdots, n+1\}$.

We refer to the *parsing problem* as the problem of deciding whether a given input picture p belongs to a language L specified in same way. A parsing algorithm is based on a reading of the given picture throughout a recognizer model. As in the string case, some recognizer models can have a sort of built-in scanning strategy while for some others the way to read the picture should be explicitly specified. We introduce for this some notations.

Given a picture, the four sides *left*, *right*, *top* and *bottom* will be referred by the initial letter while the four corners will be referred to as the $tl-$, $tr-$, $bl-$ and *br*-corner, respectively. We assume that a scanning strategy starts at one corner and then explore the picture by following some "direction". We consider all *side-to-side directions* $l2r, r2l, t2b$, and $b2t$. For example, a scanning follows a *l2r-direction*, if we read position (x, y) only after all positions to the its left that is after *all* positions (i, j) with $j < y$. Similarly for all the other side-to-side directions. Another possibility is to put somehow the two dimensions together and consider *corner-to-corner directions*: *tl2br*, *tr2bl*, *bl2tr*, *br2tl*. We say that a scanning of a picture follows a *tl2br-direction* if we can read position (x, y) only after all positions to the top and to the left of (x, y) that is after all positions (i, j) with $i \leq x$ and $j \leq y$. Similarly for the other corner-to-corner directions. Observe that, unlike the 1D case, once fixed a scanning direction there can be different reading paths on the picture p along that direction.

2.1 A Small Collection of Examples

In this section we collects some picture languages that will be mentioned in the paper to better explain the peculiarity of various classes of languages here discussed.

Example 1. Language of squares over a one-letter alphabet $\Sigma = \{a\}$

Example 2. Language on a two-letters alphabet $\Sigma = \{0, 1\}$ of squares whose positions in the main diagonal contain 1 while all other positions contain 0.

Example 3. Language $L_{fc=lc}$ of pictures over $\Sigma = \{a, b\}$, with more than one column, whose first column is equal to the last one.

From this example we can generate some ”variants” as the followings. Obviously all those examples can be rewritten using rows instead of columns.

Example 4. Language $L_{\exists c=fc}$ of pictures whose first column is equal to some other one; language $L_{\exists c=lc}$ language of pictures whose last column is equal to some other one; $L_{\exists c'=c''}$ of pictures that have two equal columns. All languages are defined over $\Sigma = \{a, b\}$ and all pictures have more than one column.

We also mention some examples that are somewhat a two-dimensional version of non-regular string languages. All languages are defined over a two-letters alphabet.

Example 5. Language L_1 of pictures of size $(n, 2n)$ (i.e. as two squares) such that the left halve is equal to the right one.

Example 6. Language L_2 of pictures of size $(n, 2n)$ (i.e. as two squares) such that the left halve contains all as while the right halve contains all bs (each row is a string $a^n b^n$).

Example 7. Language L_3 of pictures such that first row is a palindrome.

2.2 Local Picture Languages

An interesting simple family of picture languages is the one of *local picture languages* (*LOC*) first introduced in [10]. It will be used to define the class REC of recognizable picture languages defined in the next section.

We first introduce the following notation: $[\![p]\!]$ denotes the set all possible sub-pictures of size $(2, 2)$ of the bordered picture $\hat{p}$. In the sequel, a picture of size $(2, 2)$ will be also referred to as a *tile*.

Informally, a picture language is *local* if it can be defined by a finite set of *allowed tiles*. More precisely, let Γ be a finite alphabet.

Definition 1. *A two-dimensional language $L \subseteq \Gamma^{++}$ is* local *if there exists a set Θ of tiles over $\Gamma \cup \{\#\}$ such that $L = \{p \in \Gamma^{++} |\, [\![p]\!] \subseteq \Theta\}$ and we will write $L = L(\Theta)$.*

Notice that this family is a natural generalization to two dimensions of the family of local string languages defined by means of a set of strings of length 2 over an alphabet that includes also border symbols. As simple example, notice that the language of squares over two-letter in Example 2 is a local language since it can be described by the following set of tiles.

$$\Theta = \left\{ \begin{array}{l}
\begin{array}{|c|c|}\hline 1&0\\\hline 0&1\\\hline\end{array},
\begin{array}{|c|c|}\hline 0&0\\\hline 1&0\\\hline\end{array},
\begin{array}{|c|c|}\hline 0&1\\\hline 0&0\\\hline\end{array},
\begin{array}{|c|c|}\hline 0&0\\\hline 0&0\\\hline\end{array},
\begin{array}{|c|c|}\hline \#&1\\\hline \#&0\\\hline\end{array},
\begin{array}{|c|c|}\hline \#&0\\\hline \#&0\\\hline\end{array},
\begin{array}{|c|c|}\hline 0&0\\\hline \#&\#\\\hline\end{array},
\begin{array}{|c|c|}\hline 0&1\\\hline \#&\#\\\hline\end{array},
\\[2ex]
\begin{array}{|c|c|}\hline 0&\#\\\hline 1&\#\\\hline\end{array},
\begin{array}{|c|c|}\hline 0&\#\\\hline 0&\#\\\hline\end{array},
\begin{array}{|c|c|}\hline \#&\#\\\hline 0&0\\\hline\end{array},
\begin{array}{|c|c|}\hline \#&\#\\\hline 1&0\\\hline\end{array},
\begin{array}{|c|c|}\hline \#&\#\\\hline \#&1\\\hline\end{array},
\begin{array}{|c|c|}\hline \#&\#\\\hline 0&\#\\\hline\end{array},
\begin{array}{|c|c|}\hline \#&0\\\hline \#&\#\\\hline\end{array},
\begin{array}{|c|c|}\hline 1&\#\\\hline \#&\#\\\hline\end{array}
\end{array} \right\}$$

Notice that we exploit the 1s in the diagonal to verify the square shapes of the pictures and in fact it is easy to understand that language in Example 1 of squares with only one symbol is not a local language.

We remark that given a local language L (by means of a finite set of tiles Θ), one can do the parsing of a picture p of size (m, n) in time linear in mn. A way to do this computation is to choose any scanning strategy (for example choose the t2b-direction, and read p from the tl-corner by proceeding row by row reading each row from left to right, till reaching the br-corner). At each step of the scanning, say reading position (i, j) in p, check that the sub-picture $\begin{array}{|c|c|}\hline p_{(i,j)} & p_{(i+1,j)} \\ \hline p_{(i,j+1)} & p_{(i+1,j+1)} \\ \hline \end{array}$ belongs to the set Θ of allowed tiles.

3 Tiling Recognizable Picture Languages

We now define tiling recognizable picture languages by means of local languages and projection. We recall first the notion of *projection of a language*. Let Γ and Σ be two finite alphabets and let $\pi : \Gamma \to \Sigma$ be a mapping we call *projection*. In the usual way we extend projections to pictures and to picture languages over Γ. If $p = \pi(p')$ we also refer to p' as a *pre-image* of p. Similarly for languages.

Definition 2. *A* tiling system (TS) *is a quadruple* $(\Sigma, \Gamma, \Theta, \pi)$ *where* Σ *and* Γ *are finite alphabets,* Θ *is a set of tiles over* $\Gamma \cup \{\#\}$ *and* $\pi : \Gamma \to \Sigma$ *is a projection.*

Therefore, a tiling system is composed by a local language over a given alphabet Γ and a projection $\pi : \Gamma \to \Sigma$.

Definition 3. *A two-dimensional language* $L \subseteq \Sigma^{**}$ *is recognized by a* tiling system $(\Sigma, \Gamma, \Theta, \pi)$ *if* $L = \pi(L(\Theta))$. *The family of all* tiling recognizable *picture languages is denoted by REC.*

Tiling systems $(\Sigma, \Gamma, \Theta, \pi)$ for picture languages are in some sense a generalization of automata for string languages. Indeed, in the one-dimensional case, the quadruple $(\Sigma, \Gamma, \Theta, \pi)$ corresponds exactly to the graph of the automaton: Γ represents the edges set, Θ describes the edges adjacency while π gives the edges labels. A word of the local language defined by Θ corresponds to an accepting path in the graph and its projection by π gives the actual word recognized by the automaton (cf. [9]). Then, when rectangles degenerate in strings the definition of recognizability coincides with the classical one for strings. Moreover observe that the definition of recognizability in terms of tiling systems is implicitly non-deterministic. In fact, by referring to the one-dimensional case, if no particular constraints are given for the set Θ, the tiling system $\mathcal{T} = (\Sigma, \Gamma, \Theta, \pi)$ corresponds in general to a non-deterministic automaton.

Let us give first two simple examples.

Example 8. Let L be the language of squares over one letter alphabet $\Sigma = a$ as in Example 1. Then L is recognizable by a tiling system that have as local language the language in Example 2 and a projection π that maps both 0 and 1 to letter a.

Example 9. Let $L_{fc=lc}$ be the language of pictures over $\Sigma = \{a, b\}$ whose the first column is equal to the last one. Language $L_{fc=lc} \in REC$. Informally we can define a local language where information about first column symbols of a picture p is brought along horizontal direction, by means of subscripts, to match the last column of p. Tiles are defined to have always same subscripts within a row while, in the right-border tiles, subscripts and main symbols should match. Here are some left border, right border and middle tiles, respectively: $\begin{array}{|c|c|}\hline \# & z_z \\ \hline \# & t_t \\ \hline\end{array}$, $\begin{array}{|c|c|}\hline z_z & \# \\ \hline t_t & \# \\ \hline\end{array}$, and $\begin{array}{|c|c|}\hline z_z & s_z \\ \hline t_t & r_t \\ \hline\end{array}$ with $r, s, z, t \in \Sigma$. Below it is an example of a picture $p \in L_{fc=lc}$ together with a corresponding local picture p'.

$$p = \begin{array}{|c|c|c|c|c|}\hline b & b & a & b & b \\ \hline a & a & b & a & a \\ \hline b & a & a & a & b \\ \hline a & b & b & b & a \\ \hline\end{array} \quad p' = \begin{array}{|c|c|c|c|c|}\hline b_b & b_b & a_b & b_b & b_b \\ \hline a_a & a_a & b_a & a_a & a_a \\ \hline b_b & a_b & a_b & a_b & b_b \\ \hline a_a & b_a & b_a & b_a & a_a \\ \hline\end{array}.$$

The definition of REC in terms of tiling systems turns out to be very robust: in [11,13] it is shown that the family REC and the family of languages defined by existential monadic second order formulas coincide. And this is actually the generalization of Büchi's theorem for strings to two-dimensional languages. Moreover in [15], it is proved that REC have a counterpart as machine models in the *two-dimensional on-line tessellation acceptor (OTA)* introduced by K. Inoue and A. Nakamura in [14]. Other models of automata for REC are proposed in [2,6,20]. Tiling systems can be also simulated by domino systems [15] and Wang tiles [8] and grammars [7].

Furthermore, the family REC is closed with respect to different types of operations (see [11] for all the proofs): row and column concatenation and their closures, union, intersection and rotation. As immediate application, consider the following example.

Example 10. All languages in Example 4 are in REC: they can be obtained as:
$L_{\exists c=fc} = L_{fc=lc} \obar \Sigma^{**}$
$L_{\exists c=lc} = \Sigma^{**} \obar L_{fc=lc}$
$L_{\exists c=c'} = \Sigma^{**} \obar L_{fc=lc} \obar \Sigma^{**}$.

It is interesting to remark that REC contains also some languages whose one-dimensional version is non regular such as languages in Examples 6 and 7.

All those properties confirm the close analogy with the one-dimensional case. The big difference with regular string languages regards the complementation operation. In [11] it is shown that the language in Example 5 does not belong to REC while its complement does. As result family REC is *not* closed under complementation and this suggests that it is not possible to eliminate non-determinism from REC family. Strictly connected with this problems are the negative results on parsing in REC.

The parsing problem in REC. Let L be a language specified by means of a tiling system. To check whether a given input picture p belongs to L we have to find one of the pre-images of p. To do this, we start a computation that takes p and rewrites symbols in p in the local alphabet, in a way that is compatible with the projection π and with the allowed set of local tiles. The process is accomplished following some scanning strategy and ends when all symbols in p are rewritten in the local alphabet. Observe that a tiling system do not force to any particular scanning strategy to read the picture.

Moreover note that, as already remarked, such computation process is in general non-deterministic: at each step there can be several choices; when there is no possible tile matching the next position, one can go back on already scanned positions, till the last choice and replace it with another one. This means that during the computation of a picture of size (m, n), to rewrite a given position one may eventually backtracks on all positions where tiles were already placed, that is on $O(mn)$ steps.

It can be proved that the parsing problem for REC languages is NP-complete (see [17]) where TS are called *homomorphisms of local lattice languages*.

A natural step forward in the investigation of REC family is therefore to consider subclasses where the computation is feasible. We always pretend that, when pictures have only one row, all the definitions coincide with corresponding ones in the strings case. We consider the notions of unambiguity and determinism: recall that unambiguity prescribes only one accepting computation while determinism admits only one possible next "move" at each step of the computation. In some sense the "uniqueness" is required globally for unambiguity while it should be also local for determinism. Notice that the notion of determinism always implies a reading strategy for the input (in the string case is implicitly assumed from left to right). Recall that for string languages deterministic, unambiguous and non-deterministic versions of the computational model correspond to the same class of languages, namely recognizable languages (this also assures that a deterministic class defined along a right-to-left reading direction would also coincide with the same class). As discussed before, because of non-closure of REC under complementation we cannot expect same results in two dimensions but we can still hope in an equivalence between unambiguity and determinism. Let us consider first the notion of unambiguity.

3.1 Unambiguous Recognizable Two-Dimensional Languages

The definition of unambiguous recognizable two-dimensional language was first given in [10]. Informally, a tiling system is unambiguous if every picture has a unique local pre-image.

Definition 4. *A tiling system* $(\Sigma, \Gamma, \Theta, \pi)$ *is an* unambiguous tiling system *for a language* $L \subseteq \Sigma^{++}$ *if and only if for any picture* $p \in L$ *there exists a* unique *local picture* $p' \in L(\Theta)$ *such that* $p = \pi(p')$.

A two-dimensional language $L \subseteq \Sigma^{**}$ is *unambiguous* if and only if it admits an unambiguous tiling system and denote by UREC the family of all unambiguous recognizable two-dimensional languages. Obviously UREC $\subseteq$ REC.

It can be verified that the TS for the language given in Example 9 is unambiguous and therefore $L_{fc=lc}$ belongs to UREC. In [3,12] it is proved a necessary condition for a language to be in UREC. Then, using such condition it is shown that language $L_{\exists c=c'}$ given in Example 4 does not belong to UREC and therefore it is proved that family UREC is strictly contained in REC. In [4] it is shown that the strict inclusion holds even in the case of unary alphabet. Remark that from the fact that language $L_{\exists c=c'}$ we infer also that UREC is not closed under column-concatenation. In [3] it is proved that UREC is not closed under row and column concatenations and star operations while it is closed under rotation and intersection operations. The closure under complementation of UREC is still an open problem. From a computational side, the parsing for a picture of size (m, n) seems to be still exponential in mn since at each step of the computation it could be necessary to backtrack on all already visited positions.

In [3] it is also proved that it is undecidable whether a given tiling system is unambiguous. As direct consequence class UREC cannot be equivalent to any deterministic model for REC.

4 Some Results on Unambiguity-Determinism Equivalences

In [1] there are introduced different notions of unambiguity by exploiting the different possibilities of scanning for a two-dimensional array. We consider tiling systems whose parsing computations can have at each position a backtracking on at most $O(m + n)$ steps. Such definitions lie between those of unambiguity and determinism (as long as we consider that a deterministic computation has zero backtracking steps at each position) while they all coincide with determinism when pictures degenerate in strings (i.e. when considering one-row pictures).

4.1 Column-, Row- and Diagonal-Unambiguous Languages

The informal definitions are very simple and natural. A tiling system is *l2r-unambiguous* if, when used to recognize a picture by reading it along a l2r direction, once computed a local column, there is only one possible next local column. Recall from Section 2 that a strategy that follows a l2r directions visits position (x, y) only after all positions (i, j) with $j < y$ and therefore it does a sort of column-by-column visit. Formally we have.

Definition 5. *A tiling system* $(\Sigma, \Gamma, \Theta, \pi)$ *is* l2r-unambiguous *if for any column* $col' \in \Gamma^{m,1} \cup \{\#\}^{m,1}$*, and picture* $p \in \Sigma^{m,1}$*, there exists at most one local column* $col'' \in \Gamma^{m,1}$*, such that* $\pi(col'') = p$ *and* $[\![p']\!] \subseteq \Theta$ *where* $p' = \{\#\}^{1,2} \ominus (col' \oplus col'') \ominus \{\#\}^{1,2}$.

The $r2l$-unambiguity is defined likewise. A tiling system is *column-unambiguous* if it is d-unambiguous $d \in \{l2r, r2l\}$ and that a language is *column-unambiguous* if it is recognized by a column-unambiguous tiling system. Finally, we denote by *Col-UREC* the class of column-unambiguous languages. Remark that, a column-unambiguous tiling system is such that, during the parsing of a picture of size (m, n), the backtracking at each step is at most m. This is because the next local column is uniquely determined without ambiguity after backtracking of m steps at most.

Similarly, define d-unambiguous tiling system for $d \in \{t2b, b2t\}$ that bring to definition of *row-unambiguous* tiling systems and languages and of class *Row-UREC*. For reasonings similar as before, during the parsing of a picture of size (m, n) with a row-unambiguous tiling system the backtracking at each step is at most n (i.e. inside the current row).

We consider also corner-to-corner directions. We say that a tiling system is *tl2br-unambiguous* if, when used to recognize a picture by reading it along a tl2br direction, once computed a local diagonal (for this direction it is a counter-diagonal), there is only one possible next local diagonal. The formal definition can be given following the footsteps of Definition 5. In fact, consider a set $X = \{x_{(i,j)} \in \Sigma \mid (i, j) \in I\}$ then if we take the indices set $I = \{(i,k) \mid i = 0,1,\ldots,m\}$ then X is a column over Σ while taking $I = \{(k,i) \mid i = 0,1,\ldots,n\}$ then X is a row over Σ. Moreover if we take $I = \{(i,k-i) \mid i = 0,1,\ldots,k\}$ then X is a counter-diagonal over Σ. Then a formal definition should only provide technical conditions for verifying that the kth and the $(k+1)$th diagonals are "coherent" with the given set of local tiles.

Similar properties define d-unambiguous tiling systems, for any corner-to-corner direction d that correspond all together to definition of *diagonal-unambiguous* languages and class *Diag-UREC*. For reasonings similar as before, during the parsing of a picture of size (m,n) with a diagonal-unambiguous tiling systems it is at most of $min(m,n)$ steps (i.e. inside the current diagonal).

In [1] there are proved necessary conditions for a language to be in Col-UREC and in Row-UREC. Using such conditions one can show that families Col-UREC and Row-UREC are strictly contained in UREC. In a different set-up it is also shown that Diag-UREC is strictly included both in Col-UREC and Row-UREC.

4.2 Deterministic Tiling Recognizable Languages

Recall that in a deterministic TS it should be guaranteed the uniqueness of each step in a computation process; moreover, as in the string case, notion of determinism need to be related to a particular scanning direction. We consider the corner-to corner directions and give the following formal definition.

Definition 6. *A tiling system* $(\Sigma,\Gamma,\Theta,\pi)$ *is* tl2br-deterministic *if for any* γ_1, γ_2, $\gamma_3 \in \Gamma \cup \{\#\}$ *and* $\sigma \in \Sigma$ *there exists at most one tile* $\begin{array}{|c|c|}\hline \gamma_1 & \gamma_2 \\ \hline \gamma_3 & \gamma_4 \\ \hline \end{array} \in \Theta$*, with* $\pi(\gamma_4) = \sigma$.

Similarly we define d-deterministic tiling systems for any corner-to-corner direction d. A recognizable two-dimensional language L is *deterministic*, if it admits a d-deterministic tiling system for some corner-to-corner direction d. We denote by *DREC*, the class of *Deterministic Recognizable Two-dimensional Languages*.

It is easy to show [1] that it is decidable whether a given tiling system is d-deterministic for some direction d. Moreover in [1] it is proved that, as one would expect in a deterministic setting, class DREC is closed under complementation.

If we closely observe definition of diagonal unambiguity, we notice that local symbol in a position on the diagonal does not depend on the other local symbols on the same diagonal. Then, by techniques as in [15], it can be proved the following theorem.

Theorem 1. *DREC = Diag-UREC*

Recently in [19] it was given a new definition of determinism for TS called *snake-determinism*. For our convenience we rename it here as *t2b-snake-determinism*. It is in fact based on a scanning strategy along the side-to-side t2b-direction that starts from the tl-corner, scans the first row rightwards, then scans the second row leftwards, and so on. This means that we scan odd rows rightwards and even row leftwards, assigning a symbol from the local alphabet to each position. Moreover tiles for the odd and for the even rows are tl2br- and tr2bl- deterministic, respectively.

Definition 7. *A tiling system* $(\Sigma, \Gamma, \Theta, \pi)$ *is* t2b-snake-deterministic *if* Σ, Γ *can be partitioned as* $\Gamma = \Gamma_1 \cup \Gamma_2$, $\Theta = \Theta_1 \cup \Theta_2$*, where*

- $(\Sigma, \Gamma, \Theta_1, \pi)$ *is tl2br-deterministic and* Θ_1 *contains only tiles like* $\begin{array}{|c|c|}\hline a_2 & b_2 \\ \hline a_1 & b_1 \\ \hline\end{array}$ *with* $a_i, b_i \in \Gamma_i \cup \{\#\}$ *for* $i = 1, 2$*;*
- $(\Sigma, \Gamma, \Theta_2, \pi)$ *is tr2bl-deterministic and* Θ_2 *contains only tiles like* $\begin{array}{|c|c|}\hline a_1 & b_1 \\ \hline a_2 & b_2 \\ \hline\end{array}$ *with* $a_i, b_i \in \Gamma_i \cup \{\#\}$ *for* $i = 1, 2$ *and* $(a_1, b_1) \neq (\#, \#)$*;*

In [19] it is proved the following theorem.

Theorem 2. *The class of languages recognized by t2b-snake-deterministic TS and the class of languages recognized by t2b-unambiguous TS coincide.*

Analogous definitions can be given for b2t-, l2r-. r2l-snake-determinism and prove corresponding versions of Theorem 2. A recognizable two-dimensional language L is *snake-deterministic*, if it admits a d-snake-deterministic tiling system for some side-to-side direction d. We denote by *snake-DREC*, the class of *Snake Deterministic TS Recognizable Two-dimensional Languages*. (Equivalently: snake-DREC is the closure under rotation of class of languages recognized by t2b-snake-deterministic TS). In [19] it is proved that the snake-DREC is closed under complementation and that it holds the following theorem, analogous of Theorem 1 for the row- and column-unambiguity.

Theorem 3. *Snake-DREC= Row-Urec* $\cup$ *Col-UREC.*

Observe that Theorems 1 and 3 somehow extend to two dimensions the equivalence between determinism and unambiguity for regular string languages.

Summarizing all the mentioned results we have the following relations that show the more complex situations we find when we go from one to two dimensions.

Theorem 4. *The following inclusion relations are all strict:*
DREC=Diag-UREC $\subset$ *(Col-UREC* $\cup$ *Row-UREC)=Snake-DREC* $\subset$ *UREC* $\subset$ *REC.*

Acknowledgments

I am grateful to my friends and co-authors Marcella Anselmo, Maria Madonia and Antonio Restivo for all the results discussed in this paper.

References

1. Anselmo, M., Giammarresi, D., Madonia, M.: From determinism to non-determinism in recognizable two-dimensional languages. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 36–47. Springer, Heidelberg (2007)
2. Anselmo, M., Giammarresi, D., Madonia, M.: A computational model for tiling recognizable two-dimensional languages. Theoretical Computer Science 410(37), 3520–3529 (2009)
3. Anselmo, M., Giammarresi, D., Madonia, M., Restivo, A.: Unambiguous Recognizable Two-dimensional Languages. RAIRO: Theoretical Informatics and Applications 40(2), 227–294 (2006)

4. Anselmo, M., Madonia, M.: Deterministic and unambiguous two-dimensional languages over one-letter alphabet. Theoretical Computer Science 410(16), 1477–1485 (2009)
5. Blum, M., Hewitt, C.: Automata on a two-dimensional tape. In: IEEE Symposium on Switching and Automata Theory, pp. 155–160 (1967)
6. Bozapalidis, S., Grammatikopoulou, A.: Recognizable picture series. Journal of Automata, Languages and Combinatorics, Special vol. on Weighted Automata (2004)
7. Crespi Reghizzi, S., Pradella, M.: Tile rewriting grammars and picture languages. Theoretical Computer Science 340(2), 257–272 (2005)
8. De Prophetis, L., Varricchio, S.: Recognizability of rectangular pictures by wang systems. Journal of Automata, Languages, Combinatorics 2, 269–288 (1997)
9. Eilenberg, S.: Automata, Languages and Machines, vol. A. Academic Press, London (1974)
10. Giammarresi, D., Restivo, A.: Recognizable picture languages. Int. Journal Pattern Recognition and Artificial Intelligence 6(2,3), 241–256 (1992)
11. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Rozenberg, G., et al. (eds.) Handbook of Formal Languages, vol. III, pp. 215–268. Springer, Heidelberg (1997)
12. Giammarresi, D., Restivo, A.: Matrix-based complexity functions and recognizable picture languages. In: Grader, E., Flum, J., Wilke, T. (eds.) Logic and Automata: History and Perspectives. Texts in Logic and Games, vol. 2, pp. 315–337. Amsterdam University Press (2007)
13. Giammarresi, D., Restivo, A., Seibert, S., Thomas, W.: Monadic second order logic over pictures and recognizability by tiling systems. Information and Computation 125(1), 32–45 (1996)
14. Inoue, K., Nakamura, A.: Some properties of two-dimensional on-line tessellation acceptors. Information Sciences 13, 95–121 (1977)
15. Inoue, K., Takanami, I.: A characterization of recognizable picture languages. In: Nakamura, A., Saoudi, A., Inoue, K., Wang, P.S.P., Nivat, M. (eds.) ICPIA 1992. LNCS, vol. 654. Springer, Heidelberg (1992)
16. Latteux, M., Simplot, D.: Recognizable Picture Languages and Domino Tiling. Theorethical Computer Science 178(1-2), 275–283 (1997)
17. Lindgren, K., Moore, C., Nordahl, M.: Complexity of two-dimensional patterns. Journal of Statistical Physics 91(5-6), 909–951 (1998)
18. Matz, O.: Regular expressions and Context-free Grammars for picture languages. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 283–294. Springer, Heidelberg (1997)
19. Lonati, V., Pradella, M.: Snake-Deterministic Tiling Systems. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 549–560. Springer, Heidelberg (2009)
20. Lonati, V., Pradella, M.: Picture-recognizability with automata based on wang tiles. In: van Leeuwen, J. (ed.) SOFSEM 2010. LNCS, vol. 5901, pp. 576–587. Springer, Heidelberg (2010)
21. Matz, O.: On piecewise testable, starfree, and recognizable picture languages. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378, p. 203. Springer, Heidelberg (1998)
22. Potthoff, A., Seibert, S., Thomas, W.: Nondeterminism versus determinism of finite automata over directed acyclic graphs. Bull. Belgian Math. Soc. 1, 285–298 (1994)
23. Siromoney, R.: Advances in array languages. In: Ehrig, H., Nagl, M., Rosenfeld, A., Rozenberg, G. (eds.) Graph Grammars 1986. LNCS, vol. 291, pp. 549–563. Springer, Heidelberg (1987)

The Complexity of Regular(-Like) Expressions

Markus Holzer and Martin Kutrib

Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{holzer,kutrib}@informatik.uni-giessen.de

Abstract. We summarize results on the complexity of regular(-like) expressions and tour a fragment of the literature. In particular we focus on the descriptional complexity of the conversion of regular expressions to equivalent finite automata and *vice versa*, to the computational complexity of problems on regular-like expressions such as, e.g., membership, inequivalence, and non-emptiness of complement, and finally on the operation problem measuring the required size for transforming expressions with additional language operations (built-in or not) into equivalent ordinary regular expressions.

1 Introduction

Regular expressions were originally introduced in [42] and allow a lovely set-theoretic characterization of languages accepted by deterministic (DFA) or nondeterministic finite automata (NFA). The *regular expressions* over an alphabet Σ and the languages they describe are defined inductively in the usual way:[1] $\emptyset$, λ, and every letter a with $a \in \Sigma$ is a regular expression, and when s and t are regular expressions, then $(s \cup t)$, $(s \cdot t)$, and $(s)^*$ are also regular expressions. The language defined by a regular expression r, denoted by $L(r)$, is defined as follows: $L(\emptyset) = \emptyset$, $L(\lambda) = \{\lambda\}$, $L(a) = \{a\}$, $L(s \cup t) = L(s) \cup L(t)$, $L(s \cdot t) = L(s) \cdot L(t)$, and $L(s^*) = L(s)^*$. Compared to automata, regular expressions are better suited for human users and therefore are often used as interfaces to specify certain pattern or languages. For example, in the widely available programming environment Unix, regular(-like) expressions can be found in legion of software tools like, e.g., `awk`, `ed`, `emacs`, `egrep`, `lex`, `sed`, `vi`, etc., to mention a few of them. The syntax used to represent them may vary, but the concepts are very much the same everywhere.

Since the regular languages are closed under several more operations, the approach to add operations like intersection ($\cap$), complementation ($\sim$), or squaring (^{2}) does not increase the expressive power of regular expressions. However, the descriptional power, that is, the succinctness of such *regular-like* expressions

[1] For convenience, parentheses in regular expressions are sometimes omitted and the concatenation is simply written as juxtaposition. The priority of operators is specified in the usual fashion: concatenation is performed before union, and star before both product and union.

can be increased. On the other hand, growing succinctness of the expressions can increase the computational complexity of decision problems. This motivates the investigation of regular and regular-like expressions (even with a set of operations capturing subregular language families only). In general $\mathrm{RE}(\Sigma, \varphi)$, where φ is a set of (regularity preserving) operations, denotes all regular(-like) expressions over alphabet Σ using only operations from φ. Hence $\mathrm{RE}(\Sigma, \{\cup, \cdot, {}^*\})$ refers to the set of all *ordinary* regular expressions, which is sometimes also denoted by $\mathrm{RE}(\Sigma)$ or simply RE, while, for example, $\mathrm{RE}(\Sigma, \{\cup, \cdot, \sim\})$ defines the star-free languages. This gives rise to our tour on the subjects listed in the abstract of problems related to the descriptional and computational complexity of regular-like expressions. It obviously lacks completeness and we give our view of what constitute the most recent interesting links to the considered problem areas.

There is no general agreement in the literature about the proper measure for regular expressions. We summarize some important ones: The measure size is defined to be the total number of symbols (including $\emptyset$, λ, alphabetic symbols from alphabet Σ, all operation symbols, and parentheses) of a completely bracketed regular expression (for example, used in [1], where it is called length). Another measure related to the reverse polish notation of a regular expression is rpn, which gives the number of nodes in the syntax tree of the expressions (parentheses are not counted). This measure is equal to the length of a (parenthesis-free) expression in postfix notation [1]. The alphabetic width awidth is the total number of alphabetic symbols from Σ (counted with multiplicity) [14,48]. Relations between these measures have been studied, e.g., in [14,15,38,24].

Theorem 1. *Let L be a regular language. Then*

1. $\mathsf{size}(L) \leq 3 \cdot \mathsf{rpn}(L)$ *and* $\mathsf{size}(L) \leq 8 \cdot \mathsf{awidth}(L) - 3$,
2. $\mathsf{awidth}(L) \leq \frac{1}{2} \cdot (\mathsf{size}(L) + 1)$ *and* $\mathsf{awidth}(L) \leq \frac{1}{2} \cdot (\mathsf{rpn}(L) + 1)$, *and*
3. $\mathsf{rpn}(L) \leq \frac{1}{2} \cdot (\mathsf{size}(L) + 1)$ *and* $\mathsf{rpn}(L) \leq 4 \cdot \mathsf{awidth}(L) - 1$,

where $\mathsf{size}(L)$ ($\mathsf{rpn}(L)$, $\mathsf{awidth}(L)$, respectively) refers to the minimum size (rpn, awidth, respectively) among all ordinary *regular expressions denoting L.*

Further not so well known measures for the complexity of regular expressions can be found in [7,14,15,29]. To our knowledge, these latter measures received far less attention to date.

2 Descriptional Complexity of Regular Expressions

The equivalence of the expressive power of regular expressions and finite automata dates back to [42]. Originally, Kleene's proof involved McCulloch-Pitts nerve nets [45], which are a precursor of the concept of finite automata. An elementary construction given in [50] shows that an n-state DFA over a binary alphabet can be simulated by a McCulloch-Pitts nerve net with $2n + 1$ neurons, and by a counting argument that at least $\Omega((n \cdot \log n)^{1/3})$ neurons are required in the worst case [4] to simulate an n-state DFA. The upper bound was further improved to $O(\sqrt{n})$ in [34,39], which was shown to be tight under some additional constraints. After this little detour on McCulloch-Pitts nerve nets now let us focus on the direct transformation of regular expressions into equivalent finite automata.

2.1 From Regular Expressions to Finite Automata

A classical way to transform regular expressions into equivalent finite automata is Thompson's construction [59], which builds NFAs for trivial regular expressions and then composes them into more complex NFAs by connecting these automata by λ-transitions in parallel for union, in sequence for concatenation, and in an iterative fashion for Kleene star. This yields an NFA with λ-transitions (λ-NFA) with a linear number of transitions. Thompson's classical construction went through several stages of optimization (cf. [38]). After a preliminary result in [38] and a tight bound in terms of reverse polish notation from [30] also a tight bound in terms of alphabetic width was found in [24]. We summarize the results from [24,30,38] in the next theorem—here size of a automaton refers to the sum of the number of states and the number transitions:

Theorem 2. *Let $n \geq 1$, and r be a regular expression of alphabetic width n. Then size $\frac{22}{5}n$ is sufficient for an equivalent λ-NFA accepting $L(r)$. In terms of reverse polish length, the bound is $\frac{22}{15}(\mathit{rpn}(r)+1)+1$. Furthermore, there are infinitely many languages for which both bounds are tight.*

The aid for the tight bound in terms of the alphabetic width stated in the previous theorem is a certain normal form for regular expressions, which is a refinement of the *star normal form* from [8] and reads as follows—transformation into strong star normal form preserves the described language, and is weakly monotone with respect to all usual size measures:

Definition 3. *The operators $\circ$ and $\bullet$ are defined on regular expressions*[2] *over alphabet Σ. The first operator is given by: $a^\circ = a$, for $a \in \Sigma$, $(r \cup s)^\circ = r^\circ \cup s^\circ$, $r^{?\circ} = r^\circ$, $r^{*\circ} = r^\circ$; finally, $(rs)^\circ = rs$, if $\lambda \notin L(rs)$ and $r^\circ \cup s^\circ$ otherwise. The second operator is given by: $a^\bullet = a$, for $a \in \Sigma$, $(r \cup s)^\bullet = r^\bullet \cup s^\bullet$, $(rs)^\bullet = r^\bullet s^\bullet$, $r^{*\bullet} = r^{\bullet\circ *}$; finally, $r^{?\bullet} = r^\bullet$, if $\lambda \in L(r)$ and $r^{?\bullet} = r^{\bullet ?}$ otherwise. The* strong star normal form *of an expression r is then defined as $r^\bullet$.*

What about the transformation if λ-transitions are not allowed? One way to obtain such an NFA directly is to construct the *position automaton*, or *Glushkov automaton* [22]. Intuitively, the states of this automaton correspond to the alphabetic symbols or, more precisely, to positions between subsequent alphabetic symbols in the regular expression. An accessible advantage of this construction is given, e.g., in [2,6]: for a regular expression of alphabetic width $n \geq 0$, the position automaton always has precisely $n+1$ states. Simple examples, such as the singleton set $\{a^n\}$, show that this bound is tight. Nevertheless, several optimizations give NFAs having often a smaller number of states, while the underlying constructions are mathematically sound refinements of the basic construction. A structural comparison of the position automaton with its refined versions, namely

[2] Since $\emptyset$ is only needed to denote the empty set, and the need for λ can be substituted by the operator $L^? = L \cup \{\lambda\}$, an alternative is to introduce also the $^?$-operator and instead forbid the use of $\emptyset$ and λ inside non-atomic expressions. This is sometimes more convenient, since one avoids unnecessary redundancy already at the syntactic level [24].

the so-called *follow automaton* [38] and the *equation automaton*, or *Antimirov automaton* [5], is given in [3,11].

Despite the mentioned optimizations, all of these constructions share the same doom on a large number of transitions. An easy upper bound on the number of transitions in the position automaton is $O(n^2)$, independent of alphabet size. It is not hard to prove that the position automaton for the regular expression

$$r_n = (a_1 \cup \lambda) \cdot (a_2 \cup \lambda) \cdots (a_n \cup \lambda)$$

has $\Omega(n^2)$ transitions. It appears to be difficult to avoid such a quadratic blow-up in actual size if we stick to the NFA model. Also if we transform the expression first into a λ-NFA and perform the standard algorithm for removing λ-transitions, see, e.g., [33], we obtain no better result. This naturally raises the question of comparing the descriptional complexity of NFAs over regular expressions. For about 40 years, it appears to have been considered as an unproven factoid that a quadratic number of transitions will be inherently necessary in the worst case (cf. [35]). A barely superlinear lower bound of $\Omega(n \log n)$ on the size of any NFA accepting the language of the expression r_n was proved [35]. More interestingly, the main result of that paper is an algorithm transforming a regular expression of size n into an equivalent NFA with at most $O(n \cdot (\log n)^2)$ transitions. In fact, this upper bound made their lower bound look reasonable at once! Shortly thereafter, an efficient implementation of that conversion algorithm was found [31], and the lower bound was improved in [44] to $\Omega(n \cdot \frac{(\log n)^2}{\log \log n})$. Later work [56] established that any NFA accepting language $L(r_n)$ indeed must have at least $\Omega(n \cdot (\log n)^2)$ transitions. So the upper bound of $O(n \cdot (\log n)^2)$ from [35] is asymptotically tight:

Theorem 4. *Let $n \geq 1$ and r be a regular expression of alphabetic width n. Then size $n \cdot (\log n)^2$ is sufficient for an equivalent NFA to accept $L(r)$. Furthermore, there are infinitely many languages for which this bound is tight.*

Notice that the example witnessing the lower bound is over an alphabet of growing size. For alphabets of size two, the upper bound was improved first [17] to $O(n \cdot \log n)$, and then even to $n \cdot 2^{O(\log^* n)}$, where $\log^*$ denotes the iterated binary logarithm [56]. Thus the question, posed in [36], whether a conversion from regular expressions over a binary alphabet into NFAs of linear size is possible, is almost settled by now.

Theorem 5. *Let $n \geq 1$ and r be a regular expression of alphabetic width n over the binary alphabet. Then size $n \cdot 2^{O(\log^* n)}$ is sufficient for an equivalent NFA to accept $L(r)$.*

Finally, we briefly discuss the problem of converting regular expressions into equivalent DFAs. Again, this problem has been studied by many authors. A taxonomy comparing many different conversion algorithms is given in [60]. Regarding the descriptional complexity, a tight bound of $2^n + 1$ in terms of alphabetic width is already given in [43]. The mentioned work also establishes a matching lower bound, but for a rather nonstandard definition of size. In terms of alphabetic width, the best lower bound known to date is from [15]. Together, we have the following result:

Theorem 6. *Let $n \geq 1$ and r be a regular expression of alphabetic width n over a binary alphabet. Then size $2^n + 1$ is sufficient for a DFA to accept $L(r)$. In contrast, for infinitely many n there are regular expressions r_n of alphabetic width n over a binary alphabet, such that the minimal DFA accepting $L(r_n)$ has at least $\frac{5}{4}2^{\frac{n}{2}}$ states.*

2.2 From Finite Automata to Regular Expressions

There are a few classical algorithms for converting finite automata into regular expressions, which look different at first glance. A comparison of the *state elimination algorithm* [9], *algorithms based on Arden's lemma* [12], or *McNaughton-Yamada's algorithm* [48], is given in [55]. There it is pointed out, that all of these approaches are more or less reformulations of the same underlying algorithmic idea, and can be recast as variations of the standard state elimination algorithm. It produces a regular expression of alphabetic width at most $|\Sigma| \cdot 4^n$, where n is the number of states of the finite automaton. This exponential bound is known to be asymptotically tight, because in [14] the existence of languages L_n, for $n \geq 1$, that admit n-state finite automata over an alphabet of size $O(n^2)$, but require regular expression size at least $2^{\Omega(n)}$ was shown. The question whether a comparable size blow-up can also occur for constant alphabet size [15] was settled by two different groups of researchers recently. A lower bound of $2^{\Omega(\sqrt{n/\log n})}$ for the succinctness gap between DFAs and regular expressions over binary alphabets was reported in [20], while a parallel effort [25] resulted in a tight lower bound of $2^{\Omega(n)}$. We thus have the following result:

Theorem 7. *Let $n \geq 1$ and A be an n-state DFA or NFA over an alphabet of size polynomial in n. Then size $2^{\Theta(n)}$ is sufficient and necessary in the worst case for a regular expression describing $L(A)$. This already holds for constant alphabets with at least two letters.*

We remark that the hidden constant in the lower bound obtained for binary alphabets is much smaller compared to the lower bound of $\Omega(2^n)$ previously obtained in [14] for large alphabets. This is no coincidence. Perhaps surprisingly, one can prove an *upper* bound of $o(2^n)$ when given a DFA over a binary alphabet:

Theorem 8. *Let $n \geq 1$ and A be an n-state DFA over a binary alphabet. Then size $O(1.742^n)$ is sufficient for a regular expression describing $L(A)$. Such an expression can be constructed by state elimination in output polynomial time.*

Similar bounds, but with somewhat larger constants in place of 1.742, can be derived for larger alphabets. Moreover, the same holds for NFAs having a comparably low density of transitions. This result is derived using the classical state elimination algorithm. One simply has to choose a good *elimination ordering*, which is the sequence in which the states are eliminated. Finite automata having a simple structure in the underlying undirected graph tend to allow better elimination orderings. Having a relatively low number of transitions, DFAs over binary alphabets are nothing but one specific utterance of this more general phenomenon. The general theorem reads as follows—cycle rank is a structural measure for regular languages proposed in [13], which measures the degree of connectivity of directed graphs.

Theorem 9. *Let $n \geq 1$ and A be an n-state NFA over alphabet Σ, whose underlying directed graph has cycle rank at most r. Then size $|\Sigma| \cdot 4^r \cdot n$ is sufficient for a regular expression describing $L(A)$. Such an expression can be constructed by state elimination.*

For the particular cases of *finite* or *unary* regular languages, the situation is significantly different. Indeed, the case of finite languages was already addressed in the very first paper on the descriptional complexity of regular expressions [14]. They give a specialized conversion algorithm for finite languages, which is different from the state elimination algorithm. Their results imply that every n-state DFA accepting a finite language can be converted into an equivalent regular expression of size $n^{O(\log n)}$. They also provide a lower bound of $n^{\Omega(\log \log n)}$ when using an alphabet of size $O(n^2)$. The challenge of tightening this gap was settled in [29], where a lower bound technique from communication complexity is adapted, which originated in the study of monotone circuit complexity.

Theorem 10. *Let $n \geq 1$ and A be an n-state DFA or NFA over an alphabet Σ of size $n^{O(1)}$. Then size $n^{\Theta(\log n)}$ is sufficient and necessary in the worst case for a regular expression describing $L(A)$. This still holds for constant alphabets with at least two letters.*

The case of unary languages was discussed in [15]. Here the main idea is that one can exploit the simple cycle structure of unary DFAs and of unary NFAs in Chrobak normal form. The main results are summarized in the following theorem.

Theorem 11. *Let $n \geq 1$ and A be an n-state DFA over an alphabet Σ of size $n^{O(1)}$. Then size $\Theta(n)$ is sufficient and necessary in the worst case for a regular expression describing $L(A)$. When considering NFAs, the upper bound changes to $O(n^2)$.*

3 Computational Complexity of Regular-Like Expressions

We discuss the computational complexity of the decision problems *membership* (MEMBER), *inequivalence* (INEQ), *non-empty complement* (NEC), for regular-like expressions over the alphabet Σ and set of operations φ. Algorithms that solve the equivalence problem have been given, for example, in [10,21], where the former also handles regular expressions extended by the operations intersection and complementation.

Key tools for proving lower bounds of computational complexity are reducibility techniques [54]. We say that a set A is reducible to set B if the ability to answer questions about B implies the ability to answer questions about A by effective methods. If the reducibility of A to B can be done efficiently, then the computational complexity of answering the question for A is a lower bound for the computational complexity of answering questions for B. In particular, we consider three reducibilities: $A \leq_m^{\log} B$ if there is a many-one function $f : A \rightarrow B$ in L such that for all x we have $x \in A$ if and only if $f(x) \in B$. Sharper translational results are obtained for refined reductions [49] by defining $A \leq_m^{\text{log-lin}} B$ if there is a many-to-one function $f : A \rightarrow B$ in L and a constant c such that for all x we have $x \in A$ if and only if $f(x) \in B$ and $|f(x)| \leq c \cdot |x| + c$. Additionally, we consider $A \leq_m^{\text{p-lin}} B$ if there is a many-to-one function $f : A \rightarrow B$ in $\mathsf{P} \cap \mathsf{DLINSPACE}$ and a constant c such that for all x we have $x \in A$ if and only if $f(x) \in B$ and $|f(x)| \leq c \cdot |x| + c$.

We start to summarize known results about the membership problem. The tight complexity for $\mathrm{MEMBER}(\Sigma, \{\cup, \cdot, {}^*\})$, i.e., the problem to decide whether a word belongs to the language described by a regular-like expression from the set $\mathrm{RE}(\Sigma, \{\cup, \cdot, {}^*\})$, has been reported in [40]. In [58] the containment of the membership problem for regular-like expressions with complementation and squaring operators in the complexity class P has been derived. Later in [51] it has been shown that the membership problem for the star-free expressions $\mathrm{RE}(\Sigma, \{\cup, \cdot, \sim\})$ is complete in P. This implies the completeness also for the former class of regular expressions. We summarize these results in the following theorem, and since in our settings the symbols of Σ can be encoded by a binary alphabet without changing the complexity, in the sequel we stick with the alphabet $\{0, 1\}$; this is also valid for the other problems under consideration.

Theorem 12. *1.* $\mathrm{MEMBER}(\{0, 1\}, \{\cup, \cdot, {}^*\})$ *is* $\leq_m^{\log}$*-complete in* L.
2. Let $\varphi \subseteq \{{}^*, {}^2\}$. *Then* $\mathrm{MEMBER}(\{0, 1\}, \{\cup, \cdot, \sim\} \cup \varphi)$ *is* $\leq_m^{\log}$*-complete in* P.

By proving that the recognition problem for context-free languages is $\leq_m^{\text{log-lin}}$-reducible to the membership problem of regular-like expressions with intersection operation, its hardness for the complexity class LOGCFL followed [52]. In the same paper the containment is shown, so that its completeness is obtained:

Theorem 13. *Both* $\mathrm{MEMBER}(\Sigma, \{\cup, \cdot, {}^*, \cap\})$ *and* $\mathrm{MEMBER}(\Sigma, \{\cup, \cdot, {}^2, \cap\})$ *are* $\leq_m^{\text{log-lin}}$*-complete in* LOGCFL.

For the special case of unary alphabets the currently known bound is NL [52]:

Theorem 14. $\mathsf{NL} \leq_m^{\log} \mathrm{MEMBER}(\{1\}, \{\cup, \cdot, \cap\})$.

Next we consider the problem $\mathrm{NEC}(\Sigma, \{\cup, \cdot, {}^*\})$—recall that non-empty complement means that the described language is not equal to Σ^*. The following result on classical regular expressions is obtained in [49,57].

Theorem 15. $\mathrm{NEC}(\{0, 1\}, \{\cup, \cdot, {}^*\})$ *is* $\leq_m^{\text{log-lin}}$*-complete in* $\mathsf{NLINSPACE}$.

In [57] it is shown that adding the squaring operator increases the complexity significantly:

Theorem 16. $\mathrm{NEC}(\{0,1\},\{\cup,\cdot,{}^*,{}^2\})$ *is* $\leq_m^{\text{log-lin}}$*-complete in* $\mathsf{EXPSPACE}$*. In particular, there is a constant* $c > 1$ *such that* $\mathrm{NEC}(\Sigma,\{\cup,\cdot,{}^*,{}^2\}) \notin \mathsf{NSPACE}(c^n)$ *and* $\mathrm{NEC}(\Sigma,\{\cup,\cdot,{}^*,{}^2\}) \in \mathsf{NSPACE}(2^n)$*.*

Basically, the idea to prove the lower bound is to encode computations of a complexity bounded Turing machine in regular-like expressions. For the upper bound, given an expression of length n first the squarings are expanded which gives an expression whose length is at most 2^n. Then Theorem 15 is applied which gives a space bound of $O(2^n)$.

Clearly, NEC is a special case of INEQ in the sense that, if a regular-like expression describes the language Σ^*, then an algorithm which decides $\mathrm{INEQ}(\Sigma,\varphi)$ immediately yields an algorithm that decides $\mathrm{NEC}(\Sigma,\varphi)$. Formally, we have $\mathrm{NEC}(\Sigma,\varphi) \leq_m^{\text{log-lin}} \mathrm{INEQ}(\Sigma,\varphi)$. A lower bound on the complexity of NEC is also a lower bound on that of INEQ. We conclude that we can immediately replace NEC by INEQ in Theorems 15 and 16, which yields the next result.

Theorem 17. $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot,{}^*\})$ *is* $\leq_m^{\text{log-lin}}$*-complete in* $\mathsf{NLINSPACE}$*, and the set* $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot,{}^*,{}^2\})$ *is* $\leq_m^{\text{log-lin}}$*-complete in* $\mathsf{EXPSPACE}$*.*

The proofs of the lower bounds in the previous theorems use reducibilities to regular-like expressions of star-height one. Therefore, the lower bounds also hold for the respective NEC or INEQ problems restricted to expressions of star-height one.

We now turn to show how the removal of the Kleene star operation affects the complexities of the problems [57,58]. Clearly, these are finite word problems. If $r \in \mathrm{RE}(\Sigma,\{\cup,\cdot\})$ or $r \in \mathrm{RE}(\Sigma,\{\cup,\cdot,{}^2\})$, the language $L(r)$ described by r is finite. In fact, if $\mathsf{size}(r) = n$, then $w \in L(r)$ implies $|w| \leq 2^n$. As intuitively expected in this cases the complexities are better to some extend.

Theorem 18. $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot\})$ *is* $\leq_m^{\text{log}}$*-complete in* NP *and, moreover, the problem* $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot,{}^2\})$ *is* $\leq_m^{\text{log-lin}}$*-complete in* NEXP*. Therefore there are rational constants* $c, d > 1$ *such that* $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot,{}^2\}) \notin \mathsf{NTIME}(c^n)$ *and* $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot,{}^2\}) \in \mathsf{NTIME}(d^n)$*.*

One reason for considering inequivalence rather than equivalence problems is that such problems are more suitable to solution by nondeterministic devices, which simply can guess a word in the symmetric difference. If an inequivalence problem turns out to be complete in some nondeterministic complexity class that is not (known to be) closed under complementation, then it may not be immediate (or even true) that the corresponding equivalence problem is complete in that class. On the other hand, since deterministic time (space) classes are closed under complementation for constructible bounds, a lower bound on the deterministic complexity of an inequivalence problem immediately gives a lower bound on the deterministic complexity of the corresponding inequivalence problem. Since NP is not known to be closed under complementation, Theorem 18 does not provide an interesting lower bound of $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot\})$. However, it implies an exponential upper bound: $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot\}) \in \mathsf{DTIME}(c^n)$ for some constant c. The exponential difference between the upper and lower bounds on deterministic time is closely related to famous open problems in complexity theory [57]:

Theorem 19

1. $\mathrm{NEC}(\{0,1\},\{\cup,\cdot,{}^*\}) \in \mathsf{P}$ *if and only if* $\mathsf{NLINSPACE} \subseteq \mathsf{P}$.
2. $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot\}) \in \mathsf{P}$ *if and only if* $\mathsf{P} = \mathsf{NP}$.
3. $\mathrm{NEC}(\{0,1\},\{\cup,\cdot,{}^*\}) \in \mathsf{DLINSPACE}$ *iff* $\mathsf{DLINSPACE} = \mathsf{NLINSPACE}$ *(LBA problem).*

Next, we consider regular-like expressions with the additional operation complementation. This includes the class of star-free expressions containing only the operations union, concatenation, and complementation. Even though star-free expressions cannot describe all regular languages, they can describe certain regular languages much more succinctly than classical regular expressions. Roughly speaking, it is possible to encode computations of one-tape one-head Turing machines which are $\mathrm{tow}(\lceil \log_b \rceil, 0)$ space bounded in star-free expressions of length $O(n)$, where $\mathrm{tow}(k,\ell)$ is the *tower function* given by $\mathrm{tow}(0,\ell)=\ell$ and $\mathrm{tow}(k+1,\ell) = 2^{\mathrm{tow}(k,\ell)}$, for $k \geq 0$ and real ℓ. So, several decision problems are enormously difficult. An analysis of the algorithm for checking equivalence presented in [10] revealed that for no fixed k the running time is bounded above by $\mathrm{tow}(k,n)$. It turned out that this complexity is inherent to the problem. So, the use of complementations makes decision problems expensive [57]:

Theorem 20. $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot,{}^*,\sim\}) \in \mathsf{NSPACE}(\mathrm{tow}(n,0))$.

The number of levels of the tower function is closely related to the depth of nesting of complementation operations in the expressions being checked for equivalence [57]:

Theorem 21. *Let $k \geq 1$ be a constant integer. If the depth of nesting of complementation operations in the regular-like expressions is at most k, then* $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot,{}^*,\sim\}) \in \mathsf{NSPACE}(\mathrm{tow}(k, 2\cdot n))$.

Now we turn to the lower bounds for the problems [57]: The next theorem concerns the case of unlimited complementation nesting.

Theorem 22. $\mathsf{NSPACE}(\mathrm{tow}(\lceil \log_b \rceil, 0)) \leq_m^{\text{p-lin}} \mathrm{NEC}(\{0,1\},\{\cup,\cdot,\sim\})$ *and additionally* $\mathrm{NEC}(\{0,1\},\{\cup,\cdot,\sim\}) \notin \mathsf{NSPACE}(\mathrm{tow}(\lceil \log_b \rceil, 0))$.

Whether the gap between the upper and lower bound can be decreased is an open question. Better matching upper and lower bounds are obtained by fixing the depth of complementation nesting. The next theorem summarizes the results shown in [57].

Theorem 23. *Let $k \geq 1$ be a constant integer. In the below given statements the* NEC*-problems are* restricted *to regular-like expressions of nesting depth of complementation at most k.*

1. *Then the decision problem* $\mathrm{NEC}(\{0,1\},\{\cup,\cdot,{}^*,\sim\})$ *is* $\leq_m^{\text{p-lin}}$*-complete in* $\bigcup_{d\geq 0} \mathsf{NSPACE}(\mathrm{tow}(k, d\cdot n))$. *In particular, there is a rational constant $c > 0$ such that* $\mathrm{NEC}(\{0,1\},\{\cup,\cdot,{}^*,\sim\}) \notin \mathsf{NSPACE}(\mathrm{tow}(k, c\cdot n))$ *but on the other hand* $\mathrm{NEC}(\{0,1\},\{\cup,\cdot,{}^*,\sim\}) \in \mathsf{NSPACE}(\mathrm{tow}(k, 2\cdot n))$.

2. *Moreover,* $\bigcup_{d\geq 0}\mathsf{NSPACE}(\mathrm{tow}(k-3, d\cdot\sqrt{n})) \leq_m^{\text{p-lin}} \mathrm{NEC}(\{0,1\},\{\cup,\cdot,\sim\})$*, for all integers* $k \geq 4$*.*
3. $\mathrm{NEC}(\{0,1\},\{\cup,\cdot,\sim\}) \notin \mathsf{NSPACE}(\mathrm{tow}(k-3, c\cdot\sqrt{n}))$*, for a rational* $c > 0$*.*

The remaining operation we consider is the intersection. In [37] several results concerning various decision problems for regular-like expressions with intersection operation are obtained. E.g., $\mathrm{NEC}(\{0,1\},\{\cup,\cdot,{}^*,\cap\}) \notin \mathsf{NSPACE}(c^{\sqrt{n/\log n}})$, for some $c > 1$, is shown. Thus the same lower bound follows for INEQ. The upper bound for the problems is the complexity class EXPSPACE, since it is easy to build a NFA with 2^n states which describes the language of a regular-like expression with intersections of size n. Then a product construction can be used for every intersection [16]. Therefore, we obtain an upper bound c^n, for some constant $c \geq 1$. The lower bound INEQ has been improved to $\mathsf{NSPACE}(c^{n/\log n})$ in [57]. Finally, in [16] it has been further improved to match the upper bound for both problems:

Theorem 24. *Both* $\mathrm{NEC}(\{0,1\},\{\cup,\cdot,{}^*,\cap\})$ *and* $\mathrm{INEQ}(\{0,1\},\{\cup,\cdot,{}^*,\cap\})$ *are* $\leq_m^{\text{log-lin}}$*-complete in* EXPSPACE*.*

The results so far revealed that the complexity of $\mathrm{INEQ}(\Sigma,\varphi)$ or $\mathrm{NEC}(\Sigma,\varphi)$ does not depend significantly on Σ provided it contains at least two elements. But the complexity of a problem can drastically be affected by the restriction to a unary alphabet. For example, Theorem 22 showed an expensive complexity for $\mathrm{INEQ}(\Sigma,\{\cup,\cdot,\sim\})$, while the following results from [57,58] and [52] show that it becomes much cheaper when Σ is a singleton. The proof relies on the fact that any expression r from $\mathrm{RE}(\{1\},\{\cup,\cdot,\sim\})$ either describes a finite or co-finite language and, moreover, that the words that are longer than $\mathsf{size}(r)$ are either all in or all *not* in $L(r)$. On the other hand, trading complementation for Kleene star makes the problem intractable again.

Theorem 25. *1.* $\mathrm{INEQ}(\{1\},\{\cup,\cdot,\sim\}) \in \mathsf{P}$*.*
2. $\mathrm{INEQ}(\{1\},\{\cup,\cdot,{}^*\})$ *is* $\leq_m^{\log}$*-complete in* NP*.*
3. $\mathsf{NL} \leq_m^{\log} \mathrm{INEQ}(\{1\},\{\cup,\cdot,\cap\})$ *and* $\mathrm{INEQ}(\{1\},\{\cup,\cdot,\cap\}) \in \mathsf{LOGCFL}$*.*
4. $\mathrm{INEQ}(\{1\},\{\cup,\cdot\}) \in \mathsf{NL}$*.*

In [58] the problem has been raised to determine the computational complexity of $\mathrm{INEQ}(\{1\},\{\cup,\cdot,{}^*,{}^2,\sim\})$, that is, all operations considered so far are allowed. In particular, the problem can be solved in time $\mathrm{tow}(k,n)$, for any fixed k. The problem has been solved in [53] by reducing the problem under consideration to Presburger Arithmetic. Removing the Kleene star operation yields a better complexity [58] and in [16] it is shown that the upper bound is not increased when the complementation is exchanged by both the Kleene star and intersection operation. We summarize these results in the following theorem.

Theorem 26

1. *There is a constant* $k \geq 1$ *such that* $\mathrm{INEQ}(\{1\},\{\cup,\cdot,{}^*,{}^2,\sim\})$ *is contained in* $\mathsf{DTIME}(\mathrm{tow}(k,n))$*.*
2. $\mathrm{INEQ}(\{1\},\{\cup,\cdot,{}^2,\sim\})$ *is* $\leq_m^{\log}$*-complete in* PSPACE*.*
3. $\mathrm{INEQ}(\{1\},\{\cup,\cdot,{}^*,{}^2,\cap\}) \in \mathsf{PSPACE}$*.*

4 Operation Problem for Regular Expressions

Regular languages are closed under various language operations regular expressions are a natural host for dozens of operation problems such as we have already seen in the previous section. Let $\circ$ be a fixed operation on languages that preserves regularity. Then the $\circ$-language operation problem for regular expressions is defined as follows:

– Given regular expressions s and t over alphabet Σ of size n and m, resp.—here size refers to one of the measures described earlier.
– How much size is sufficient and necessary in the worst case (in terms of n and m) to describe language $L(s) \circ L(t)$ by an *ordinary* regular expression?

Obviously, this problem generalizes to unary language operations like, for example, complementation. Treating the operation $\circ$ as a built-in function we end up with regular-like expressions from $\mathrm{RE}(\Sigma, \{\cup, \cdot, {}^*\} \cup \{\circ\})$, and one may ask a similar question as above when converting to equivalent ordinary regular expressions.

Determining the descriptional complexity of various basic language operations remained largely open until recently. For instance, it has long been known that it is non-elementary if we add complementation as built-in operator [58], but it remained open for a long time, whether the upper bound for the complementation problem of $2^{2^{O(n)}}$, induced by the naive algorithm, which converts first the given expression into an NFA, determinizes, complements the resulting DFA, and finally converts back to a regular expression, is asymptotically tight. Preliminary results were reported in [15] were a stunningly large gap between the lower and upper bound on the effect of complementation remained. Finally, the problem was solved in [20], and the above mentioned upper bound turned out to be tight already for binary alphabets [19,28]. The complementation problem for regular expressions over unary alphabets was already settled in [15].

Theorem 27. *Let $n \geq 1$ and r be a regular expression of size n over an alphabet Σ with at least two letters. Then size $2^{2^{\Theta(n)}}$ is sufficient and necessary in the worst case for a regular expressions describing the complement $\Sigma^* \setminus L(r)$. For unary alphabets, the tight bound reads as $2^{\Theta(\sqrt{n \log n})}$.*

This and some related results were driven by the introduction of new lower bound techniques for regular expressions. Properties related to regular expression size were first subject to systematic study in [14]. There a highly specialized (pumping) method for proving regular expression lower bounds, which seemingly requires a largely growing alphabet was developed. Based on this work encodings of the witness languages were proposed in [20] to reduce the alphabet size, retaining all necessary features required to mimic the original proof. A technique based on communication complexity that applies only for finite languages is proposed in [29]. The most general technique up to now, introduced in [25], reads as follows—for the definition of star height we refer to [46,47]:

Table 1. Lower and upper bounds on alphabetic width of language operations on languages of alphabetic width m and n, denoted by RE∘RE to RE, and on the required size for transforming expressions extended with the operator ∘ into ordinary regular expressions, denoted by $\mathrm{RE}(\Sigma, \{\cup, \cdot, ^*\} \cup \{\circ\})$ to RE. The bounds are for an alphabet fixed to two letters. All binary operations being symmetric, we assume $m \leq n$, and # stands for counting operators.

Operation ∘	Problem description ...	
	RE ∘ RE to RE	$\mathrm{RE}(\Sigma, \{\cup, \cdot, ^*\} \cup \{\circ\})$ to RE
$\sim$	$2^{2^{\Theta(n)}}$	non-elementary
$\cap$	$2^{\Omega(m)} \leq \cdot \leq n \cdot 2^{O(m\cdot(1+\log(n/m)))}$	$2^{2^{\Theta(n)}}$
ш	$2^{\Omega(m)} \leq \cdot \leq n \cdot 2^{O(m\cdot(1+\log(n/m)))}$	$\begin{cases} 2^{2^{\Omega(n/\log n)}} \leq \cdot \leq 2^{2^{O(n)}} \\ 2^{\Theta(n)} \text{ for } \lvert\Sigma\rvert \geq n \end{cases}$
#		$2^{\Theta(n)}$

Theorem 28. *Every regular language $L \subseteq \Sigma^*$ satisfies the inequality*

$$\mathsf{awidth}(L) \geq 2^{\frac{1}{3}\cdot(\mathsf{height}(L)-1)} - 1,$$

where $\mathsf{height}(L)$ refers to the star height of the regular language L, which is the minimum star height among all regular expressions describing L.

The theorem seems difficult to apply, since proving lower bounds on alphabetic width *via* lower bounds on star height appears to be trading a hard problem for an even harder one [32]. But the situation is not hopeless at all, because already early research done in [46,47] on the star height problem established a subclass of regular languages for which the star height can be determined easily. Together with clever encodings by star-height preserving homomorphisms as demonstrated in [28] makes the above theorem to a general purpose tool for proving lower bounds on alphabetic width for regular expressions.

Further results on regular expressions enhanced by the operations complementation, intersection, shuffle or interleaving, and counting (built-in or not) are summarized in Table 1 and are from [18,23,28,41,58]. Also the effect of a few other language operations on regular expression size has been studied, such as, e.g., half-removal, circular shift, and quotients. For a definition of these operations and descriptional complexity results we refer to [26,27]. It is worth mentioning that the latter two operations surprisingly induce a polynomial increase in regular expression size only. Regarding language operations in subregular language families, only a few scattered results for regular expressions are known to date [15,20,28].

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools. Addison-Wesley, Reading (1986)

3. Allauzen, C., Mohri, M.: A unified construction of the Glushkov, follow, and Antimirov automata. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 110–121. Springer, Heidelberg (2006)
4. Alon, N., Dewdney, A.K., Ott, T.J.: Efficient simulation of finite automta by neural nets. J. ACM 38, 495–514 (1991)
5. Antimirov, V.M.: Partial derivatives of regular expressions and finite automaton constructions. Theoret. Comput. Sci. 155, 291–319 (1996)
6. Berry, G., Sethi, R.: From regular expressions to deterministic automata. Theoret. Comput. Sci. 48, 117–126 (1986)
7. Bille, P., Thorup, M.: Regular expression matching with multi-strings and intervals. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 1297–1308. SIAM, Philadelphia (2010)
8. Brüggemann-Klein, A.: Regular expressions into finite automata. Theoret. Comput. Sci. 120, 197–213 (1993)
9. Brzozowski, J.A., McCluskey, E.J.: Signal flow graph techniques for sequential circuit state diagrams. IEEE Trans. Comput. C-12, 67–76 (1963)
10. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM 11, 481–494 (1964)
11. Champarnaud, J.M., Ouardi, F., Ziadi, D.: Normalized expressions and finite automata. Int. J. Algebra Comput. 17, 141–154 (2007)
12. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall, Boca Raton (1971)
13. Eggan, L.C.: Transition graphs and the star height of regular events. Michigan Math. J. 10, 385–397 (1963)
14. Ehrenfeucht, A., Zeiger, H.P.: Complexity measures for regular expressions. J. Comput. System Sci. 12, 134–146 (1976)
15. Ellul, K., Krawetz, B., Shallit, J., Wang, M.W.: Regular expressions: New results and open problems. J. Autom., Lang. Comb. 10, 407–437 (2005)
16. Fürer, M.: The complexity of the inequivalence problem for regular expressions with intersection. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 234–245. Springer, Heidelberg (1980)
17. Geffert, V.: Translation of binary regular expressions into nondeterministic ϵ-free automata with $O(n \log n)$ transitions. J. Comput. System Sci. 66, 451–472
18. 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)
19. Gelade, W.: Foundations of XML: Regular Expressions Revisited. PhD thesis, School voor Informatietechnologie, University of Hasselt, Belgium, and University of Maastricht, the Netherlands (2009)
20. Gelade, W., Neven, F.: Succinctness of the complement and intersection of regular expressions. In: Theoretical Aspects of Computer Science (STACS 2008), Schloss Dagstuhl, Germany. Dagstuhl Seminar Proceedings, IBFI, vol. 08001, pp. 325–336 (2008)
21. Ginzburg, A.: A procedure for checking equality of regular expressions. J. ACM 14, 355–362 (1967)
22. Glushkov, V.M.: The abstract theory of automata. Russian Math. Surveys 16, 1–53 (1961)
23. Gruber, H.: On the Descriptional and Algorithmic Complexity of Regular Languages. PhD thesis, Institut für Informatik, Universität Giessen, Germany (2010)
24. Gruber, H., Gulan, S.: Simplifying regular expressions. A quantitative perspective. In: Language and Automata Theory and Applications (LATA 2010), LNCS. Springer, Heidelberg (to appear 2010)

25. Gruber, H., Holzer, M.: Finite automata, digraph connectivity, and regular expression size. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 39–50. Springer, Heidelberg (2008)
26. Gruber, H., Holzer, M.: Provably shorter regular expressions from deterministic finite automata. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 383–395. Springer, Heidelberg (2008)
27. Gruber, H., Holzer, M.: Language operations with regular expressions of polynomial size. Theoret. Comput. Sci. 410, 3281–3289 (2009)
28. Gruber, H., Holzer, M.: Tight bounds on the descriptional complexity of regular expressions. In: DLT 2009. LNCS, vol. 5583, pp. 276–287. Springer, Heidelberg (2009)
29. Gruber, H., Johannsen, J.: Optimal lower bounds on regular expression size using communication complexity. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 273–286. Springer, Heidelberg (2008)
30. Gulan, S., Fernau, H.: An optimal construction of finite automata from regular expressions. In: Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008), Dagstuhl Seminar Proceedings, IBFI, Schloss Dagstuhl, Germany, vol. 08002, pp. 211–222 (2008)
31. Hagenah, C., Muscholl, A.: Computing epsilon-free NFA from regular expressions in $\mathrm{O}(\mathrm{n}\log^2(\mathrm{n}))$ time. RAIRO Inform. Théor. 34, 257–278 (2000)
32. Hashiguchi, K.: Algorithms for determining relative star height and star height. Inform. Comput. 78, 124–169 (1988)
33. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
34. Horne, B.G., Hush, D.R.: Bounds on the complexity of recurrent neural network implementations of finite state machines. Neural Networks 9, 243–252 (1996)
35. Hromkovič, J., Seibert, S., Wilke, T.: Translating regular expressions into small ϵ-free nondeterministic finite automata. J. Comput. System Sci. 62, 565–588 (2001)
36. Hromkovič, J.: Descriptional complexity of finite automata: Concepts and open problems. J. Autom., Lang. Comb. 7, 519–531 (2002)
37. Hunt III, H.B.: The equivalence problem for regular expressions with intersections is not polynomial in tape. Technical Report TR 73-161, Department of Computer Science, Cornell University, Ithaca, New York (1973)
38. Ilie, L., Yu, S.: Follow automata. Inform. Comput. 186, 140–162 (2003)
39. Indyk, P.: Optimal simulation of automata by neural nets. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 337–348. Springer, Heidelberg (1995)
40. Jiang, T., Ravikumar, B.: A note on the space complexity of some decision problems for finite automata. Inform. Process. Lett. 40, 25–31 (1991)
41. Kilpeläinen, P., Tuhkanen, R.: Regular expressions with numerical occurrence indicators – Preliminary results. In: Symposium on Programming Languages and Software Tools, Department of Computer Science, pp. 163–173. University of Kuopio, Finland (2003)
42. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Automata Studies, pp. 3–42. Princeton University Press, Princeton (1956)
43. Leiss, E.L.: The complexity of restricted regular expressions and the synthesis problem for finite automata. J. Comput. System Sci. 23, 348–354 (1981)
44. Lifshits, Y.: A lower bound on the size of ϵ-free NFA corresponding to a regular expression. Inform. Process. Lett. 85, 293–299 (2003)
45. McCulloch, W.S., Pitts, W.H.: A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophysics 5, 115–133 (1943)

46. McNaughton, R.: The loop complexity of pure-group events. Inform. Control 11, 167–176 (1967)
47. McNaughton, R.: The loop complexity of regular events. Inform. Sci. 1, 305–328 (1969)
48. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IRE Trans. Elect. Comput. EC-9, 39–47 (1960)
49. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: Symposium on Switching and Automata Theory (SWAT 1972), pp. 125–129. IEEE, Los Alamitos (1972)
50. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
51. Petersen, H.: Decision problems for generalized regular expressions. In: Descriptional Complexity of Automata, Grammars and Related Structures (DCAGRS 2000), London, Ontario, pp. 22–29 (2000)
52. Petersen, H.: The membership problem for regular expressions with intersection is complete in LOGCFL. In: Alt, H., Ferreira, A. (eds.) STACS 2002. LNCS, vol. 2285, pp. 513–522. Springer, Heidelberg (2002)
53. Rangel, J.L.: The equivalence problem for regular expressions over one letter is elementary. In: Symposium on Switching and Automata Theory (SWAT 1974), pp. 24–27. IEEE, Los Alamitos (1974)
54. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)
55. Sakarovitch, J.: The language, the expression, and the (small) automaton. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845, pp. 15–30. Springer, Heidelberg (2006)
56. Schnitger, G.: Regular expressions and NFAs without ϵ-transitions. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 432–443. Springer, Heidelberg (2006)
57. Stockmeyer, L.J.: The Complexity of Decision Problems in Automata Theory and Logic. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts (1974)
58. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: Symposium on Theory of Computing (STOC 1973), pp. 1–9. ACM Press, New York (1973)
59. Thompson, K.: Regular expression search algorithm. J. ACM 11, 419–422 (1968)
60. Watson, B.: A taxonomy of finite automata construction algorithms. Technical Report 93/43, Eindhoven University of Technology, Department of Mathematics and Computing Science (1995)

On Decision Problems for Simple and Parameterized Machines*

Oscar H. Ibarra

Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Abstract. We introduce what we believe to be the simplest known classes of language acceptors for which we can prove that decision problems (like universe, disjointness, containment, etc.) are undecidable. We also look at classes with undecidable decision problems and show that the problems become decidable for the "parameterized" versions of the machines.

* This research was supported in part by NSF Grant CCF-0524136.

DNA Computing and Its Implications for Theoretical Computer Science[⋆]

Lila Kari

Department of Computer Science
University of Western Ontario
London, Ontario, N6A 5B7 Canada

DNA computing is an area of natural computing based on the idea that molecular biology processes can be used to perform arithmetic and logic operations on information encoded as DNA strands. The aim of this review is to describe some of the ways in which DNA computing research has impacted fields in theoretical computer science. We namely describe how properties of DNA-based information, and in particular the Watson-Crick complementarity of DNA single strands, have influenced areas of theoretical computer science such as formal language theory, coding theory, automata theory and combinatorics on words.

More precisely, we summarize notions and results in formal language theory and coding theory arising from the problem of design of optimal encodings for DNA computing experiments: the problem of DNA encodings design, an analysis of intramolecular bonds (bonds within a given DNA strand), the design of languages that avoid certain undesirable intermolecular bonds (bonds between two or more DNA strands), and of languages whose words avoid even imperfect bindings between their constituent strands. We also present another representation of DNA partial double strands, as two-line vectors, we describe the sticking operation that combines them, and two computational models based on this representation: sticker systems and Watson-Crick automata.

We also describe the influence that properties of DNA-based information have had on research in combinatorics on words, by enumerating several natural generalizations of classical concepts of combinatorics of words: pseudo-palindromes, pseudo-periodicity, Watson-Crick conjugate and commutative words, involutively bordered words, pseudoknot bordered words, pseudo-powers, and pseudo-DeBruijn sequences. In addition, we outline natural extensions in this context of two of the most fundamental results in combinatorics of words, namely Fine and Wilf's theorem and Lyndon-Schützenberger result.

We conclude by presenting general thoughts on DNA-based information, bioinformation and biocomputation.

[⋆] This work was supported by The Natural Sciences and Engineering Council of Canada Discovery Grant and Canada Research Chair Award to L.K.

Numeration Systems: A Link between Number Theory and Formal Language Theory

Michel Rigo*

Université de Liège, Institut de Mathématiques, Grande Traverse 12 (B 37), B-4000 Liège, Belgium
M.Rigo@ulg.ac.be

Abstract. We survey facts mostly emerging from the seminal results of Alan Cobham obtained in the late sixties and early seventies. We do not attempt to be exhaustive but try instead to give some personal interpretations and some research directions. We discuss the notion of numeration systems, recognizable sets of integers and automatic sequences. We briefly sketch some results about transcendence related to the representation of real numbers. We conclude with some applications to combinatorial game theory and verification of infinite-state systems and present a list of open problems.

1 Introduction

It is challenging to give a talk about the interactions existing between formal language theory and number theory. The topic is vast, has several entry points and many applications. To cite just a few: non-adjacent form (NAF) representations to speed up computations arising in elliptic curve cryptography [61], verification of infinite-state systems [23], combinatorial game theory, fractals and tilings [82,20], transcendence results, dynamical systems and ergodic theory [19, Chap. 5–7], [13,73]. For instance, there exist tight and fruitful links between properties sought for in dynamical systems and combinatorial properties of the corresponding words and languages.

The aim of this paper is to briefly survey some results mostly emerging from the seminal papers of Cobham of the late sixties and early seventies [35,36,37], while also trying to give some personal interpretations and some research directions. We do not provide an exhaustive survey of the existing literature but we will give some pointers that we hope could be useful to the reader.

When one considers such interactions, the main ingredient is definitely the notion of numeration system, which provides a bridge between a set of numbers (integers, real numbers or elements of some other algebraic structures [68,9]) and formal language theory. On the one hand, arithmetic properties of numbers or sets of numbers are of interest and on the other hand, syntactical properties of the corresponding representations may be studied. To start with, we consider the familiar integer base $k \geq 2$ numeration system. Any integer $n > 0$ is uniquely

* Dedicated to the memory of my grandfather Georges Henderyckx 1930–2010.

represented by a finite word (its k-ary representation) $\mathrm{rep}_k(n) = d_\ell \cdots d_0$ over the alphabet $A_k = \{0, \ldots, k-1\}$ such that $\sum_{i=0}^{\ell} d_i\, k^i = n$ and $d_\ell \neq 0$. Note that imposing the uniqueness of the representation allows us to define a map $\mathrm{rep}_k : \mathbb{N} \to A_k^*$. Nevertheless, in many contexts it is useful to consider all the representations of an integer n over a given finite alphabet $D \subset \mathbb{Z}$, that is all the words $c_\ell \cdots c_0 \in D^*$ such that $\sum_{i=0}^{\ell} c_i\, k^i = n$. For instance, if w is the k-ary representation of n and if the alphabet D is simply A_k, then for all $j > 0$, $0^j w$ is another representation of n.

In the same way, any real number $r \in (0,1)$ is represented by an infinite word $d_1 d_2 \cdots$ over A_k such that $\sum_{i=1}^{+\infty} d_i\, k^{-i} = r$. Uniqueness of the representation may be obtained by taking the maximal word for the lexicographic ordering on A_k^ω satisfying the latter equality; in this case, the sequence $(d_i)_{i \geq 1}$ is not ultimately constant and equal to $k-1$: there is no N such that, for all $n \geq N$, $d_n = k-1$. Therefore, to represent a real number $r > 0$, we take separately its integer part $\lfloor r \rfloor$ and its fractional part $\{r\}$. Base k-complements or signed number representations [70] can be used to represent negative elements as well, the sign being determined by the most significant digit which is thus 0 or $k-1$. By convention, the empty word ε represents 0, i.e., $\mathrm{rep}_k(0) = \varepsilon$. If the numeration system is fixed, say the base k is given, then any integer n (resp. any real number $r > 0$) corresponds to a finite (resp. infinite) word over A_k (resp. over $A_k \cup \{\star\}$, where $\star$ is a new symbol used as a separator). Therefore any set of numbers corresponds to a language of representations and we naturally seek for some link between the arithmetic properties of the numbers belonging to the set and the syntactical properties of the corresponding representations. Let X be a subset of $\mathbb{N}$. Having in mind Chomsky's hierarchy, the set X could be considered quite "simple" from an algorithmic point of view whenever the set of k-ary representations of the elements in X is a regular (or rational) language accepted by a finite automaton. A set $X \subseteq \mathbb{N}$ satisfying this property is said to be *k-recognizable*. Note that X is k-recognizable if and only if $0^* \mathrm{rep}_k(X)$ is regular. As an example, a DFA (i.e., a deterministic finite automaton) accepting exactly the binary representations of the integers congruent to 3 (mod 4) is given in Figure 1. Similarly, a set $X \subseteq \mathbb{R}$ of real numbers is *k-recognizable* if there exists a finite (non-deterministic) Büchi automaton accepting *all* the k-representations over A_k of the elements in X, that is, the representations starting with an arbitrary number of leading zeroes, and in particular the ones ending with $(k-1)^\omega$. Such an automaton is often called a *Real Number Automaton* [25]. The Büchi automaton in Figure 2 (borrowed from a talk given by V. Bruyère) accepts all

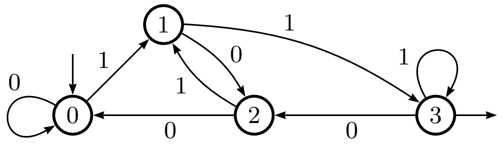


Fig. 1. A finite automaton accepting $0^*\,\mathrm{rep}_2(4\mathbb{N}+3)$

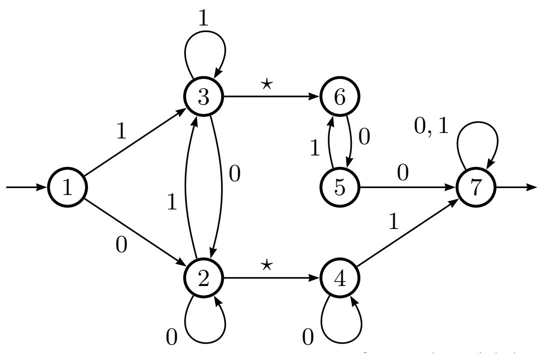


Fig. 2. A Büchi automaton accepting $\{2n + (0, 4/3) \mid n \in \mathbb{Z}\}$

the possible binary encodings (using base 2-complement for negative numbers) of elements in the set $\{2n+(0,4/3) \mid n \in \mathbb{Z}\}$. For instance 3/2 is encoded by the language of infinite words $0^+1\star10^\omega \cup 0^+1\star01^\omega$. Note that the transition $3 \xrightarrow{\star} 6$ (resp. $2 \xrightarrow{\star} 4$) is considered for an odd (resp. even) integer part and the series $\sum_{i=1}^{+\infty} 4^{-i} = 1/3$ corresponds to the cycle $\{5, 6\}$.

To generalize the k-ary integer base system, it is quite natural to consider an increasing sequence of integers, like the Fibonacci sequence, as a numeration basis to get a greedy decomposition of any integer (see Definition 2) or the negative powers of a real number $\beta > 1$ to develop any real $r \in (0, 1)$ as $\sum_{i=1}^{+\infty} c_i\,\beta^{-i}$ with the coefficients c_i belonging to a convenient finite alphabet. Let us point out Fraenkel's paper [54] for some general ideas about representations of integers in various numeration systems. Among non-standard decompositions of integers, let us mention the so-called *combinatorial numeration system* going back to Lehmer and Katona, where integers are decomposed using binomial coefficients with some prescribed property, also see [33], and the *factorial numeration system* [72]. In Frougny and Sakarovitch's chapter [19, Chap. 2] and in Frougny's chapter [76, Chap. 7] many details on recognizable sets and about the representation of integers and real numbers are given. In particular, Parry's β-developments of real numbers [80] are presented in the latter reference. Abstract numeration systems (see Definition 6) are a general framework to study recognizable sets of integers, see [71] and [19, Chap. 3].

The seminal work of Cobham may be considered as a starting point for the study of recognizable sets for at least three reasons. Let us sketch these below. Details and definitions will be given in the next sections.

(i) Cobham's theorem from 1969 [36] states that the recognizability of a set of integers, as introduced above, strongly depends on the choice of the base, e.g., there are sets which are 2-recognizable but not 3-recognizable. The only subsets of $\mathbb{N}$ that are recognizable in all bases are exactly the ultimately periodic sets, i.e., the finite unions of arithmetic progressions. See Theorem 1 in Section 2 below for the exact statement of the result. It is an easy exercise to show that an arithmetic progression is k-recognizable for all $k \geq 2$

(e.g., Figure 1). See for instance [85, prologue] about the *machine à diviser de Blaise Pascal.* In that direction, an interesting question [7] is to obtain the state complexity of the minimal automaton recognizing a given divisibility criterion in an integer base. For this state complexity question studied in the wider context of linear numeration systems (cf. Definition 3), see [32].

The base dependence of recognizability shown by Cobham's result strongly motivates the general study of recognizable sets and the introduction of non-standard or exotic numeration systems based on an increasing sequence satisfying a linear recurrence relation.

For integer base k numeration systems, nice characterizations of recognizable sets are well-known: logical characterization by first order formulas in a suitable extension of the Presburger arithmetic $\langle \mathbb{N}, + \rangle$, k-automatic characteristic sequence generated through a uniform morphism of length k, characterization in terms of algebraic formal power series for a prime base. See the excellent survey [29] where the so-called Cobham–Semenov' theorem, which extends Cobham's original result from 1969 to subsets of $\mathbb{N}^d$, $d \geq 2$, is also presented. Recall that the *characteristic sequence* $(x_i)_{i\geq 0} \in \{0,1\}^{\mathbb{N}}$ of $X \subseteq \mathbb{N}$ is defined by $x_i = 1$ if and only if $i \in X$. It is not our goal to give here a full list of pointers to the existing bibliography on the ramifications of Cobham's theorem, see for instance [48]. For presentations of Cobham's theorem based on Georges Hansel's work, see [81,12] together with [84].

(ii) The next paper of Cobham from 1972 [37] introduced the concept of k-automatic sequences (originally called tag sequences, see Definition 5) and linked numeration systems with the so-called substitutions and morphic words (see Definition 4). It is easy to see that a set $X \subseteq \mathbb{N}$ is k-recognizable if and only if the characteristic sequence of X is a k-automatic infinite word over $\{0,1\}$. For a comprehensive book on k-automatic sequences, see [12]. As we will see, this approach gives another way to generalize the notion of a recognizable set by considering sets having a morphic characteristic sequence (see Remark 2). Details will be presented in Section 3.

(iii) As the reader may already have noticed, this survey is mainly oriented towards sets of numbers (integers) giving rise to a language of representations. Another direction should be to consider a single real number and the infinite word representing it in a given base. Alan Cobham also conjectured the following result proved later on by Adamczewski and Bugeaud. *Let α be an algebraic irrational real number. Then the base-k expansion of α cannot be generated by a finite automaton.* Cobham's question follows a question of Hartmanis and Stearns [64]: *does it exist an algebraic irrational number computable in linear time by a (multi-tape) Turing machine?* In the same way, if an infinite word w over the finite alphabet A_k of digits has some specific combinatorial properties (like, a low factor complexity, or being morphic or substitutive), is the corresponding real number having w as k-ary representation transcendental? Let us mention that several surveys in that direction are worth of reading [77, Chap. 10], [19, Chap. 8], [2,92]. We will briefly sketch some of these developments in Section 4.

In Section 5, we sketch some of the links existing between numeration systems, combinatorics on words and combinatorial game theory. Cobham's theorem about base dependence also appears in the framework of the verification of infinite state systems, see Section 6. Finally, in Section 7 we give some paths to the literature that the interested reader may follow and in Section 8, we present some open questions.

2 Cobham's Theorem and Base Dependence

Two integers $k, \ell \geq 2$ are *multiplicatively independent* if the only integers m, n such that $k^m = \ell^n$ are $m = n = 0$. Otherwise stated, $k, \ell \geq 2$ are multiplicatively independent if and only if $\log k / \log \ell$ is irrational. Recall a classical result in elementary number theory, known as Kronecker's theorem: *if* $\theta > 0$ *is irrational, then the set* $\{\{n\theta\} \mid n > 0\}$ *is dense in* $[0, 1]$. Such result is an important argument in the proof of the following result.

Theorem 1 (Cobham's theorem [36]). *Let $k, \ell \geq 2$ be two multiplicatively independent integers. A set $X \subseteq \mathbb{N}$ is simultaneously k-recognizable and ℓ-recognizable if and only if X is ultimately periodic.*

Obviously the set $P_2 = \{2^n \mid n \geq 1\}$ of powers of two is 2-recognizable because $\text{rep}_2(P_2) = 10^*$. But since P_2 is not ultimately periodic, Cobham's theorem implies that P_2 cannot be 3-recognizable. To see that a given infinite ordered set $X = \{x_0 < x_1 < x_2 < \cdots\}$ is k-recognizable for *no* base $k \geq 2$ at all, we can use results like the following one where the behavior of the ratio (*resp.* difference) of any two consecutive elements in X is studied through the quantities

$$\mathbf{R}_X := \limsup_{i\to\infty} \frac{x_{i+1}}{x_i} \text{ and } \mathbf{D}_X := \limsup_{i\to\infty} (x_{i+1} - x_i).$$

Theorem 2 (Gap theorem [37]). *Let $k \geq 2$. If $X \subseteq \mathbb{N}$ is a k-recognizable infinite subset of $\mathbb{N}$, then either $\mathbf{R}_X > 1$ or $\mathbf{D}_X < +\infty$.*

Corollary 1. *Let $a \in \mathbb{N}_{\geq 2}$. The set of primes and the set $\{n^a \mid n \geq 0\}$ are never k-recognizable for any integer base $k \geq 2$.*

Proofs of the Gap theorem and its corollary can also be found in [50]. It is easy to show that $X \subseteq \mathbb{N}$ is k-recognizable if and only if it is k^n-recognizable, $n \in \mathbb{N} \setminus \{0\}$. As a consequence of Cobham's theorem, sets of integers can be classified into three categories:

– ultimately periodic sets which are recognizable for all bases,
– sets which are k-recognizable for some $k \geq 2$, and which are ℓ-recognizable only for those $\ell \geq 2$ such that k and ℓ are multiplicatively dependent bases, for example, the set P_2 of powers of two,
– sets which are k-recognizable for no base $k \geq 2$ at all, for example, the set of squares.

Definition 1. *An infinite ordered set $X = \{x_0 < x_1 < x_2 < \cdots\}$ such that $\mathbf{D}_X < +\infty$ is said to be* syndetic *or with* bounded gaps. *Otherwise stated, X is syndetic if there exists $C > 0$ such that, for all $n \geq 0$, $x_{n+1} - x_n < C$.*

If $X \subseteq \mathbb{N}$ is ultimately periodic, then X is syndetic. Note that the converse does not hold. For instance, consider the complement of the set $\{2^n \mid n \geq 0\}$ which is syndetic, 2-recognizable but not ultimately periodic.

Example 1 (Thue–Morse set). Let $n \in \mathbb{N}$. Denote by $s_k(n)$ the classical number-theoretic function summing up the digits appearing in $\mathrm{rep}_k(n)$. As a classical example, consider the set $T = \{n \in \mathbb{N} \mid s_2(n) \equiv 0 \bmod 2\}$. This set is 2-recognizable and syndetic but not ultimately periodic. It appears in several contexts [11] and in particular, it provides a solution to Prouhet's problem (also known as the Prouhet–Tarry–Escott problem which is a special case of a multi-grade equation).

The set of squares is k-recognizable for no integer base k but as we shall see this set is recognizable for some non-standard numeration systems (see Example 4). One possible extension of k-ary numeration systems is to consider a numeration basis.

Definition 2. *A* numeration basis *is an increasing sequence $U = (U_n)_{n\geq 0}$ of integers such that $U_0 = 1$ and $\sup_{i\geq 0} U_{i+1}/U_i$ is bounded.*

Using the greedy algorithm, any integer $n > 0$ has a unique decomposition

$$n = \sum_{i=0}^{\ell} c_i\, U_i$$

where the coefficients c_i belong to the finite set $A_U = \{0, \ldots, \sup\lceil U_{i+1}/U_i\rceil - 1\}$. Indeed there exists a unique $\ell \geq 0$ such that $U_\ell \leq n < U_{\ell+1}$. Set $r_\ell = n$. For all $i = \ell, \ldots, 1$, proceed to the Euclidean division $r_i = c_i\, U_i + r_{i-1}$, with $r_{i-1} < U_i$. The word $c_\ell \cdots c_0$ is the (normal) *U-representation* of n and is denoted by $\mathrm{rep}_U(n)$. Naturally, these non-standard numeration systems include the usual integer base k system by taking $U_n = k^n$ for all $n \geq 0$. The *numerical value map* $\mathrm{val}_U : A_U^* \to \mathbb{N}$ maps any word $d_\ell \cdots d_0$ over A_U onto $\sum_{i=0}^{\ell} d_i U_i$.

Remark 1. By contrast with abstract numeration systems that will be introduced later on, when dealing with a numeration basis we often use the terminology of a *positional numeration system* to emphasize the fact that a digit $d \in A_U$ in the ith position (counting from the right, i.e., considering the least significant digit first) of a U-representation has a weight d multiplied by the corresponding element U_i of the basis.

Having in mind the notion of k-recognizable sets, a set $X \subseteq \mathbb{N}$ is said to be *U-recognizable* if $\mathrm{rep}_U(X) = \{\mathrm{rep}_U(n) \mid n \in X\}$ is a regular language over the alphabet A_U. Note that $\mathrm{rep}_U(X)$ is regular if and only if $0^* \,\mathrm{rep}_U(X)$ is regular.

Definition 3. *A numeration basis $U = (U_n)_{n\geq 0}$ is said to be* linear, *if there exist $a_0, \dots, a_{k-1} \in \mathbb{Z}$ such that*

$$\forall n \geq 0,\ U_{n+k} = a_{k-1}U_{n+k-1} + \cdots + a_0 U_n. \tag{1}$$

If $\lim_{n\to\infty} U_{n+1}/U_n = \beta$ for some real $\beta > 1$, then U is said to satisfy the dominant root condition *and β is called the* dominant root *of the recurrence.*

If $\mathbb{N}$ is U-recognizable, then U is a linear numeration basis [89,19] (hint: observe that $\mathrm{rep}_U(\{U_n \mid n \geq 0\}) = 10^*$). For a discussion on sufficient conditions on the recurrence relation satisfied by U for the U-recognizability of $\mathbb{N}$, see [65] and [75]. In particular, as shown by the next result, *for a linear numeration basis U, the set $\mathbb{N}$ is U-recognizable if and only if all ultimately periodic sets are U-recognizable.*

Theorem 3 (Folklore [19]). *Let $p, r \geq 0$. If $U = (U_n)_{n\geq 0}$ is a linear numeration basis, then*

$$\mathrm{val}_U^{-1}(p\,\mathbb{N} + r) = \left\{ c_\ell \cdots c_0 \in A_U^* \mid \sum_{k=0}^{\ell} c_k\, U_k \in p\,\mathbb{N} + r \right\}$$

is accepted by a DFA that can be effectively constructed. In particular, if $\mathbb{N}$ is U-recognizable, then any ultimately periodic set is U-recognizable.

Example 2. Consider the Fibonacci numeration system given by the basis $F_0 = 1$, $F_1 = 2$ and $F_{n+2} = F_{n+1} + F_n$ for all $n \geq 0$. For this system, $0^*\,\mathrm{rep}_F(\mathbb{N})$ is given by the set of words over $\{0, 1\}$ avoiding the factor 11 and the set of even numbers is U-recognizable [32] using the DFA shown in Figure 3.

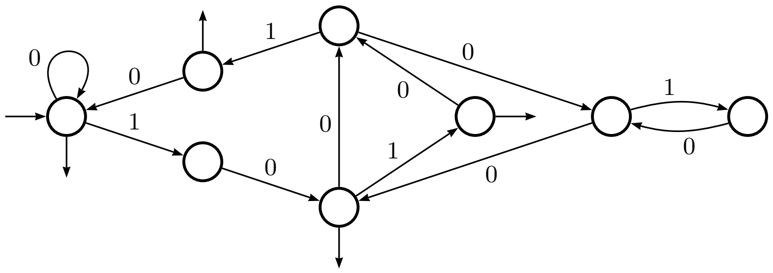


Fig. 3. A finite automaton accepting $0^*\,\mathrm{rep}_F(2\mathbb{N})$

To conclude this section, we present a linear numeration basis U such that the set of squares $\mathcal{Q} = \{n^2 \mid n \in \mathbb{N}\}$ is U-recognizable. This set will also be used in Example 4 to get a set having a morphic characteristic sequence.

Example 3. Consider the sequence given $U_n = (n+1)^2$ for all $n \geq 0$. This sequence satisfies, for all $n \geq 0$, the relation $U_{n+3} = 3\,U_{n+2} - 3\,U_{n+1} + U_n$. In that case, $\mathrm{rep}_U(\mathbb{N}) \cap 10^*10^* = \{10^a10^b \mid b^2 < 2a + 4\}$ showing with the pumping lemma that $\mathbb{N}$ is not U-recognizable [89]. But trivially, we have $\mathrm{rep}_U(\mathcal{Q}) = 10^*$.

3 Substitutions and Abstract Numeration Systems

For basic facts on morphisms over A^* or the usual distance put on A^ω (which gives a notion of convergence), see classic textbooks like [12,19,76]. Let A be a finite alphabet and $\sigma : A^* \to A^*$ be a morphism. If there exist a letter $a \in A$ and a word $u \in A^+$ such that $\sigma(a) = au$ and moreover, if $\lim_{n\to+\infty} |\sigma^n(a)| = +\infty$, then σ is said to be *prolongable* on a.

Definition 4. *Let $\sigma : A^* \to A^*$ be a morphism prolongable on a. We have*

$$\sigma(a) = a\,u,\ \sigma^2(a) = a\,u\,\sigma(u),\ \sigma^3(a) = a\,u\,\sigma(u)\,\sigma^2(u),\ \ldots\ .$$

Since, for all $n \in \mathbb{N}$, $\sigma^n(a)$ is a prefix of $\sigma^{n+1}(a)$ and because $|\sigma^n(a)|$ tends to infinity when $n \to +\infty$, the sequence $(\sigma^n(a))_{n\geq 0}$ converges to an infinite word denoted by $\sigma^\omega(a)$ and given by

$$\sigma^\omega(a) := \lim_{n\to+\infty} \sigma^n(a) = a\,u\,\sigma(u)\,\sigma^2(u)\,\sigma^3(u)\cdots\ .$$

This infinite word is a fixed point of σ. An infinite word obtained in this way by iterating a prolongable morphism is said to be purely morphic. *In the literature, one also finds the term* pure morphic. *If $x \in A^{\mathbb{N}}$ is purely morphic and if $\tau : A \to B$ is a coding (or letter-to-letter morphism), then the word $y = \tau(x)$ is said to be* morphic.

Definition 5. *Let $k \geq 2$. A morphic word $w \in B^\omega$ is* k-automatic *if there exists a morphism $\sigma : A^* \to A^*$ and a coding τ such that $w = \tau(\sigma^\omega(a))$ and, for all $c \in A$, $|\sigma(c)| = k$. A morphism satisfying this latter property is said to be* uniform.

The link between k-recognizable sets and k-automatic sequences is given by the following result. In particular, in the proof of this result, it is interesting to note that an automaton is canonically associated with a morphism.

Theorem 4. *[37] An infinite word $w = w_0w_1w_2\cdots$ over an alphabet A is k-automatic if and only if, for all $a \in A$, the set $X_a = \{i \in \mathbb{N} \mid w_i = a\}$ is k-recognizable.*

Otherwise stated, $w = w_0w_1w_2\cdots \in A^\omega$ is k-automatic if and only if there exists a deterministic finite automaton with output (DFAO) $\mathcal{M}$ where Q is the set of states of $\mathcal{M}$, $\delta : Q \times A_k \to Q$ (resp. $\tau : Q \to A$) is the transition function (resp. output function) of $\mathcal{M}$, such that $\tau(\delta(q_0, \mathrm{rep}_k(n))) = w_n$ for all $n \geq 0$.

Remark 2. Using automata as a model of computation, U-recognizable sets naturally raise some interest. On the same level, sets of integers having a morphic characteristic sequence can be considered as another natural generalization of the concept of k-recognizability. Iterations of a morphism may be used to get inductively further elements of the set defined by the morphism and a coding. As will be shown by Theorem 6, similarly to the case of uniform morphisms (as given in Definition 5) described above, the computation of a given element can also be done by using a DFAO and representations of integers in an abstract numeration system.

Example 4. Consider the alphabet $A = \{a, b, c\}$ and the morphism $\sigma : A^* \to A^*$ defined by $\sigma : a \mapsto abcc, b \mapsto bcc, c \mapsto c$. We get

$$\sigma^\omega(a) = abccbccccbccccccbccccccccbccccccccccbcc \cdots .$$

It is easy to see that considering the coding $\tau : a, b \mapsto 1$ and $\tau : c \mapsto 0$, the word $\tau(\sigma^\omega(a))$ is the characteristic sequence of the set of squares.

The *factor complexity* of an infinite word w is the non-decreasing function $p_w : \mathbb{N} \to \mathbb{N}$ mapping n onto the number of distinct factors (or subwords) occurring in w. See for instance [19, Chap. 4]. For a survey on the factor complexity of morphic words, see [8]. In 1972, Cobham already observed that if w is k-automatic, then p_w is in $\mathcal{O}(n)$. For instance, the factor complexity of the characteristic sequence of the Thue–Morse set T considered in Example 1 is computed in [27,39].

Theorem 5 (Morse–Hedlund's Theorem). *Let $x = x_0x_1x_2 \cdots$ be an infinite word over A. The following conditions are equivalent.*

– *The complexity function p_x is bounded by a constant, i.e., there exists C such that for all $n \in \mathbb{N}$, $p_x(n) \leq C$.*
– *There exists $N_0 \in \mathbb{N}$ such that for all $n \geq N_0$, $p_x(n) = p_x(N_0)$.*
– *There exists $N_0 \in \mathbb{N}$ such that $p_x(N_0) = N_0$.*
– *There exists $m \in \mathbb{N}$ such that $p_x(m) = p_x(m + 1)$.*
– *The word x is ultimately periodic.*

In particular, non ultimately periodic sequences with low complexity are the so-called *Sturmian sequences* whose factor complexity is $p(n) = n + 1$ for all $n \geq 1$. Note that such sequences are over a binary alphabet, $p(1) = 2$. For a survey on Sturmian words, see for instance [76]. Since Pansiot's work [79], the factor complexity of a non ultimately periodic purely morphic word w is well-known, see for instance [19, Chap. 4] or the survey [8], there exists constants C_1, C_2 such that $C_1 f(n) \leq p_w(n) \leq C_2 f(n)$ where $f(n) \in \{n, n \log n, n \log\log n, n^2\}$.

Remark 3. F. Durand has achieved a lot of work towards a general version of Cobham's theorem for morphic words [45,46,47]. Without giving much details (see for instance [48] for a detailed account), with a non-erasing morphism σ over $A = \{a_1, \ldots, a_t\}$ (i.e., $\sigma(\sigma_i) \neq \varepsilon$ for all i) generating a morphic word w (also using an extra coding) is associated a matrix $\mathbf{M}_\sigma$ (like the adjacency matrix of a graph) where, for all i, j, $(\mathbf{M}_\sigma)_{i,j}$ is the number of occurrences of the letter a_i in the image $\sigma(a_j)$. Considering the morphism in Example 4, we get

$$\mathbf{M}_\sigma = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 2 & 1 \end{pmatrix}.$$

Then considering the irreducible components (i.e., the strongly connected components of the associated automaton) of the matrix $\mathbf{M}_\sigma$ and the theorem of

Perron–Frobenius, a real number $\beta > 0$ is associated with the morphism. The word w is therefore said to be *β-substitutive.* Let $\alpha, \beta > 1$ be two multiplicatively independent Perron numbers (the notion of multiplicative independence extends to real numbers > 1). Under some mild assumptions [48], *if w is both α-substitutive and β-substitutive, then it is ultimately periodic.* It is a natural generalization of the fact that if $k, \ell \geq 2$ are multiplicatively independent, then a word which is both k-automatic and ℓ-automatic is ultimately periodic.

Example 5. The consecrated Fibonacci word, i.e., the unique fixed point of $\sigma : 0 \mapsto 01, 1 \mapsto 0$, is α-substitutive where α is the Golden ratio $(1+\sqrt{5})/2$. Therefore, this infinite word is k-automatic for no integer $k \geq 2$. Indeed, k and the Golden ratio are multiplicatively independent.

In view of Theorem 3, it is desirable for a numeration basis U that the set $\mathbb{N}$ be U-recognizable. In that case, one can use a finite automaton to test whether or not a given word over A_U is a valid U-representation. Taking this requirement as a basic assumption and observing that for all integers x, y, we have $x < y$ if and only if $\text{rep}_U(x)$ is genealogically less than $\text{rep}_U(y)$, we introduce the concept of an abstract numeration system. To define the *genealogical order* (also called *radix* or *military order*), first order words by increasing length and for words of the same length, take the usual lexicographical order induced by the ordering of the alphabet.

Definition 6. *Let L be an infinite regular language over a totally ordered alphabet $(A, <)$. An* abstract numeration system *is the triple $S = (L, A, <)$. Ordering by increasing genealogical order the words in L provides a one-to-one correspondence between L and $\mathbb{N}$. The nth word in L (starting from 0) is denoted by $\text{rep}_S(n)$ and the inverse map $\text{val}_S : L \to \mathbb{N}$ is such that $\text{val}_S(\text{rep}_S(n)) = n$. Any numeration basis U such that $\mathbb{N}$ is U-recognizable is a particular case of an abstract numeration system. In this respect, a set $X \subseteq \mathbb{N}$ is S-*recognizable*, if $\text{rep}_S(X)$ is a regular language.*

*A sequence $w = w_0w_1\cdots$ is S-*automatic *if there exists a DFAO $\mathcal{M}$ where $\delta : Q \times A_k \to Q$ (resp. $\tau : Q \to A$) is the transition function (resp. output function) of $\mathcal{M}$, such that $\tau(\delta(q_0, \text{rep}_S(n)) = w_n$ for all $n \geq 0$.*

Example 6. Again the set of squares $\mathcal{Q}$ is S-recognizable for the abstract numeration system $S = (a^*b^* \cup a^*c^*, \{a, b, c\}, a < b < c)$. Indeed, we have

$$a^*b^* \cup a^*c^* = \varepsilon, a, b, c, aa, ab, ac, bb, cc, aaa, \ldots$$

and one can check that $\text{rep}_S(\mathcal{Q}) = a^*$ because the growth function of the language is $\#((a^*b^* \cup a^*c^*) \cap \{a, b, c\}^n) = 2n + 1$.

Theorem 4 can be generalized as follows [83] or [19, Ch. 3].

Theorem 6. *An infinite word $w = w_0w_1w_2\cdots$ over an alphabet A is morphic if and only if there exists an abstract numeration system S such that w is S-automatic.*

Note that for generalization of Theorems 4 and 6 to a multidimensional setting, see Salon's work [86,87] and [31] respectively. Moreover, thanks to the above result, Durand's work can also to some extent be expressed in terms of abstract numeration systems. Observe that in Example 6 the abstract numeration system is based on a regular language having a polynomial growth. This corresponds to the case where the dominating eigenvalue of the matrix associated with the morphism is 1. Such a situation (polynomial versus exponential growth) is considered in [49]. Indeed, note that in the discussion about a morphic version of Durand–Cobham's theorem in Remark 3 we only considered morphisms with exponential growth, i.e., the dominating eigenvalue being > 1.

4 Transcendental Numbers

This short section is based on a lecture given by B. Adamczewski during the last CANT summer school in Liège [19, Chap. 8] and on [92]. We also refer the reader to [2]. It illustrates one of the strong links existing between combinatorics on words and number theory. For a survey on combinatorics on words, see for instance [17,34]. Recall that a complex number which is a root of a non-zero polynomial with rational (or equivalently integer) coefficients is said to be *algebraic*. Otherwise, it is said to be *transcendental*. Since Borel's work, one thinks that base-k expansion of algebraic irrational numbers are "complex" and not much is known about their properties.

With any infinite word $w = w_1w_2\cdots$ over the alphabet of digits $A_k = \{0,\ldots,k-1\}$ is associated the real number $\sum_{i=1}^{+\infty} w_i\, k^{-i}$ in $[0,1]$. Clearly, a real number α is algebraic (over $\mathbb{Q}$) if and only if, for all $z \in \mathbb{Z}$, $\alpha + z$ is algebraic. Indeed, if α is a root of the polynomial $P(X) \in \mathbb{Q}(X)$, then $\alpha + z$ is a root of $P(X - z) \in \mathbb{Q}(X)$. Hence, we can restrict ourselves to numbers in $(0,1)$.

Transcendence of a number whose binary expansion is Sturmian has been proved in 1997 [51].

Example 7. Consider again the Fibonacci word $f = f_1f_2f_3\cdots = 010010\cdots$. The real number $\sum_{i=1}^{+\infty} f_i\, 2^{-i}$ is transcendental.

Let $k \in \mathbb{N} \setminus \{0,1\}$. The factor complexity of the k-ary expansion w of every irrational algebraic number satisfies

$$\liminf_{n\to\infty}(p_w(n) - n) = +\infty.$$

The main tool is a p-adic version of the Thue–Siegel–Roth theorem due to Ridout.

A combinatorial transcendence criterion obtained in [4] using Schmidt's subspace theorem [88] is used to obtain the following result.

Theorem 7 (Adamczewski and Bugeaud [3]). *Let $k \in \mathbb{N}\setminus\{0,1\}$. The factor complexity of the k-ary expansion w of a real irrational algebraic number satisfies*

$$\lim_{n\to+\infty} \frac{p_w(n)}{n} = +\infty.$$

Let $k \geq 2$ be an integer. If α is a real irrational number whose k-ary expansion has factor complexity in $\mathcal{O}(n)$, then α is transcendental. Since, it is well-known [37] that automatic sequences have factor complexity $p(n) \in \mathcal{O}(n)$, we can deduce that if a real irrational number has an automatic k-ary expansion, then it is transcendental.

Theorem 8 (Bugeaud and Evertse [30]). *Let $k \geq 2$ be an integer and ξ be a real irrational algebraic number with $0 < \xi < 1$. Then for any real number $\eta < 1/11$, the factor complexity $p(n)$ of the k-ary expansion of ξ satisfies*

$$\lim_{n \to +\infty} \frac{p(n)}{n(\log n)^\eta} = +\infty.$$

In [5], it is shown that the binary expansion of an algebraic number contains infinitely many occurrences of 7/3-powers. Hence the binary expansion of an algebraic number contains infinitely many overlaps.

5 Combinatorial Game Theory

Numeration systems, number theory and therefore formal language theory also have interesting connections with combinatorial game theory. In classical textbooks like [63,15] allusion to the game of Nim is made. See [16,57] for background on two player *combinatorial games*: no chance, no hidden information, same options for the two players who play alternatively, ... In particular, in removal games, we are looking for a *winning strategy* which allows a player to consummate a win regardless of the moves chosen by the other player. If such a strategy exists for given initial conditions, it is therefore natural to ask about the algorithmic complexity of the computation of the winning strategy. A first question to answer is to determine the status $\mathcal{N}$ or $\mathcal{P}$ of a given position [57].

A $\mathcal{N}$*-position*, or winning position, is a position for which a winning strategy exists for the player who starts. A $\mathcal{P}$*-position* is a position for which all options lead to a $\mathcal{N}$-position, and is thus winning for the second player[1]. In the game of Nim played on two piles of tokens, two players play alternatively and remove a positive number of tokens from one of the piles. The player removing the last token win. Otherwise stated, the first player unable to move loses (normal condition). In [12], connections between the game of Nim (values of the Sprague-Grundy function) and the notion of 2-regular functions in the sense of Allouche and Shallit is observed (finiteness of the 2-kernel). In the famous *Wythoff's game*, an extra move is allowed: removing the same positive number of tokens on both piles. The game of Nim can be easily generalized to n piles of tokens contrarily to Wythoff's game where extensions have been presented but no suitable generalization is known: for the $\mathcal{P}$-positions playing with an odd number of piles is similar to the game of Nim and playing with an even number

[1] In the game graph $\mathcal{G}$ where vertices are positions and directed edges are the allowed moves, the set of $\mathcal{P}$-positions is the kernel of $\mathcal{G}$: there is no move between any two $\mathcal{P}$-positions and from any $\mathcal{N}$-position, there exists a move to a $\mathcal{P}$-position.

of piles is hard [44,43,53,52]. In the last reference, Wythoff's game is considered as a *Prime game*. Informally, a game whose generalization to more than one or two piles seems to be very hard.

For instance, A. Fraenkel makes great use of various numeration systems to get characterizations of $\mathcal{P}$-positions [56] . As an example, in Wythoff's game, a position (x, y) is a $\mathcal{P}$-position if and only if the F-representation $\mathrm{rep}_F(x)$ ends with an even number of zeroes and $\mathrm{rep}_F(y) = \mathrm{rep}_F(x)0$ is the left shift of the first component, where F is the numeration basis given by the Fibonacci sequence from Example 2 [53]. Similarly, (x, y) is a $\mathcal{P}$-position if there exists n such that $(x, y) = (\lfloor n\alpha\rfloor, \lfloor n\alpha^2\rfloor)$ where α is the Golden ratio. So complementary Beatty sequences also enter the picture of combinatorial games [42,53,38].

In [41] moves that can be adjoined without changing the set of $\mathcal{P}$-positions are characterized using the formalism of morphisms and the fact that the computation of the successor function in the Fibonacci system is realized by a finite transducer [59]. Let $a \in \mathbb{N} \setminus \{0, 1\}$. In the parameterized version of Wythoff's game where a player can remove k tokens from one pile and ℓ from the other [53], with the condition $|k - \ell| < a$, the Ostrowski numeration system [18] based on the convergents of a continued fraction is used.

It is interesting to note that obviously the $\mathcal{P}$-positions of Wythoff's game are also characterized by the Fibonacci word introduced in Example 7. The nth $\mathcal{P}$-position is given by the pair of indices of the nth symbol 0 and nth symbol 1 occurring in the Fibonacci word. This simple observation relates combinatorial properties of morphic words like the Fibonacci or Tribonacci words to characterizations of $\mathcal{P}$-positions of games [44,43,52]. Morphic characterizations of $\mathcal{P}$-positions seems to recently raise some interest among combinatorial game theorists [52].

6 Applications for Verification of Infinite State Systems

Sets of numbers recognized by finite automata arise when analyzing systems with unbounded mixed variables taking integer or real values. Therefore are considered systems such as timed or hybrid automata [21]. One needs to develop data structures representing sets manipulated during the exploration of infinite state systems. For instance, it is often needed to compute the set of reachable configurations of such a system.

Let $k \geq 2$ be an integer. Recall that A set $X \subseteq \mathbb{R}$ is *k-recognizable* if there exists a Büchi automaton accepting all the k-representations of the elements in X. This notion extends to subsets of $\mathbb{R}^d$ and to *Real Vector Automata* or *RVA*. Also Büchi–Bruyère's Theorem giving a first order logical characterization of k-recognizable sets of integers holds in this context of real numbers for a suitable structure $\langle \mathbb{R}, \mathbb{Z}, +, 0, <, V_k\rangle$, see [25]. Roughly speaking definability in $\langle \mathbb{R}, \mathbb{Z}, +, 0, <\rangle$ of subsets of $\mathbb{R}^d$ is the natural extension of ultimately periodicity of subsets in $\mathbb{N}$.

Theorem 9. [24] *If a subset $X \subseteq \mathbb{R}^d$ is definable by a first-order formula in $\langle \mathbb{R}, \mathbb{Z}, +, 0, < \rangle$, then X written in base $k \geq 2$ is recognizable by a weak deterministic RVA $\mathcal{A}$.*

Weakness means that each strongly connected component of $\mathcal{A}$ contains only accepting states or non-accepting states.

Theorem 10. [22] *Let $k, \ell \geq 2$ be two multiplicatively independent integers. If $X \subseteq \mathbb{R}$ is both k- and ℓ-recognizable by two weak deterministic RVA, then it is definable in $\langle \mathbb{R}, \mathbb{Z}, +, 0, < \rangle$.*

The extension of Cobham–Semenov's theorem for subsets of $\mathbb{R}^d$ in this setting is discussed in [23]. The case of two coprime bases was first considered in [22]. Though written in a completely different language, a similar result was independently obtained in [1]. This latter paper is motivated by the study of some fractal sets.

Remark 4. Weak deterministic RVA have a particular interest from an algorithmic point of view. They recognize languages that are recognizable by deterministic Büchi and deterministic co-Büchi automata. For instance, minimization algorithms in $\mathcal{O}(n \log n)$ exist for this class [74].

7 Abridged Bibliographic Notes

With a gentle introduction to the logical formalism, a good way to start with integer base numeration systems is to consider [29]. Each time I come back to this very well written survey, I learn something new. Then, it is a good idea to move to the "state of the art" linear numeration basis where the characteristic polynomial of the recurrence is the minimal polynomial of a Pisot number [28]. In parallel, one should consider Frougny's chapter [76, Chap. 7] and her very interesting work on the normalization map [58] and beta-expansions [60]. As a good textbook on some of the aspects presented here, consider [12]. The original paper of Cobham [37] is also worth of reading. For some general surveys on factor complexity and the Thue–Morse word, without any required background, see [8,11].

Then I cannot resist advertising [19] where in the spirit of Lothaire's series, we try to present the fruitful links existing between combinatorics on words, automata theory and number theory. It presents in a self-contained expository book much more material than is presented in this survey (ergodic theory, Rauzy fractal, joint spectral radius,...).

For a list of pointers on Cobham's theorem in various contexts, see [48] for an updated survey. Accounts of Perron–Frobenius theory can be found in many classical textbooks, but probably [73] is worth reading.

Connections between symbolic dynamics and formal language theory are fruitful: for the reader with no background in dynamics (for instance, no knowledge in measure theory is required) and on a very introductory level, consider [90]. Then, move to the survey [13] and [82].

8 Some Open Problems

We conclude with some general (and probably quite hard) open problems.

– As mentioned in Section 3, the most general version of Cobham's theorem still relies on some mild assumptions about the considered morphisms (details are not given in this survey). F. Durand refers to these as "*good substitutions*". One could hope to relax these hypotheses and still get the same result with full generality [48]. Up to now there is no proof of a Cobham-like theorem for a substitution having no main sub-substitution having the same dominating eigenvalue like $a \mapsto aa0$, $0 \mapsto 01$ and $1 \mapsto 0$. In this latter example, the dominating eigenvalue is 2 but the substitution restricted to the alphabet $\{0, 1\}$ has $(1+\sqrt{5})/2$ as dominating eigenvalue.
– Come back again to Cobham's theorem but this time for Gaussian integers $\mathbb{G} = \{a + ib \mid a, b \in \mathbb{Z}\}$. Indeed, these numbers have nice representations using the so-called canonical number systems [68]. For canonical numeration systems in algebraic number fields, every integer has a unique finite expansion which is computed starting with the least significant digit first. A Cobham-like conjecture for Gaussian integers [62] is related to the famous Four Exponentials conjecture: *let* $\{\lambda_1, \lambda_2\}$ *and* $\{x_1, x_2\}$ *be two pairs of rationally independent complex numbers. Then, one of the numbers* $e^{\lambda_1 x_1}$, $e^{\lambda_1 x_2}$, $e^{\lambda_2 x_1}$, $e^{\lambda_2 x_2}$ *is transcendental*, for instance see [91].
– The philosophy of Cobham's theorem also appears when considering *self-generating sets* as introduced by Kimberling [69]. For instance, consider the affine maps $f : \mathbb{N} \to \mathbb{N}, x \mapsto 2x + 1$ and $g : \mathbb{N} \to \mathbb{N}, x \mapsto 4x + 2$. A self-generating set obtained from f and g can be defined as the smallest subset S of $\mathbb{N}$ containing 0 and such that $f(S) \subset S$ and $g(S) \subset S$. In our example, the first few elements in S are

$$0, 1, 2, 3, 5, 6, 7, 10, 11, 13, 14, 15, 21, 23, 26, 27, 29, 30, 31, 42, 43, 47, 53, \ldots.$$

One can therefore study the k-recognizability of S. If one considers maps where the multiplicative constants are multiplicatively independent, then Allouche, Shallit and Skordev conjectured that the corresponding set cannot be k-recognizable [10]. With some technical hypothesis about the multiplicative coefficients when there are at least three affine maps, this conjecture has been proved to be true in [67]. One could hope to prove this conjecture in full generality. A possible connection with smooth numbers (having only small prime factors in their decomposition) has been pointed out by J. Shallit.
– In combinatorial game theory the *Sprague-Grundy function* g is of great interest. For instance, the positions for which g vanishes are exactly the $\mathcal{P}$-position of the game and when considering *sums of games* (several games are played simultaneously and at each turn, the player chooses on which of those games he will made a move), it can be used to distinguish $\mathcal{N}$-positions [16]. For Wythoff's game, little is known about this function (see for instance [55]) even if its recursive definition is simple. The value of $g(x, y)$ is the minimum excluded value (Mex) of the set of $g(u, v)$ where (u, v) is ranging

amongst the options reachable from (x, y). By definition $\operatorname{Mex}\emptyset = 0$ and $\operatorname{Mex} S = \min(\mathbb{N} \setminus S)$ for all finite set S.

	0	1	2	3	4	5	6	7	8	9	$\cdots$
0	0	1	2	3	4	5	6	7	8	9	$\cdots$
1	1	2	0	4	5	3	7	8	6	10	
2	2	0	1	5	3	4	8	6	7	11	
3	3	4	5	6	2	0	1	9	10	12	
4	4	5	3	2	7	6	9	0	1	8	
5	5	3	4	0	6	8	10	1	2	7	
$\vdots$											$\ddots$

Let F be the Fibonacci numeration basis. As suggested by the developments considered in [41] could the above infinite array reveal some morphic structure, like having a finite F-kernel where this set could be defined as the set of subarrays

$$(g(x, y))_{\mathrm{rep}_F(x)\in\{0,1\}^*u,\ \mathrm{rep}_F(y)\in\{0,1\}^*v}$$

for given suffixes u, v? For the generalization of k-kernel, see for instance [83].

- Theorem 10 is a Cobham-like theorem for sets of real numbers, definability in $\langle \mathbb{R}, \mathbb{Z}, +, 0, < \rangle$ being the counterpart to ultimate periodicity of a set of integers. Can a simpler proof of this result be achieved, for instance by considering the techniques developed in [1]? Also could this result be extended to other kind of representations of real numbers. For instance, considering β-expansions of real numbers, we could define β-recognizable sets of real numbers and consider two multiplicatively independent real numbers $\alpha, \beta > 1$. As a first step (and to mimic what has chronologically been done for sets of integers), one could consider a set of real numbers $X \subseteq \mathbb{R}$ which is both k-recognizable and β-recognizable by two weak deterministic RVA, with $k \geq 2$ an integer and β a Pisot number like the Golden ratio, and ask is X definable in $\langle \mathbb{R}, \mathbb{Z}, +, 0, < \rangle$?
- About abstract numeration systems, several questions about S-recognizable sets are open. For instance, is there some reasonable logical characterization of the S-recognizable sets of integers which could be compared to the characterization in the extended Presburger arithmetic $\langle \mathbb{N}, +, V_k \rangle$. But one can notice that in general, if X and Y are S-recognizable, there is no reason to have a S-recognizable set $X + Y$ (even when considering multiplication by a constant). Another question is to relate the growth function $n \mapsto \#(L \cap A^n)$ of the regular language L on which the abstract numeration system S is based and the S-recognizable set. For instance, if $P \in \mathbb{N}[X]$ is a polynomial such that $P(\mathbb{N})$ is S-recognizable, what can be said about the growth function of the language of numeration. Results like the one found in [14] could be of interest.
- Recently numeration systems based on the powers of a rational number have been introduced [6] (motivated by a number theoretic question from Mahler).

These numerations reveal interesting and intriguing properties. For instance, little is known about the properties of the language of numeration $L_{3/2}$. For a given prefix w, compute the number of words of length n in the quotient $w^{-1}L_{3/2}$.

– It is well-known since the work of Cobham [35] that a morphic infinite word $w = \tau(\sigma^\omega(a))$ where σ and τ are arbitrary morphisms (where both morphisms can be erasing and τ is not necessarily a coding) can be generated by a non-erasing morphism μ and a coding ν. See for instance [12] for a comprehensive proof or [66] for an alternative presentation. All the known proofs rely on morphisms and are quite long: could one describe in the formalism of automata theory a somehow simpler proof?
– Let me also mention Hollander's conjecture when for a linear numeration basis U, the dominant root condition is not satisfied [65]. He has conjectured that $\text{rep}_U(\mathbb{N})$ can be regular only if there exists n such that

$$\lim_{j\to\infty} U_{jn+k}/U_{(j-1)n+k}$$

exists and is independent of k, and the characteristic polynomial $p(X)$ of U is such that $p(X) = q(X^n)$ where $q(X)$ is the minimal polynomial for a recurrence which gives a regular language [26].
– Let p be a prime. Derksen proved that the zero set of a linear recurrence over a field of characteristic p is p-automatic [40,2]. Could such a result and Cobham's theorem be used to get back the classical Skolem–Mahler–Lech theorem (the zero set of a linear recurrence over a field of characteristic 0 is ultimately periodic)?
– The reader fond of logic could also look back at the list of open problems given by Michaux and Villemaire [78]. This survey paper is devoted to problems related to Büchi's characterization of sets of natural numbers recognizable by finite automata in base k, as well as to Cobham's and Semenov's extensions of it.

Acknowledgments

I would like to thank Boris Adamczewski, Valérie Berthé, Véronique Bruyère, Eric Duchêne, Narad Rampersad for the careful reading of a first draft of this paper. They provided me useful suggestions to improve this text.

References

1. Adamczewski, B., Bell, J.P.: An analogue of Cobham's theorem for fractals. Trans. Amer. math. Soc. (to appear)
2. Adamczewski, B., Bell, J.P.: Automata in number theory. In: Pin, J.-E. (ed.) Handbook, Automata: from Mathematics to Applications. Europ. Math. Soc. Publishing House (to appear)
3. Adamczewski, B., Bugeaud, Y.: On the complexity of algebraic numbers I: Expansion in integer bases. Ann. of Math. 165, 547–565 (2007)

4. Adamczewski, B., Bugeaud, Y., Luca, F.: Sur la complexité des nombres algébriques. C. R. Math. Acad. Sci. Paris 339, 11–14 (2004)
5. Adamczewski, B., Rampersad, N.: On patterns occurring in binary algebraic numbers. Proc. Amer. Math. Soc. 136, 3105–3109 (2008)
6. Akiyama, S., Frougny, C., Sakarovitch, J.: Powers of rationals modulo 1 and rational base number systems. Israel J. Math. 168, 53–91 (2008)
7. Alexeev, B.: Minimal DFA for testing divisibility. J. Comput. System Sci. 69(2), 235–243 (2004)
8. Allouche, J.-P.: Sur la complexité des suites infinies. Bull. Belg. Math. Soc. 1, 133–143 (1994)
9. Allouche, J.-P., Cateland, E., Gilbert, W.J., Peitgen, H.-O., Shallit, J., Skordev, G.: Automatic maps in exotic numeration systems. Theory Comput. Systems 30, 285–331 (1997)
10. Allouche, J.-P., Shallit, J., Skordev, G.: Self-generating sets, integers with missing blocks, and substitutions. Discrete Math. 292(1-3), 1–15 (2005)
11. Allouche, J.-P., Shallit, J.O.: The ubiquitous Prouhet-Thue-Morse sequence. In: Ding, C., Helleseth, T., Niederreiter, H. (eds.) Sequences and Their Applications, Proceedings of SETA 1998, pp. 1–16. Springer, Heidelberg (1999)
12. Allouche, J.-P., Shallit, J.O.: Automatic Sequences, Theory, Applications, Generalizations. Cambridge University Press, Cambridge (2003)
13. Barat, G., Berthé, V., Liardet, P., Thuswaldner, J.: Dynamical directions in numeration. Ann. Inst. Fourier (Grenoble) 56, 1987–2092 (2006)
14. Béal, M.-P., Lombardy, S., Sakarovitch, J.: Conjugacy and equivalence if weighted automata and functional transducers. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 58–69. Springer, Heidelberg (2006)
15. Berge, C.: Graphes et hypergraphes. Dunod, Paris, Deuxième édition, Collection Dunod Université, Série Violette, No. 604 (1973)
16. Berlekamp, E.R., Conway, J.H., Guy, R.K.: Winning ways for your mathematical plays, 2nd edn., vol. 1. A.K. Peters Ltd., Natick (2001)
17. Berstel, J., Karhumäki, J.: Combinatorics on words: a tutorial. Bull. European Assoc. Theor. Comput. Sci. 79, 178–228 (2003)
18. Berthé, V.: Autour du système de numération d'Ostrowski. Bull. Belg. Math. Soc. Simon Stevin 8, 209–239 (2001)
19. Berthé, V., Rigo, M. (eds.): Combinatorics, Automata and Number Theory. Encyclopedia of Mathematics and its Applications, vol. 135. Cambridge University Press, Cambridge (2010)
20. Berthé, V., Siegel, A.: Tilings associated with beta-numeration and substitutions. Integers 5(3):A2, 46 (2005) (electronic)
21. 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)
22. Boigelot, B., Brusten, J.: A generalization of Cobham's theorem to automata over real numbers. Theoret. Comput. Sci. 410(18), 1694–1703 (2009)
23. Boigelot, B., Brusten, J., Leroux, J.: A generalization of Semenov's theorem to automata over real numbers. In: Schmidt, R.A. (ed.) Automated Deduction – CADE-22. LNCS, vol. 5663, pp. 469–484. Springer, Heidelberg (2009)
24. Boigelot, B., Jodogne, S., Wolper, P.: An effective decision procedure for linear arithmetic over the integers and reals. ACM Trans. Comput. Log. 6(3), 614–633 (2005)

25. Boigelot, B., Rassart, S., Wolper, P.: On the expressiveness of real and integer arithmetic automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 152–163. Springer, Heidelberg (1998)
26. Boyd, D.W.: Irreducible polynomials with many roots of maximal modulus. Acta Arith. 68(1), 85–88 (1994)
27. Brlek, S.: Enumeration of factors in the Thue-Morse word. Discrete Appl. Math. 24, 83–96 (1989)
28. Bruyère, V., Hansel, G.: Bertrand numeration systems and recognizability. Theoret. Comput. Sci. 181, 17–43 (1997)
29. Bruyère, V., Hansel, G., Michaux, C., Villemaire, R.: Logic and p-recognizable sets of integers. Bull. Belg. Math. Soc. 1, 191–238 (1994)
30. Bugeaud, Y., Evertse, J.-H.: On two notions of complexity of algebraic numbers. Acta Arith. 133, 221–250 (2008)
31. Charlier, E., Kärki, T., Rigo, M.: Multidimensional generalized automatic sequences and shape-symmetric morphic words. Discrete Math. 310, 1238–1252 (2010)
32. Charlier, E., Rampersad, N., Rigo, M., Waxweiler, L.: State complexity of testing divisibility (2010) (preprint)
33. Charlier, E., Rigo, M., Steiner, W.: Abstract numeration systems on bounded languages and multiplication by a constant. Integers 8(A35), 19 (2008)
34. Choffrut, C., Karhumäki, J.: Combinatorics of words. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of formal languages, Word, Language, Grammar, vol. 1, pp. 329–438. Springer, New York (1997)
35. Cobham, A.: On the Hartmanis-Stearns problem for a class of tag machines. In: IEEE Conference Record of 1968 Ninth Annual Symposium on Switching and Automata Theory, pp. 51–60 (1968)
36. Cobham, A.: On the base-dependence of sets of numbers recognizable by finite automata. Math. Systems Theory 3, 186–192 (1969)
37. Cobham, A.: Uniform tag sequences. Math. Systems Theory 6, 164–192 (1972)
38. Crisp, D., Moran, W., Pollington, A., Shiue, P.: Substitution invariant cutting sequences. J. Théor. Nombres Bordeaux 5(1), 123–137 (1993)
39. de Luca, A., Varricchio, S.: On the factors of the Thue-Morse word on three symbols. Inform. Process. Lett. 27, 281–285 (1988)
40. Derksen, H.: A Skolem-Mahler-Lech theorem in positive characteristic and finite automata. Invent. Math. 168, 175–224 (2007)
41. Duchêne, E., Fraenkel, A.S., Nowakowski, R., Rigo, M.: Extensions and restrictions of Wythoff's game preserving Wythoff's sequence as set of P-positions. J. Combinat. Theory Ser. A 117, 545–567 (2010)
42. Duchêne, E., Rigo, M.: Invariant games. Theoret. Comput. Sci. (to appear)
43. Duchêne, E., Rigo, M.: Cubic Pisot unit combinatorial games. Monat. für Math. 155, 217–249 (2008)
44. Duchêne, E., Rigo, M.: A morphic approach to combinatorial games: the tribonacci case. Theor. Inform. Appl. 42, 375–393 (2008)
45. Durand, F.: A generalization of Cobham's theorem. Theory Comput. Syst. 31(2), 169–185 (1998)
46. Durand, F.: Sur les ensembles d'entiers reconnaissables. J. Théor. Nombres Bordeaux 10(1), 65–84 (1998)
47. Durand, F.: A theorem of Cobham for non primitive substitutions. Acta Arith. 104, 225–241 (2002)
48. Durand, F., Rigo, M.: On Cobham's theorem. In: Handbook, Automata: from Mathematics to Applications. Europ. Math. Soc. Publishing House (to appear)

49. Durand, F., Rigo, M.: Syndeticity and independent substitutions. Adv. in Appl. Math. 42, 1–22 (2009)
50. Eilenberg, S.: Automata, Languages, and Machines, vol. A. Academic Press, London (1974)
51. Ferenczi, S., Mauduit, C.: Transcendence of numbers with a low complexity expansion. J. Number Theory 67, 146–161 (1997)
52. Fraenkel, A.S.: Complementary iterated floor words and the flora game. SIAM J. on Discrete Math. (to appear)
53. Fraenkel, A.S.: How to beat your Wythoff games' opponent on three fronts. Amer. Math. Monthly 89, 353–361 (1982)
54. Fraenkel, A.S.: Systems of numeration. Amer. Math. Monthly 92, 105–114 (1985)
55. Fraenkel, A.S.: The Sprague-Grundy function for Wythoff's game. Theoret. Comput. Sci. 75, 311–333 (1990)
56. Fraenkel, A.S.: Heap games, numeration systems and sequences. Annals of Combin. 2, 197–210 (1998)
57. Fraenkel, A.S.: Complexity, appeal and challenges of combinatorial games. Theoret. Comput. Sci. 313(3), 393–415 (2004)
58. Frougny, C.: Representations of numbers and finite automata. Math. Systems Theory 25, 37–60 (1992)
59. Frougny, C.: On the sequentiality of the successor function. Inform. and Comput. 139(1), 17–38 (1997)
60. Frougny, C., Solomyak, B.: Finite beta-expansions. Ergod. Th. & Dynam. Sys. 12, 713–723 (1992)
61. Grabner, P., Heuberger, C., Prodinger, H.: Distribution results for low-weight binary representations for pairs of integers. Theoret. Comput. Sci. 319, 307–331 (2004)
62. Hansel, G., Safer, T.: Vers un théorème de Cobham pour les entiers de Gauss. Bull. Belg. Math. Soc. Simon Stevin 10, 723–735 (2003)
63. Hardy, G.H., Wright, E.M.: An Introduction to the Theory of Numbers, 5th edn. Oxford University Press, Oxford (1985)
64. Hartmanis, J., Stearns, R.E.: On the computational complexity of algorithms. Trans. Amer. Math. Soc. 117, 285–306 (1965)
65. Hollander, M.: Greedy numeration systems and regularity. Theory Comput. Systems 31, 111–133 (1998)
66. Honkala, J.: On the simplification of infinite morphic words. Theoret. Comput. Sci. 410(8-10), 997–1000 (2009)
67. Kärki, T., Lacroix, A., Rigo, M.: On the recognizability of self-generating sets. J. Integer Sequences, Article 10.2.2 (2010)
68. Kátai, I., Szabó, J.: Canonical number systems for complex integers. Acta Sci. Math (Szeged) 37(3-4), 255–260 (1975)
69. Kimberling, C.: A self-generating set and the golden mean. J. Integer Seq., Article 00.2.8, 3(2) (2000)
70. Knuth, D.E.: The art of computer programming. In: Seminumerical algorithms, 2nd edn. Addison-Wesley Series in Computer Science and Information Processing, vol. 2, Addison-Wesley Publishing Co., Reading (1981)
71. Lecomte, P.B.A., Rigo, M.: Numeration systems on a regular language. Theory Comput. Systems 34, 27–44 (2001)
72. Lenstra, H.: Profinite Fibonacci numbers. Nieuw. Arch. Wiskd. (5) 6(4), 297–300 (2005); Also in Europ. Math. Soc. Newsletter 61, 15–18 (September 2006)
73. Lind, D., Marcus, B.: An Introduction to Symbolic Dynamics and Coding. Cambridge University Press, Cambridge (1995)

74. Löding, C.: Efficient minimization of deterministic weak ω-automata. Inform. Process. Lett. 79(3), 105–109 (2001)
75. Loraud, N.: β-shift, systèmes de numération et automates. J. Théorie Nombres Bordeaux 7, 473–498 (1995)
76. Lothaire, M.: Algebraic Combinatorics on Words. Encyclopedia of Mathematics and Its Applications, vol. 90. Cambridge University Press, Cambridge (2002)
77. Lothaire, M.: Applied Combinatorics on Words. Encyclopedia of Mathematics and Its Applications, vol. 105. Cambridge University Press, Cambridge (2005)
78. Michaux, C., Villemaire, R.: Open questions around Büchi and Presburger arithmetics. In: Logic: from Foundations to Applications (Staffordshire, 1993), pp. 353–383. Oxford Sci. Publ., New York (1996)
79. Pansiot, J.-J.: Complexité des facteurs des mots infinis engendrés par morphismes itérés. In: ICALP 1984. LNCS, vol. 172, pp. 380–389. Springer, Berlin (1984)
80. Parry, W.: On the β-expansions of real numbers. Acta Math. Acad. Sci. Hung. 11, 401–416 (1960)
81. Perrin, D.: Finite automata. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science. Formal Models and Semantics, vol. B, pp. 1–57. Elsevier/MIT Press (1990)
82. Pytheas Fogg, N.: Substitutions in Dynamics, Arithmetics and Combinatorics. In: Berthé, V., Ferenczi, S., Mauduit, C., Siegel, A. (eds.). LNM, vol. 1794. Springer, Heidelberg (2002)
83. Rigo, M., Maes, A.: More on generalized automatic sequences. J. Automata, Languages, and Combinatorics 7, 351–376 (2002)
84. Rigo, M., Waxweiler, L.: A note on syndeticity, recognizable sets and Cobham's theorem. Bull. European Assoc. Theor. Comput. Sci. 88, 169–173 (2006)
85. Sakarovitch, J.: Éléments de théorie des automates. Vuibert, English corrected edition: Elements of Automata Theory. Cambridge University Press, Cambridge (2003/2009)
86. Salon, O.: Suites automatiques à multi-indices. In: Séminaire de Théorie des Nombres de Bordeaux, pp. 4.01–4.27 (1986-1987)
87. Salon, O.: Suites automatiques à multi-indices et algébricité. C. R. Acad. Sci. Paris 305, 501–504 (1987)
88. Schmidt, W.M.: Diophantine Approximation. LNM, vol. 785. Springer, Heidelberg (1980)
89. Shallit, J.O.: Numeration systems, linear recurrences, and regular sets. Inform. Comput. 113, 331–347 (1994)
90. Silva, C.E.: Invitation to ergodic theory. Student Mathematical Library, vol. 42. American Mathematical Society, Providence (2008)
91. Waldschmidt, M.: Diophantine approximation on linear algebraic groups. Grundlehren der Mathematischen Wissenschaften, vol. 326. Springer, Berlin (2000); Transcendence properties of the exponential function in several variables
92. Waldschmidt, M.: Words and transcendence. In: Analytic Number Theory - Essays in Honour of Klaus Roth, ch. 31, pp. 449–470. Cambridge University Press, Cambridge (2009), http://arxiv.org/abs/0908.4034

Algorithmic Properties of Millstream Systems

Suna Bensch, Henrik Björklund, and Frank Drewes

Department of Computing Science, Umeå University
90187 Umeå, Sweden
{suna,henrikb,drewes}@cs.umu.se

Abstract. Millstream systems have recently been proposed as a formalization of the linguistic idea that natural language should be described as a combination of different modules related by interfaces. In this paper we investigate algorithmic properties of Millstream systems having regular tree grammars as modules and MSO logic as interface logic. We focus on the so-called completion problem: Given trees generated by a subset of the modules, can they be completed into a valid configuration of the Millstream system?

1 Introduction

Millstream systems [1] have recently been introduced as a generic mathematical framework for the description of natural language providing the possibility to formalize and reason about the relation between different linguistic levels, such as phonology, morphology, syntax and semantics. Millstream systems are motivated by contemporary linguistic theories that refrain from the idea of transformational grammars in the Chomskian tradition in which linguistic levels are hierarchically ordered. The authors in [11,7], for example, propose to view them as autonomous modules that work simultaneously but are linked with each other through interfaces that describe the interactions and interdependencies between these linguistic levels. Both authors argue that the human way of processing language is more adequately described by such a non-hierarchical approach, as the human brain seems to store and process different linguistical levels in parallel, at the same time linking them according to certain rules in order to create a whole that is more than the sum of its parts. Matching this view, a Millstream system consists of several individual *modules* specifying tree languages $L_1, \ldots, L_k$, and a logical *interface* relating the (trees yielded by the) modules. A configuration is a tuple $(t_1, \ldots, t_k) \in L_1 \times \cdots \times L_k$, augmented with links as specified by the interface.

Let us consider an example that illustrates the linguistic ideas that have motivated Millstream systems. Figure 1 shows the syntactic and semantic structure, depicted as trees (a) and (b), respectively, of the sentence *John loves Sarah* and the established interface links which are depicted as dotted lines linking syntactic categories occurring in structure (a) with semantic categories occurring in structure (b). The syntactic tree (a) divides the sentence S into a nominal phrase $\mathrm{NP_S}$ and a verbal phrase VP. The nominal phrase $\mathrm{NP_S}$ consists of the lexical

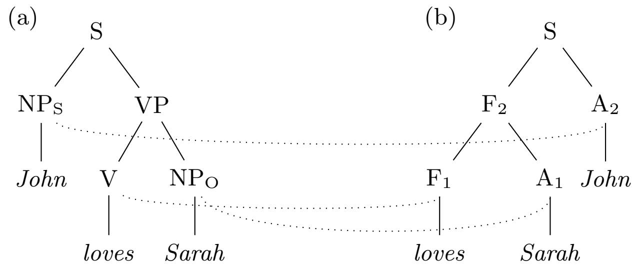


Fig. 1. Syntactical and semantic structures of *John loves Sarah*

item *John* and the verbal phrase VP is divided into a verb V and a nominal phrase $\mathrm{NP_O}$, where V and $\mathrm{NP_O}$ consist of the lexical items *loves* and *Sarah*, respectively. The semantic tree (b) depicts that the transitive verb *loves* is of category $\mathrm{F_1}$ which represents a function that is applied to the argument $\mathrm{A_1}$ (the object) whose lexical item is *Sarah* and yields a function as its result, namely $\mathrm{F_2}$. Function $\mathrm{F_2}$ in turn is applied to the argument $\mathrm{A_2}$ (the subject) whose lexical item is *John*. Thus, the analysis of the sentence *John loves Sarah* does not only result in its syntactic and semantic trees, but also includes relationships between them, namely the links in Figure 1. The syntactic category V, for example, is linked with the semantic category F_1 which illustrates that these particular occurrences of V and F_1 correspond to each other. The syntactic subject $\mathrm{NP_S}$ and the object $\mathrm{NP_O}$ are linked with the semantic arguments A_2 and A_1, respectively, which reflects that the syntactic arguments (subject and object) of a (transitive) verb correspond to the semantic arguments of the semantic function of that verb. The conditions that such links have to fulfill are described by the interface. The reader is referred to [12] for a discussion of the syntax-semantics interface from the linguistic point of view and to [1] for more a detailed discussion of how this can be formalized in terms of Millstream systems.

In particular, Millstream systems are of interest for natural language processing, and in particular for natural language understanding and natural language generation. Simply put, the task of natural language understanding is to construct a suitable semantic representation of a sentence that has been heard (phonology) and parsed (syntax). Within the framework of Millstream systems this corresponds to the problem where we are given a syntactic tree (and possibly a phonological tree if a phonological module is involved) and the goal is to construct an appropriate semantic tree. Conversely, natural language generation can be seen as the problem to construct an appropriate syntactic (and/or phonological) tree from a given semantic tree.

In abstract terms, the situations just described are identical. In both cases, a Millstream system is given, the input is a partial configuration consisting of some of the required trees, and the goal is to complete the configuration by adding the missing trees and the links between the trees. In this paper, we study whether and how this problem, called the completion problem, can be solved for so-called regular MSO Millstream systems, i. e. systems in which the modules are regular tree grammars (or, equivalently, finite tree automata) and the interface conditions are expressed in monadic second-order (MSO) logic. We prove that the emptiness problem (where no tree of the configuration is known) is undecidable, but the completion problem is decidable if no direct links exist between the unknown trees. Finally, motivated by the observation that the completion problem is decidable if the configurations are of bounded tree width, we establish sufficient conditions under which this is the case. Moreover, structures of bounded tree-width seem to be of particular interest for natural language processing. For example, Kornai and Tuza [8] argue that bounded path-width is related to the bounded capacity of the short-term memory and its influence on human generation and understanding of language.

The rest of this paper is organized as follows. The next section contains the definition of Millstream systems and other basic notions. In Section 3, the undecidability of the emptiness problem is shown. Section 4 contains the proof that the completion problem is decidable in those cases where there are no direct links between unknown parts of the configuration, and in Section 5 sufficient conditions for bounding the tree width of configurations are studied. To obey the page limit, some proofs have been moved into the appendix.

2 Definitions and Preliminaries

The set of natural numbers is denoted by $\mathbb{N}$, and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. For $k \in \mathbb{N}$, we let $[k] = \{1, \dots, k\}$. For a set S, the set of all nonempty finite sequences (or strings) over S is denoted by S^+; if the empty sequence ϵ is included, we write S^*.

A *ranked alphabet* is a finite set Σ of pairs (f, k), where f is a symbol and $k \in \mathbb{N}$ is its *rank*. We denote (f, k) by $f^{(k)}$, or simply by f if k is understood.

We define trees over Σ in one of the standard ways, by identifying the nodes of a tree t with sequences of natural numbers. Thus, T_Σ consists of all mappings $t \colon V(t) \to \Sigma$ (called trees), such that the set $V(t)$ of *nodes* of t is a finite and non-empty prefix-closed subset of $\mathbb{N}_+^*$, and, for every node $v \in V(t)$, if $t(v) = f^{(k)}$, then $\{i \in \mathbb{N} \mid vi \in V(t)\} = [k]$. In other words, the *children* of v are $v1, \dots, vk$ (and v is their *parent*). For $v, u \in V(t)$ we write $v \leq u$ if v is a prefix of u and $v < u$ if v is a proper prefix. A *Σ-tree generator* is any kind of device G that specifies a tree language $L(G) \subseteq \mathrm{T}_\Sigma$. We assume familiarity with generators of regular tree languages, e.g., finite tree automata, regular tree grammars, and monadic second-order logic (MSO) (see, e.g., [6]).

For a tree $t \in \mathrm{T}_\Sigma$, the subtree of t rooted at v (defined in the usual way) is denoted by t/v. We denote a tree t as $f[t_1, \dots, t_k]$ if $t(\epsilon) = f^{(k)}$ and $t/i = t_i$ for $i \in [k]$, omitting the brackets if $k = 0$.

For a tuple $T \in \mathrm{T}_\Sigma^k$, we let $V(T)$ denote the set $\{(i, v) \mid i \in [k] \text{ and } v \in V(t_i)\}$. Thus, $V(T)$ is the disjoint union of the sets $V(t_i)$. Furthermore, we let $V(T, i)$ denote its ith component, i.e., $V(T, i) = \{i\} \times V(t_i)$ for all $i \in [k]$.

We now define a logical representation of trees, which is fairly standard (see, e.g., [9]). For this, let Λ be any type of predicate logic that allows us to make use of n-ary predicate symbols $P^{(n)}$, and let F_Λ denote the set of all formulas in Λ without free variables (i.e., the set of sentences of Λ). For $k \in \mathbb{N}$, we say that a predicate symbol $P^{(n)}$ is k-typed if it comes with an associated type $(i_1, \ldots, i_n) \in [k]^n$. We write $P\colon i_1 \times \cdots \times i_n$ to specify the type of P.

Given a (finite) set $\mathcal{P}$ of predicate symbols, a logical structure $\langle D; (\psi_P)_{P \in \mathcal{P}} \rangle$ consists of a set D called the domain and, for each $P^{(n)} \in \mathcal{P}$, a predicate $\psi_P \subseteq D^n$. If an existing structure Z is enriched with additional predicates $(\psi_P)_{P \in \mathcal{P}'}$ (where $\mathcal{P} \cap \mathcal{P}' = \emptyset$), we denote the resulting structure by $\langle Z; (\psi_P)_{P \in \mathcal{P}'} \rangle$. In this paper, we will only consider structures with finite domains.

A tuple $T = (t_1, \ldots, t_k) \in \mathrm{T}_\Sigma^k$ of trees will be represented by the structure

$$|T| = \langle V(T); (V_i)_{i \in [k]}, (\mathrm{lab}_g)_{g \in \Sigma}, (\downarrow_i)_{i \in [r]} \rangle,$$

using the following predicates $V_i^{(1)}$ $(i \in [k])$, $\mathrm{lab}_g^{(1)}$ $(g \in \Sigma)$ and $\downarrow_i^{(2)}$ $(i \in [r])$:

- For every $i \in [k]$, $V_i = V(T, i)$. Thus, $V_i(d)$ expresses that d belongs to t_i.
- For every $g \in \Sigma$, $\mathrm{lab}_g = \{(i, v) \in V(T) \mid i \in [k] \text{ and } t_i(v) = g\}$. Thus, $\mathrm{lab}_g(d)$ expresses that the label of d is g.
- For every $j \in [r]$, $\downarrow_j = \{((i, v), (i, vj)) \mid i \in [k] \text{ and } v, vj \in V(t_i)\}$. Thus, $\downarrow_j(d, d')$ expresses that d' is the jth child of d in one of the trees $t_1, \ldots, t_k$. In the following, we write $d \downarrow_j d'$ instead of $\downarrow_j(d, d')$.

Note that, in the definition of $|T|$, we have blurred the distinction between predicate symbols and their interpretation as predicates, because this interpretation is fixed. In the following, especially in intuitive explanations, we shall sometimes also identify the logical structure $|T|$ with the tuple T it represents.

To define Millstream systems, we first formalize our notion of interfaces. The idea is that a tuple $T = (t_1, \ldots, t_k)$ of trees, represented as $|T|$, is augmented with additional *interface links* that are subject to logical conditions.

Definition 1 (Interface). *Let Σ be a ranked alphabet. An* interface on T_Σ^k *($k \in \mathbb{N}$) is a pair $INT = (\mathcal{I}, \Phi)$, such that*

- *$\mathcal{I}$ is a finite set of k-typed predicate symbols called* interface symbols*, and*
- *Φ is a finite set of formulas in F_Λ that may, in addition to the fixed vocabulary of Λ, contain the predicate symbols in $\mathcal{I}$ and those occurring in the structures $|T|$ (where $T \in \mathrm{T}_\Sigma^k$). These formulas are called* interface conditions.

A configuration *(w.r.t. INT) is a structure $C = \langle |T|; (\psi_I)_{I \in \mathcal{I}} \rangle$ with $T \subseteq \mathrm{T}_\Sigma^k$ such that $\psi_I \subseteq V(T, i_1) \times \cdots \times V(T, i_l)$ for each $I\colon i_1 \times \cdots \times i_l$ in $\mathcal{I}$, and C satisfies the interface conditions in Φ (if each $I \in \mathcal{I}$ is interpreted as ψ_I).*

For $I \in \mathcal{I}$ and nodes $v_1, \ldots, v_l$, we say that a configuration C as above contains the link $I(v_1, \ldots, v_l)$ if $(v_1, \ldots, v_l) \in \psi_I$.

Definition 2 (Millstream system). *Let Σ be a ranked alphabet and $k \in \mathbb{N}$. A* Millstream system *(MS, for short) is a system $MS = (M_1, \ldots, M_k; INT)$ consisting of Σ-tree generators $M_1, \ldots, M_k$, called the* modules *of MS, and an interface INT on T_Σ^k. The language $L(MS)$ generated by MS is the set of all configurations $\langle |T|; (\psi_I)_{I \in \mathcal{I}} \rangle$ such that $T \in L(M_1) \times \cdots \times L(M_k)$.*

In the remainder of the paper, we mainly consider *regular MSO Millstream systems*. This is the special case of Millstream systems where the modules are regular tree grammars (or equivalent devices) and Λ is monadic second-order predicate logic. Similarly, we talk about regular FO Millstream systems if only first-order predicate logic is considered. If we do not want to restrict the type of modules considered, we speak of Λ Millstream systems. Recall that, in fact, the regular tree languages are exactly those which can be specified by MSO formulas over trees, which means that each component language $L(M_i)$ of a regular MSO Millstream system can, if convenient, be assumed to be equal to T_Σ.

3 Emptiness for Millstream Systems Is Undecidable

In this section we show that it is undecidable whether the language of a given Millstream system is empty. More precisely, we show the following.

Theorem 3. *Emptiness for regular FO Millstream systems is undecidable.*

Proof sketch. The proof is by reduction from Post's correspondence problem (PCP), which is well known to be undecidable [10]. An instance of PCP over a finite alphabet Σ is a set $I = \{(u_1, w_1), \ldots, (u_n, w_n)\}$ of pairs of words over Σ, i.e., $u_i, w_i \in \Sigma^*$ for $i = 1, \ldots, n$. The question is whether there exists an integer $m \in \mathbb{N}$ and a sequence of indices $i_1, i_2, \ldots, i_m \in \{1, \ldots, n\}$ such that

$$u_{i_1} u_{i_2} \cdots u_{i_m} = w_{i_1} w_{i_2} \cdots w_{i_m}. \tag{1}$$

For an instance $I = \{(u_1, w_1), \ldots, (u_n, w_n)\}$ of PCP over $\Sigma = \{a, b\}$, we show how to construct a regular FO Millstream system $MS_I = (M_1, M_2; INT)$ such that $L(MS_I) \neq \emptyset$ if and only if I has a solution.

The modules M_1 and M_2 generate monadic trees over the ranked alphabet $\Gamma = \{a^{(1)}, b^{(1)}, 1^{(1)}, 2^{(1)}, \ldots, n^{(1)}, \Diamond^{(0)}\}$. Thus, we may view $L(M_1)$ and $L(M_2)$ as regular string languages. The idea is that these languages contain all possible indexed left- and right-hand sides of solutions to Equation (1). To be precise, we let $L(M_1) = (1 \cdot u_1 + \cdots + n \cdot u_n)^* \Diamond$ and $L(M_2) = (1 \cdot w_1 + \cdots + n \cdot w_n)^* \Diamond$.

The interface INT, has two interface symbols, $Link_1$ and $Link_2$, and is used to ensure that only pairs of trees that satisfy Equation 1 can be produced. For instance, the following three formulas are used to ensure that the sequence of indices in the tree produced by M_1 is the same as in the tree produced by M_2:

$$\begin{aligned}
\phi_1 &\equiv \forall x \forall y : (root_1(x) \wedge root_2(y)) \rightarrow Link_1(x, y) \\
\phi_2 &\equiv \forall x \forall y : Link_1(x, y) \rightarrow (index(x) \wedge SameLabel(x, y)) \\
\phi_3 &\equiv \forall x \forall y : Link_1(x, y) \rightarrow \\
&\qquad \exists x' \exists y' : NextIndex_1(x, x') \wedge NextIndex_2(y, y') \wedge \\
&\qquad (Link_1(x', y') \vee (\mathrm{lab}_\Diamond(x') \wedge \mathrm{lab}_\Diamond(y')))
\end{aligned}$$

Here, $SameLabel(x, y)$ and $NextIndex_1(x, x')$ are abbreviations of formulas expressing that x and y have the same label and that x' is the first node below x that has a label in $[n]$, respectively. The full interface definition is given in the Appendix. □

4 The Completion Problem

In this section, the completion problem for regular MSO Millstream systems is studied. Let us first define this problem. Given a Millstream system $MS = (M_1, \ldots, M_k; INT)$ over a ranked alphabet Σ and a set $K \subseteq [k]$, the (uniform) *K-completion problem for MS* is defined as follows:

Instance. A family $\kappa = (\kappa_i)_{i \in K}$ of trees $\kappa_i \in \mathrm{T}_\Sigma$.
Question. Is there a *completion* of κ, i.e., a configuration $\langle |(t_1, \ldots, t_k)|, \Psi \rangle$ in $L(MS)$ such that $t_i = \kappa_i$ for all $i \in K$?

Intuitively, the trees κ_i, $i \in K$, are the "known trees" of an otherwise unknown configuration in $L(MS)$, which is sought. Note that there can be zero, one, finitely or infinitely many such configurations. Thus, one may also wish to compute a representation of the set of all completions of κ. If we talk about the problem at a general level, we simply call it the completion problem.

The completion problem is of obvious linguistic relevance, as discussed in the introduction. There are two extreme cases of the completion problem. The first is when $K = [k]$, asking whether k given trees can be linked consistently. The second is when $K = \emptyset$, asking whether $L(MS)$ is nonempty. The first is trivially decidable by enumeration, provided that a logic is used for which it can be decided whether a given configuration satisfies a given formula. The second was studied in the previous section, which yields the following corollary of Theorem 3.

Corollary 4. *The completion problem for regular FO Millstream systems is undecidable.*

Our next goal is to identify conditions under which the completion problem becomes decidable. For this, we now define a normal form of MSO Millstream systems, called typed MSO Millstream system, that turns out to be useful. Intuitively, in a typed MSO Millstream system with k modules, every variable is associated with an index $i \in [k]$, indicating that this variable is meant to range only over (sets of) nodes in $V(t_i)$. In the definition, we let $V_i(X)$ abbreviate $\forall x \colon (x \in X \to V_i(x))$.

Definition 5 (Typed MSO Millstream system). *Let $MS = (M_1, \ldots, M_k;$ $INT)$ be an MSO Millstream system. An interface condition φ of MS is* typed *if each quantified subformula of φ is of one of the forms $\exists \xi \colon (V_i(\xi) \wedge \varphi')$ and $\forall \xi \colon (V_i(\xi) \to \varphi')$, where ξ is an individual or set variable and $i \in [k]$. We abbreviate such formulas by $\exists \xi^{(i)} \colon \varphi'$ and $\forall \xi^{(i)} \colon \varphi'$, respectively, and call i the type of ξ. MS is* typed *if each of its interface conditions is typed.*

The following lemma states that MSO Millstream systems can, without loss of generality, be assumed to be typed. The rather straightforward proof, which can be found in the appendix, is based on a recursive construction that replaces every variable ξ by k variables $\xi_1^{(1)}, \ldots, \xi_k^{(k)}$.

Lemma 6. *Every MSO Millstream system can effectively be turned into a typed MSO Millstream system MS', such that $L(MS') = L(MS)$.*

In the following, we assume that variables of type i occur only in the "right places" in typed formulas. For example, if $\mathcal{I}$ contains $I\colon 2\times 1$ and $I(x,y)$ occurs in a typed formula, then x and y are of types 2 and 1, respectively. This is no restriction, because $I(x,y)$ would necessarily be false, otherwise.

Let $C = \langle (t_1,\dots,t_k), \Psi\rangle$ be a configuration of $MS = (M_1,\dots,M_k; INT)$, where $INT = (\mathcal{I},\Phi)$, and let $K \subseteq [k]$. We now define the configuration C/K which is obtained by removing the trees with indices in K and their nodes from the interface links, but memorizing the latter by attaching subscripts to the interface symbols, thus hard coding the information into the interface symbol itself. In particular, we make use of a new alphabet of interface symbols which depends on the trees t_i, $i \in K$. Before defining C/K formally, we introduce some convenient notation. For a finite number of indexed elements $a_1,\dots,a_n$, and a condition φ that is true or not for each of the a_i, we denote by $(a_i \mid \varphi(a_i))$ the tuple $(a_{i_1},\dots,a_{i_m})$ such that $i_1 < \cdots < i_m$ and $\{i_1,\dots,i_m\} = \{i \in [n] \mid \varphi(a_i)\}$.

Now, we let $C/K = \langle (t_i \mid i \notin K), \Psi'\rangle$, where Ψ' is obtained from Ψ, as follows. For every interface symbol $I\colon i_1\times\cdots\times i_l$ in $\mathcal{I}$, every interface link $I(v_1,\dots,v_l)$ in C is replaced with $I_{u_1\dots u_m}(v'_1,\dots,v'_{l-m})$, where $(u_1,\dots,u_m) = (v_j \mid j \in [l] \text{ and } i_j \in K)$ and $(v'_1,\dots,v'_{l-m}) = (v_j \mid j \in [l] \text{ and } i_j \notin K)$. We define

$$L(MS)/K = \{C/K \mid C = \langle (t_1,\dots,t_k),\Psi\rangle \in L(MS) \text{ and } t_i = \kappa_i \text{ for all } i \in K\}.$$

Lemma 7. *Let $MS = (M_1,\dots,M_k; INT)$ with $INT = (\mathcal{I},\Phi)$ be an MSO Millstream system over Σ, and let $\kappa_i \in \mathrm{T}_\Sigma$ for all $i \in K$, where $K \subseteq [k]$. Then one can effectively construct an MSO Millstream system $MS' = (M'_1,\dots,M'_{k-|K|}; INT')$, with $(M'_1,\dots,M'_{k-|K|}) = (M_i \mid i \in [k]\setminus K)$ and $L(MS') = L(MS)/K$.*

Proof. By induction, and since the statement is trivially true for $K = \emptyset$, it suffices to prove the lemma for $|K| = 1$. Suppose without loss of generality that $K = \{k\}$ and, by Lemma 6, that MS is typed. Furthermore, assume that $i_1 \le \cdots \le i_l$, for all $I\colon i_1\times\cdots\times i_l$ in $\mathcal{I}$, and let $m(I) = |\{j \in [l] \mid i_j = k\}|$. In the following, we denote κ_k by κ.

The new alphabet of interface symbols contains all $I_{v_1\dots v_{m(I)}}\colon i_1\times\cdots\times i_{l-m(I)}$, such that $I\colon i_1\times\cdots\times i_l$ is in $\mathcal{I}$ and $v_1,\dots,v_{m(I)} \in V(\kappa)$. It remains to define the interface conditions of MS'. We do this by recursively turning each individual interface condition $\varphi \in \Phi$ into an appropriate interface condition φ' for MS'. In fact, φ' will in general contain atomic subformulas (not involving interface symbols) in which variables $x^{(k)}$ or $X^{(k)}$ have been replaced with constants, i.e., nodes or sets of nodes of κ. Clearly, these constants can be removed by replacing the corresponding subformulas with either true or false, because κ is fixed.

Consider a (typed) MSO formula ϕ without free variables of type k. We define ϕ' as follows.

- If ϕ is atomic, then $\phi' = \phi$ unless $\phi = I(x_1,\dots,x_{l-m(I)},v_1,\dots,v_{m(I)})$ for some interface symbol $I\colon i_1\times\cdots\times i_l$, variables $x_1,\dots,x_{l-m(I)}$, and nodes $v_1,\dots,v_{m(I)} \in V(\kappa)$. In the latter case, $\phi' = I_{v_1\cdots v_{m(I)}}(x_1,\dots,x_{l-m(I)})$.
- If $\phi = \phi_1 \wedge \phi_2$, then $\phi' = \phi'_1 \wedge \phi'_2$, and similarly for $\phi = \phi_1 \vee \phi_2$ and $\phi = \neg\phi_1$.

– If $\phi = \forall x^{(i)} \colon \phi_1$, there are two cases. Either $i < k$, in which case $\phi' = \forall x^{(i)} \colon \phi_1'$, or $i = k$, in which case $\phi' = \bigwedge_{v \in V(\kappa)} \phi_1\langle x \leftarrow v\rangle'$. Here, $\phi_1\langle x \leftarrow v\rangle'$ is the formula obtained by substituting v for all free occurrences of x in ϕ_1.
– If $\phi = \forall X^{(i)} \colon \phi_1$, then $\phi' = \forall X^{(i)} \colon \phi_1'$ if $i < k$, and $\phi' = \bigwedge_{V \subseteq V(\kappa)} \phi_1\langle X \leftarrow V\rangle'$.
– The cases $\phi = \exists x^{(i)} \colon \phi_1$ and $\phi = \exists X^{(i)} \colon \phi_1$ are similar, the only difference being that $\bigwedge$ is replaced with $\bigvee$.

By structural induction on ϕ, it can be shown that, for every assignment of (sets of) nodes of $t_1, \ldots, t_{k-1}$ to the free variables in ϕ and for every configuration $C = \langle (t_1, \ldots, t_{k-1}, \kappa), \Psi\rangle$, C satisfies ϕ if and only if C/K satisfies ϕ'. □

Given a Millstream system $MS = (M_1, \ldots, M_k; INT)$ with $INT = (\mathcal{I}, \Phi)$, we call a set $U \subseteq [k]$ *unlinked* (with respect to MS) if, for all interface symbols $I \colon i_1 \times \cdots i_l$ in $\mathcal{I}$, we have $|\{j \in [l] \mid i_j \in U\}| \le 1$. In other words, U is unlinked if there are no interface symbols that could establish direct links between the nodes of the trees generated by the modules M_i, $i \in U$.

Theorem 8. *Let $MS = (M_1, \ldots, M_k; INT)$ be a regular MSO Millstream system. For all $K \subseteq [k]$, if $[k] \setminus K$ is unlinked, then a K-completion of κ can be computed for every $\kappa = (\kappa_i)_{i \in K}$, $\kappa_i \in \mathrm{T}_\Sigma$. In particular, the K-completion problem for MS is decidable.*

Proof sketch. Let $l = k - |K|$. By Lemma 7, we can effectively construct a regular MSO Millstream system $MS' = (M_1', \ldots, M_l'; INT')$ such that $L(MS') = L(MS)/K$. Since $[k] \setminus K$ is unlinked, all interface symbols in INT' are of rank ≤ 1. Adding an additional root symbol on top of every configuration $C = \langle (t_1, \ldots, t_l), \Psi\rangle$ of MS', this shows that $L(MS')$ is essentially a regular tree language (using the fact that the regular tree languages are exactly the MSO-definable ones [14,5]). Consequently, we can check whether $L(MS')$ is empty and compute a configuration $C \in L(MS')$ if it is not. From C, one can easily construct a configuration $C_0 = \langle (t_1, \ldots, t_k), \Psi\rangle \in L(MS)$ with $C_0/K = C$ and $t_i = \kappa_i$ for all $i \in K$, by reversing the construction of C_0/K. This completes the proof. □

Under the assumptions of the theorem and abstracting from some irrelevant details, $L(MS)/K$ is a regular tree language. Hence, even finiteness can be decided.

Corollary 9. *Given a regular Millstream system $MS = (M_1, \ldots, M_k; INT)$ and $K \subseteq [k]$ s.t. $[k] \setminus K$ is unlinked, it can be decided whether $L(MS)/K$ is finite.*

5 Configurations of Bounded Tree Width

We have seen that the emptiness problem is undecidable, but that the completion problem is decidable in certain cases. Another way to achieve positive results in situations such as those studied here is to consider structures of bounded tree width. Readers who are unfamiliar with this notion may consult, e.g., [2]. In the following, we use tree width to find restrictions under which the completion problem can be solved more efficiently. For this, we regard a configuration $\langle |T|, \Psi \rangle$ as an undirected graph on $V(T)$. More precisely, we say that Ψ contains a link $(v_1, \ldots, v_l)$ and write $(v_1, \ldots, v_l) \in \Psi$, if $I(v_1, \ldots, v_l) = \text{true}$ for some interface symbol I. Two distinct nodes u, v are considered to be connected by an edge if one is a child of the other or $u, v \in \{v_1, \ldots, v_l\}$ for some link $(v_1, \ldots, v_l) \in \Psi$. We say that a set L of configurations has *bounded tree width* if there is a constant $w \in \mathbb{N}$ such that every configuration in L is of tree width at most w.

By the results of [4,3], we have the following.

Theorem 10. *Let $MS = (M_1, \ldots, M_k; INT)$ be a regular MSO Millstream system. If $L(MS)$ is of bounded tree width, then*

- *the membership problem for $L(MS)$ can be solved in linear time,*
- *the K-completion problem for MS is decidable for every $K \subseteq [k]$, and*
- *for every MSO formula φ, it can be decided whether all configurations in $L(MS)$ satisfy φ.*

Thus, Millstream systems MS for which $L(MS)$ is of bounded tree width are of particular interest. In the rest of this section, we establish two conditions that guarantee that $L(MS)$ is of bounded tree width.

Let $MS = (M_1, \ldots, M_k; INT)$ be an MSO Millstream system and $C = \langle |(t_1, \ldots, t_k)|, \Psi \rangle$ a configuration of MS. We say that a node $v \in t_i$ is *linked* if it occurs in a link in Ψ.

Definition 11 (Simple configuration). *Let $C = \langle |(t_1, \ldots, t_k)|, \Psi \rangle$ be a configuration. For nodes $x, x' \in V(t_i)$, let $gcp(x, x')$ be the* greatest common predecessor *of x and x', i.e. the largest (w.r.t. $<$) node v in t_i such that $v < x$ and $v < x'$. C is* simple *if it satisfies the following conditions:*

1. *No node in any of the trees $t_1, \ldots, t_k$ is involved in more than one link, i.e., each node $v \in t_i$ occurs in at most one link in Ψ.*
2. *For every $x_1, x_2, x_3 \in t_i$ and every $y_1, y_2, y_3 \in t_j$ such that x_1 is linked to y_1, x_2 is linked to y_2, and x_3 is linked to y_3, $gcp(x_1, x_2) \geq gcp(x_1, x_3)$ if and only if $gcp(y_1, y_2) \geq gcp(y_1, y_3)$.*

The intuition behind the second condition is that the links have to respect the branching structure of the tree. Figure 2 illustrates how a configuration can violate the criterion for simplicity.

Next, we prove that every simple configuration of a Millstream system with two modules has bounded tree-width.

Theorem 12. *If $C = \langle |(t_1, t_2)|, \Psi \rangle$ is a simple configuration of $MS = (M_1, M_2; INT)$, then C has tree-width at most* 2.

Proof sketch. For the proof, we use a graph search game by Seymour and Thomas [13]. The game is played between a robber and k cops on an undirected graph. The robber stands on a vertex v of the graph and can at any time run to any

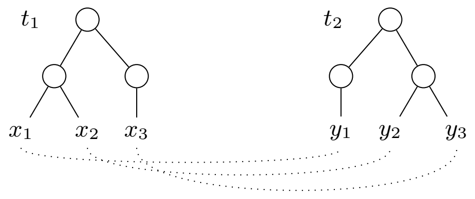


Fig. 2. The above configuration is *not* simple, since the branches of x_1 and x_2 come together *before* the branches of x_2 and x_3, while the branches of y_1 and y_2 come together *after* the branches of y_2 and y_3

vertex u such that there is a cop-free path from v to u. Each cop is at any time either placed on a vertex or is in a helicopter. The objective of the cops is to land a cop on a vertex occupied by the robber. The robber can, however, see the helicopter approaching and has time to flee to another vertex. Thus the cops have to corner the robber, i.e., create a situation where the robber is on vertex v, all neighbors of v are occupied by cops, and there is a cop in the helicopter available to land on v and capture the robber.

A graph has tree-width k if and only if $k + 1$ is the minimal number of cops that have a winning strategy against the robber on the graph [13]. Thus our aim is to show that 3 cops can catch a robber on any simple configuration $C = \langle |(t_1, t_2)|, \Psi \rangle$. The strategy will be to place one cop in each tree t_i, make sure that the robber can never again get to any node in t_i outside the subtree of the node on which the cop is placed, and in each round move the cop one step down in one of the subtrees. Thus the robber is caught in at most $2 \cdot h$ steps, where h is the maximal height of any tree in C.

The cops will use the following strategy:

1. In the first step, two cops are placed on the two roots of t_1, t_2.
2. After m steps, one cop will be placed in each tree, say on the nodes c_1 and c_2. The robber will be in one of the subtrees $t_1/c_1, t_2/c_2$ and there will be no links from nodes within these subtrees to nodes outside them. Assume that the robber is in subtree t_1/c_1. We distinguish two cases:
   (a) If there is a child node w of c_2 such that no node of t_2 outside of t_2/w is accessible to the robber, and there is no link from any node in t_1/c_1 to c_2, then the free cop is placed on w and the cop on c_2 is removed.
   (b) Otherwise the free cop is placed on the child v of c_1 such that the robber is in t_1/v and the cop on c_1 is removed.

The proof that this strategy works is given in the appendix. □

Let us now define a second notion that can be used to bound the tree width of $L(MS)$. In this definition and in the remainder of the section, if $u, u_1, \ldots, u_n$ are nodes in a configuration of a Millstream system, we write $u > \{u_1, \ldots, u_n\}$ to express that $u > u_i$ for some $i \in [n]$. We say that a node v is a *successor* of u if v is a minimal linked node such that $u > v$.

Definition 13 (Nested Millstream system). *A Millstream system $MS = (M_1, \ldots, M_k; INT)$ is* nested *if there is a constant $h \in \mathbb{N}$ such that the following hold for every configuration $C = \langle |(t_1, \ldots, t_k)|, \Psi \rangle \in L(MS)$.*

(1) There are at most h links in Ψ containing a minimal linked node (i.e., a node that is not a successor of another node).

Furthermore, for every link $(u_1, \ldots, u_l) \in \Psi$,

(2) there are at most h links $(v_1, \ldots, v_m) \in \Psi$ such that v_i is a successor of u_j for some $i \in [m]$ and $j \in [l]$, and
(3) for each of the links $(v_1, \ldots, v_m)$ in (2) and every $i \in [m]$, $v_i > \{u_1, \ldots, u_l\}$.

For a given configuration $C = \langle |T|, \Psi \rangle$ and distinct nodes $u, u' \in V(T)$, we say that u *is linked with* u' if there is a link $(v_1, \ldots, v_l) \in \Psi$ such that $u, u' \in \{v_1, \ldots, v_l\}$. A (u, u')*-path in* C is a sequence $u_0 \cdots u_n \in V(T)^*$ such that $u_0 = u$, $u_n = u'$, and for every $i \in [n]$, one of u_{i-1}, u_i is a child of the other or u_{i-1} is linked with u_i. Such a path is said to *use* each of the nodes $u_0, \ldots, u_n$. The proof of the next lemma can be found in the appendix.

Lemma 14. *Let $C = \langle |T|, \Psi \rangle \in L(MS)$, where MS is a nested Millstream system, and let $(v_1, \ldots, v_l) \in \Psi$ and $u, u' \in V(T)$. If $u > \{v_1, \ldots, v_l\}$ and there is a (u, u')-path in C that does not use any of $v_1, \ldots, v_l$, then $u' > \{v_1, \ldots, v_l\}$.*

Theorem 15. *$L(MS)$ is of bounded tree width for every nested Millstream system MS.*

Proof. Let h be the constant in Definition 13, and let $r \geq 1$ be the maximum rank of interface symbols of $MS = (M_1, \ldots, M_k; INT)$. Without loss of generality, we may assume that all configurations in $L(MS)$ contain a link $(v_1, \ldots, v_k)$ such that $v_1, \ldots, v_k$ are the roots of the trees in the configuration. This assumption removes the need for condition (1) in Definition 13, as it becomes an instance of (2).

Using the cops-and-robbers game, we show that the configurations in $L(MS)$ are of tree width at most $\hat{r} = (h + 1)r + 2$. The winning strategy for the cops works as follows.

Maintained invariant: During the game, the cops will always completely occupy at least one link $(u_1, \ldots, u_l)$, in the sense that cops are placed on each of $u_1, \ldots, u_l$. This link is called the *guarding link*. Moreover, the strategy will guarantee that the robber sits on a node $v > \{u_1, \ldots, u_l\}$. Consequently, the robber cannot move to any node $v' \not> \{u_1, \ldots, u_l\}$ (by Lemma 14).

Initially, cops are placed on the roots of the trees, making sure that the invariant holds. Now, the following is repeated, where $(u_1, \ldots, u_l)$ is the guarding link and v denotes the current position of the robber at any instant in time:

We use the still available cops to occupy all links $(u'_1, \ldots, u'_m)$ such that u'_i is a successor of u_j for some $i \in [m]$ and $j \in [l]$. By Definition 13(2), this requires at most hr cops in addition to those occupying the guarding link. (Note that, as the cops still occupy the guarding link, the invariant still holds.)

Now, let U be the set of all nodes $u > \{u_1, \ldots, u_l\}$ such that there is no successor u'_i of u_i with $u > u'_i$. (In other words, U is the set of descendants of $u_1, \ldots, u_l$ that can be reached on a path not using one of the newly occupied nodes.) There are two cases.

If $v \in U$, then the robber can only move within the tree that is given by the connected component of U that v belongs to. (All nodes of links he could reach for travelling along them are occupied by cops.) Thus, the two remaining cops can be used to corner the robber, while keeping the at most $h+1$ links occupied.

If $v \notin U$, then we have $v > \{u'_1, \ldots, u'_m\}$, for (at least) one of the newly occupied links $(u'_1, \ldots, u'_m)$. We choose this link as the new guarding link, making all other cops available again, and continue. Note that, by Definition 13(3), the sum of the sizes of the subtrees rooted at the nodes of the guarding link has become strictly smaller. Thus, the robber will eventually be caught. □

References

1. Bensch, S., Drewes, F.: Millstream systems. Report UMINF 09.21, Umeå University (2009), http://www8.cs.umu.se/research/uminf/index.cgi?year=2009&number=21
2. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernetica 11(1-2), 1–22 (1993)
3. Courcelle, B.: Graph rewriting: An algebraic and logic approach. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 193–242. Elsevier/MIT Press (1990)
4. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Inf. Comput. 85(1), 12–75 (1990)
5. Doner, J.: Tree acceptors and some of their applications. J. Commput. Syst. Sci. 4(5), 406–451 (1970)
6. Gécseg, F., Steinby, M.: Tree languages. In: Rozenberg, G.S. (ed.) Handbook of Formal Languages, Beyond Words, vol. 3, pp. 1–68. Springer, Berlin (1997)
7. Jackendoff, R.: Foundations of Language: Brain, Meaning, Grammar, Evolution. Oxford University Press, Oxford (2002)
8. Kornai, A., Tuza, Z.: Narrowness, path-width, and their application in natural language processing. Discrete Applied Mathematics 36, 87–92 (1992)
9. Libkin, L.: Elements of Finite Model Theory. Springer, Heidelberg (2004)
10. Post, E.: A variant of a recursively unsolvable problem. Bulletin of the AMS 52, 264–268 (1946)
11. Sadock, J.: Autolexical Syntax - A Theory of Parallel Grammatical Representations. The University of Chicago Press, Chicago & London (1991)
12. Sauerland, U., von Stechow, A.: Syntax-semantics interface. In: Smelser, N., Baltes, P. (eds.) International Encyclopedia of the Social and Behavioural Sciences, pp. 15412–15418. Oxford Pergamon, Oxford (2001)
13. Seymour, P.D., Thomas, R.: Graph searching and a min-max theorem for tree-width. Journal of Combinatorial Theory, Series B 58, 22–33 (1993)
14. Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. Mathematical Systems Theory 2(1), 57–81 (1968)

On a Conjecture by Carpi and D'Alessandro

Mikhail V. Berlinkov

Department of Algebra and Discrete Mathematics, Ural State University,
620083 Ekaterinburg, Russia
berlm@mail.ru

Abstract. Recently, Carpi and D'Alessandro [3] have formulated a conjecture whose validity would imply an $O(n^2)$ upper bound for the minimum length of reset words for synchronizing automata with n states. We refute this conjecture as well as a related conjecture by Rystsov [13] and suggest a weaker version that still suffices to achieve a quadratic upper bound.

1 Strong Transitivity and the Černý Conjecture

Suppose $\mathscr{A}$ is a complete deterministic finite automaton whose input alphabet is Σ and whose state set is Q. The automaton $\mathscr{A}$ is called *synchronizing* if there exists a word $w \in \Sigma^*$ whose action *resets* $\mathscr{A}$, that is, w leaves the automaton in one particular state no matter at which state in Q it is applied: $q.w = q'.w$ for all $q, q' \in Q$. Any such word w is called *reset* or *synchronizing* for the automaton.

Synchronizing automata serve as transparent and natural models of error-resistant systems in many applications (coding theory, robotics, testing of reactive systems) and also reveal interesting connections with symbolic dynamics and other parts of mathematics. For a brief introduction to the theory of synchronizing automata we refer the reader to the recent survey [20]. Here we discuss one of the main problems in this theory: proving an upper bound of magnitude $O(n^2)$ for the minimum length of reset words for n-state synchronizing automata.

In 1964 Černý [5] constructed for each $n > 1$ a synchronizing automaton with n states whose shortest reset word has length $(n-1)^2$. Soon after that he conjectured that those automata represent the worst possible case, thus formulating the following hypothesis:

Conjecture 1 (Černý). *Each synchronizing automaton with n states has a reset word of length at most $(n-1)^2$.*

By now this simply looking conjecture is arguably the most longstanding open problem in the combinatorial theory of finite automata. The best upper bound known so far is due to Pin [12] (it is based upon a combinatorial theorem conjectured by Pin and then proved by Frankl [8]): for each synchronizing automaton with n states, there exists a reset word of length $\frac{n^3-n}{6}$. Since this bound is cubic and the Černý conjecture claims a quadratic value, it is of certain importance to prove quadratic (upper) bounds for some classes of synchronizing automata.

In the rest of the paper, we assume that $\mathscr{A}$ is a synchronizing n-state automaton whose input alphabet is Σ and whose state set is Q. We also assume $\mathscr{A}$ is strongly connected because finding reset words of length $O(n^2)$ can be easily reduced to this case (see [11] for example). For $S \subseteq Q$ and $v \in \Sigma^*$, we denote by $S.v$ and $S.v^{-1}$ the image and respectively the preimage of the subset S under the action of the word v, i.e.

$$S.v = \{q.v \mid q \in S\} \text{ and } S.v^{-1} = \{q \mid q.v \in S\}.$$

A majority of existing methods for proving upper bounds for the minimal length of reset words can be classified into two types: *top-down* (or *merge*) and *bottom-up* (or *extension*) methods. Each of these methods constructs a finite sequence of words $V = (v_1, v_2, \dots, v_m)$ such that the concatenation $v_1 v_2 \cdots v_m$ of the words in the sequence is a reset word for $\mathscr{A}$. We call $m = |V|$ the *size* of the sequence V and $L_V = \max_i |v_i|$ the *length* of V. In methods of the merge type, the words in the sequence V subsequently merge the set Q to some state p, i.e.

$$|Q| > |Q.v_1| > |Q.v_1 v_2| > \cdots > |Q.v_1 v_2 \cdots v_m| = |\{p\}|.$$

In methods of the extension type, the words in V subsequently extend some state p to the set Q, i.e.

$$|\{p\}| < |p.v_m^{-1}| < |p.v_m^{-1} v_{m-1}^{-1}| < \cdots < |p.v_m^{-1} v_{m-1}^{-1} \cdots v_1^{-1}| = |Q|.$$

In both cases, the size of V cannot exceed $n-1$ whence a quadratic upper bound for the minimum length of reset words can be achieved as soon as one gets a linear upper bound for the length of V.

A merge method was used in [12] to prove the aforementioned upper bound $\frac{n^3-n}{6}$ for the general case. Some merge methods were used to prove the Černý conjecture for a few special classes of automata, see, for instance [7,14,15,19,21]. However, there are examples showing that in general the length L_V of a merging sequence V cannot be bound by a linear function of n so it is hard to believe that a quadratic upper bound for the minimum length of reset words can be obtained via merging methods.

Extension methods have become popular over the last 15 years and brought a number of impressive achievements. Using an approach of this sort, Rystsov proved quadratic upper bounds for automata in which each letter either permutes the states or fixes all but one states [16] and for so-called regular automata [13] whose definition we shall recall below. Dubuc [6] proved the Černý conjecture for *circular* automata, i.e. automata with a letter that acts as a cyclic permutation of the state set. He used an extension method combined with skilful linear algebra techniques. Also via an extension method, Kari [10] proved the Černý conjecture for *Eulerian* automata, i.e. automata whose underlying digraph is Eulerian. Recently, quadratic upper bounds have been obtained for so-called *one-cluster* automata (see [1,2]) that can be defined as automata for which there exists a letter labelling precisely one cycle.

In DLT 2008 conference Carpi and D'Alessandro [3] (see also the journal version [4]) introduced a new idea for constructing extension sequences of linear length. The idea is based on the notion of an *independent* set of words. A set of words $W = \{w_1, w_2, \ldots, w_n\} \subseteq \Sigma^*$ is said to be *independent* with respect to a fixed n-state automaton $\mathscr{A}$ if for every two given states s and t, there exists an index i such that $s.w_i = t$. The automaton $\mathscr{A}$ is called *strongly transitive* if it admits some independent set of words. It is easy to check that each synchronizing strongly connected automaton is strongly transitive [4, Corollary 2]. Moreover, if u is a reset word for $\mathscr{A}$, then $\mathscr{A}$ has an independent set of length at most $|u| + n - 1$, and this bound is tight [4, Proposition 3]. Carpi and D'Alessandro also proved [4, Theorem 2] that if an n-state automaton $\mathscr{A}$ is strongly transitive with some independent set W, then it has a reset word of length at most $(n-2)(n + L_W - 1) + 1$ and conjectured that each synchronizing automaton has an independent set of linear length. Formally, let us say that $\mathscr{A}$ satisfies the *k-Independent-Set* property for some number k if it has an independent set of words of length less than kn. Then the conjecture of Carpi and D'Alessandro can be formulated as follows.

Conjecture 2 (Carpi and D'Alessandro). *There exists some number k such that each strongly connected synchronizing automaton satisfies the k-Independent-Set property.*

Since k is a constant, in view of the quoted result from [4], the validity of the conjecture implies a quadratic upper bound for the minimum length of reset words for all synchronizing automata.

Quite similar ideas were developed earlier by Rystsov [13]. Rystsov called a finite set $W \subseteq \Sigma^*$ *regular* for an n-state automaton whose input alphabet is Σ if W contains the empty word, the length of each word in W is less than n and there is a positive integer r such that for each pair of states s, t there exist exactly r words in W which map s to t. An automaton is said to be *regular* if it admits a regular set of words. Rystsov proved [13, Theorem 7] the upper bound $2(n-1)^2$ for the minimum length of reset words for n-state regular automata and proposed the following hypothesis:

Conjecture 3 (Rystsov). *Each strongly connected automaton is regular.*

Again, the validity of this conjecture implies a quadratic upper bound for the minimum length of reset words for all synchronizing automata. However, in this paper we refute both Conjecture 2 and Conjecture 3. The interested readers are also invited to read the Steinberg's paper [17] where a clever generalization of results [4,13] is presented.

In Section 2 of this paper we present a general extension algorithm, introduce two related properties: the k-Extension property and the k-Balance property, and prove that the latter one follows from the k-Independent-Set property as well as from the property of being regular in Rystsov's sense. In Section 3 we construct a series of automata for which the k-Extension property fails for each $k < 2$ and the k-Balance property fails for each k. Thus, this series refutes both the conjecture by Carpi and D'Alessandro and the conjecture by Rystsov. Finally, in Section 4 we relax the above properties to a "local" form and discuss perspectives of extension methods.

2 Extension Algorithm and Related Properties

Here we describe the essence of extension methods implicity used in [1,2,3,6,10,13,16] and some other papers. Recall that $\mathscr{A}$ is an n-state strongly connected synchronizing automaton. Suppose C_s, C_e are some nonempty subsets of Q such that $C_s \subseteq C_e$, and v_s, v_e are some words such that $Q.v_e = C_e$ and $|C_s.v_s| = 1$. Then the following algorithm returns a reset word by constructing an extension sequence of words.

Extension Algorithm (EA)

```
input $\mathscr{A}, C_s, C_e, v_s, v_e$
initialization $v \leftarrow v_s$
               $S \leftarrow C_s$
while $|S \cap C_e| < |C_e|$, find a word $u \in \Sigma^*$ of minimum length with
    $|S.u^{-1} \cap C_e| > |S \cap C_e|$; if none exists, return Failure
    $v \leftarrow uv$
    $S \leftarrow S.u^{-1}$
return $v_e v$
```

It is easy to show that the algorithm EA is correct. First, the main loop iterates at most $|C_e| - |C_s|$ times because each iteration expands the set $S \cap C_e$ by one or more elements. Let us show that EA does not fail, i.e. some word u exists for each iteration. Consider a generic iteration of the main loop. We have $|S \cap C_e| < |C_e|$ by the loop condition. Since $\mathscr{A}$ is synchronizing, there exists a reset word w, i.e. $Q.w = \{p\}$ for some state $p \in Q$. Moreover, since $\mathscr{A}$ is strongly connected, the word w can be chosen to satisfy $p \in S$. Then $Q = p.w^{-1} \subseteq S.w^{-1} \subseteq Q$ whence $S.w^{-1} = Q$ and

$$|S.w^{-1} \cap C_e| = |Q \cap C_e| = |C_e| > |S \cap C_e|.$$

Thus, there is a word w satisfying $|S.w^{-1} \cap C_e| > |S \cap C_e|$, and therefore, there exists also a word u of minimum length with the same property. Hence, after the last iteration we have $C_e \subseteq S$ and $S = p.v^{-1}$ for some state $p \in Q$. Since $C_e.{v_e}^{-1} = Q$, we have

$$p.(v_e v)^{-1} = p.v^{-1}{v_e}^{-1} = S.{v_e}^{-1} = Q.$$

This implies that $v_e v$ is a reset word.

Since the loop of EA iterates at most $|C_e| - |C_s|$ times, the length of the word $v_e v$ does not exceed

$$|v_e| + |v_s| + (|C_e| - |C_s|) \max_{S \subset C_e} |u(S)|.$$

Thus, in order to prove an upper bound, we have to estimate the maximal possible length of the extension words (the length of the extension sequence).

A subset $S \subseteq Q$ is called *m-extendable* in the subset $C_e \subseteq Q$, if there exists some (extension) word v of length at most m such that

$$|S.v^{-1} \cap C_e| > |S \cap C_e|.$$

We say that $\mathscr{A}$ possesses the *Extension property* if every proper subset S is n-extendable (in $C_e = Q$). It is known (and easy to see) that automata with the Extension property satisfy the Černý conjecture. Indeed, suppose that $\mathscr{A}$ satisfies the Extension property. Let us set $C_e = Q$ and $v_e = 1$. Since $\mathscr{A}$ is synchronizing, there exists a letter v_s and a subset C_s such that $|C_s| > |C_s.v_s| = 1$. Applying EA for these input data, we get a reset word $v_e v$ as a result. Since

$$|v_e v| \leq |v_e| + |v_s| + (|C_e| - |C_s|) \max_{S \subset C_e} |u(S)| \leq 0 + 1 + (n-2)n = (n-1)^2,$$

we see that $\mathscr{A}$ satisfies the Černý conjecture. This argument was used in [6] and [10] to prove the Černý conjecture for circular and Eulerian automata respectively.

There are automata that do not satisfy the Extension property. For instance, it can be verified that the 6-state automaton discovered by Kari [9] contains a proper subset which is not 6-extendable and even not 7-extendable. In the next section we present an infinite series of simple examples of automata that do not satisfy the Extension property[1].

We now consider a generalization of the Extension property. Given a fixed number k, we say that an automaton $\mathscr{A}$ possesses the *k-Extension* property if every proper subset S is kn-extendable in Q. Clearly, for such an automaton, the algorithm EA returns a reset word of length at most $(n-2)kn + 1$, thus providing a quadratic upper bound for the minimum length of reset words.

One further related property is formulated in terms of a natural linear structure associated with automata. We fix a numbering $q_1, \dots, q_n$ of the states of $\mathscr{A}$ and then assign to each subset $T \subseteq Q$ its *characteristic vector* $[T]$ in the linear space $\mathbb{R}^n$ defined as follows: the i-th entry of $[T]$ is 1 if $q_i \in T$, otherwise it is equal to 0. As usually, for any two vectors $g_1, g_2 \in \mathbb{R}^n$ we denote the inner product of these vectors by (g_1, g_2). Now we say that $\mathscr{A}$ satisfies the *k-Balance* property if each proper subset S of Q admits a collection of words $v_1, v_2 \dots v_m$ (some of them may be equal) such that $|v_i| < kn$ and the vector equality

$$\sum_{i=1}^{m} [S.v_i^{-1}] = m \frac{|S|}{|Q|} [Q].$$

holds in $\mathbb{R}^n$. One can show that a synchronizing automaton satisfying the $(k-1)$-Balance property also satisfies k-Extension property, see [1,2,3,13]. Thus, the k-Balance property also implies a quadratic upper bound for the minimum length of reset words.

[1] One of the anonymous referees of this paper has informed us that an infinite series of automata that does not satisfy the Extension property has been independently constructed by John Dixon (unpublished).

Lemma 1. *If a synchronizing n-state automaton $\mathscr{A}$ satisfies the k-Independent-Set property, then $\mathscr{A}$ also satisfies the k-Balance property. Furthermore, if $\mathscr{A}$ is regular in Rystsov's sense, then $\mathscr{A}$ satisfies the* 1*-Balance property.*

Proof. Suppose r is a positive integer and $W = \{w_1, w_2, \dots w_m\}$ is a set of words of length less than kn such that for each two given states s and t there are exactly r different words $w_1(s,t), w_2(s,t), \dots, w_r(s,t) \in W$ such that $s.w_i(s,t) = t$ for each $i \in \{1 \dots r\}$. Observe that $\mathscr{A}$ satisfies the k-Independent-Set property when $m = n$ and $r = 1$ and $\mathscr{A}$ is regular in Rystsov's sense when $k = 1$. Let us fix an arbitrary state t. Then for each $s \in Q$ and each $i \in \{1 \dots r\}$ we have $s.w_i(s,t) = t$. Since the automaton is deterministic, $w_i(s,t_1) \neq w_j(s,t_2)$ for all numbers $i, j \in \{1 \dots r\}$ and all states s, t_1, t_2 such that $t_1 \neq t_2$. Hence, $m = rn$ and, for each subset $S \subseteq Q$, we have

$$\sum_{i=1}^{m} [S.w_i^{-1}] = \sum_{i=1}^{m} \sum_{t \in S} [t.w_i{}^{-1}] = \sum_{t \in S} \sum_{i=1}^{m} [t.w_i{}^{-1}]$$
$$= \sum_{t \in S} r[Q] = r|S|[Q] = m\frac{|S|}{|Q|}[Q].$$

Thus, $\mathscr{A}$ satisfies the k-Balance property.

3 Examples

The automaton $\mathscr{A}(m,k)$ for $m \geq 2, k \geq 1$ with the input alphabet $\Sigma = \{a, b\}$ is shown in Figure 1. The state set Q of the automaton consists of elements $q_0, q_1, \dots, q_m, s_1, s_2, \dots, s_k$. The letter a generates the cycle $C_a = (q_0, q_1, \dots, q_m)$ and maps the other states to q_2, i.e.

$$q.a = \begin{cases} q_{i+1}, & \text{if } q = q_i, 0 \leq i \leq m-1; \\ q_0, & \text{if } q = q_m; \\ q_2, & \text{if } q = s_j, 1 \leq j \leq k. \end{cases} \quad (1)$$

The letter b generates the cycle $C_b = (q_0, s_1, s_2, \dots, s_k)$ and fixes the other states, i.e.

$$q.b = \begin{cases} s_{j+1}, & \text{if } q = s_j, 1 \leq j \leq k-1; \\ q_0, & \text{if } q = s_k; \\ s_1, & \text{if } q = q_0; \\ q_i, & \text{if } q = q_i, 1 \leq i \leq m. \end{cases} \quad (2)$$

The following lemma can be easily checked.

Lemma 2. *$\mathscr{A}(m,k)$ is an $(m+k+1)$-state strongly connected synchronizing automaton and the word $a(ba^m)^{m-1}ba$ is a reset word for the automaton.*

The next lemma directly follows from the construction of the automaton and shows when the letter b can appear in the shortest extension word.

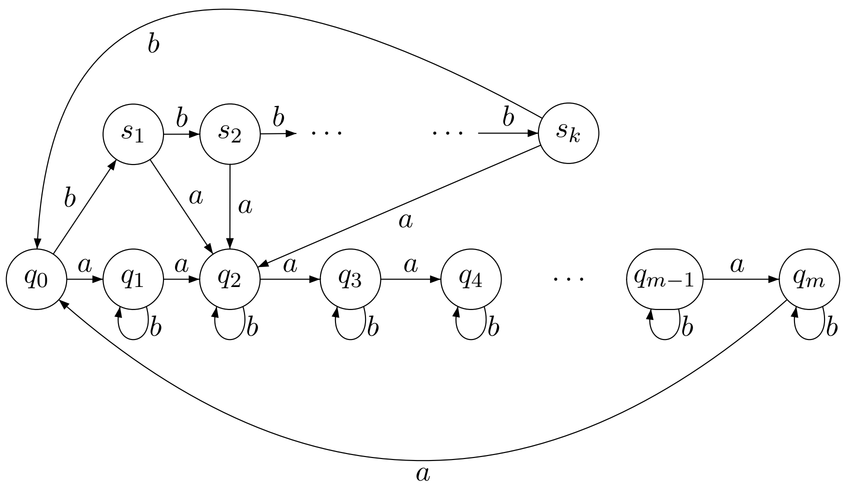


Fig. 1. Automaton $\mathscr{A}(m,k)$

Lemma 3. *Suppose $S \subset Q$ is such that $S.b^{-1} \neq S$. Then $C_b \cap S \neq \varnothing$ and $C_b \nsubseteq S$.*

For every $n > 3$, let $\mathscr{B}_n = \mathscr{A}(n-2,1)$.

Theorem 1. *For each $c < 2$, the automaton $\mathscr{B}_n$ does not satisfy the c-Extension property provided that $n > \frac{3}{2-c}$. In particular, the automaton $\mathscr{B}_n$ does not satisfy the Extension property for each $n > 3$.*

Proof. By Lemma 2 the automaton $\mathscr{B}_n$ is an n-state strongly connected synchronizing automaton. We set $S = C_b$ and let v be a word of minimal length such that $|S.v^{-1}| > |S|$. Recall that $k = 1$ and $m = n-2$. Our aim is to prove that $v = a^m b a^m$. We denote by $v(i)$ the letter in the i-th position of the word v. Since $S = C_b$, by Lemma 3 we have $v(1) = a$ and $S_1 = S.v(1)^{-1} = S.a^{-1} = \{q_m\}$. Further, since $C_b \cap S_1 = \varnothing$, we have $v(2) = a$. Applying these arguments m times, we obtain

$$S_m = S.(v(1)\cdots v(m))^{-1} = S.(a^m)^{-1} = \{q_1, s_1, s_2, \ldots, s_k\}.$$

Since $S_m.a^{-1} = \{q_0\} \subseteq S$, we have

$$v(m+1) = b \text{ and } S_m.b^{-1} = \{q_0, q_1, s_1, s_2, \ldots, s_{k-1}\}.$$

It is clear that $v(m+2) = a$, since v has been chosen to be the shortest extension word. If we repeat these arguments, we get $v = a^m b a^m$. Thus, $\mathscr{B}_n = \mathscr{A}(n-2,1)$ is an n-state synchronizing automaton, the shortest extension word for the subset S is v and its length is $2m+1 = 2n-3$. Therefore the set S is not cn-extendable in $\mathscr{B}_n$ for every c such that $n > \frac{3}{2-c}$.

Now, for any given integer $k > 1$, we denote by $\mathscr{C}_n$ the automaton $\mathscr{A}(n-k-1,k)$.

Theorem 2. *For each $k \in \mathbb{N}$, the automaton $\mathscr{C}_n$ with $n > k^2+k$ does not satisfy the $(k-1)$-Balance property, and thus, refutes both the conjecture by Carpi and D'Alessandro and the conjecture by Rystsov.*

Proof. By Lemma 2 we have that the automaton $\mathscr{C}_n$ is an n-state strongly connected synchronizing automaton. Arguing by contradiction, suppose that the $(k-1)$-Balance property holds true for $\mathscr{C}_n$ and let $S = C_b$. Then there are words $v_1, v_2 \ldots v_r$ such that $|v_i| < (k-1)n$ and the following vector equality holds true.

$$\sum_{i=1}^{r} [S.v_i^{-1}] = r\frac{|S|}{|Q|}[Q].$$

Arguing by contradiction, suppose $([S.v_j^{-1}], [C_a]) \le k$ for each $j \in \{1 \ldots r\}$. Then if we multiply the equation through by $[C_a]$, we have

$$rk \ge \sum_{i=1}^{r} [S.v_i^{-1}] = (r\frac{|S|}{|Q|}[Q], [C_a]) = r\frac{(k+1)(n-k)}{n}.$$

It is easy to check that the last inequality contradicts the condition $n > k^2+k$. Hence there exists a number j such that $([S.v_j^{-1}], [C_a]) \ge k+1$. Since $[S.v_j^{-1}]$ and $[C_a]$ are zero-one vectors, we have $|S.v_j^{-1} \cap C_a| \ge k+1$.

The argument in the proof of Theorem 1 shows that $a^m b$ is the shortest word v with the property that two states q_0, q_1 from C_a lie in $S.v^{-1}$. Basically the same argument shows that $(a^m b)^k$ is the shortest word w with the property that at least $k+1$ states from the set C_a lie in $S.w^{-1}$. Hence we have

$$|v_j| \ge |(a^m b)^k| = k(m+1) = k(n-k).$$

Since $n > k^2+k$, we have $|v_j| = k(n-k) > (k-1)n$ and we get a contradiction. Hence the $(k-1)$-Balance property fails for $\mathscr{C}_n$. By Lemma 1 so does the $(k-1)$-Independent-Set property, whence the conjecture of Carpi and D'Alessandro cannot hold for $\mathscr{C}_n$. Also by Lemma 1 $\mathscr{C}_n$ cannot be regular in Rystsov's sense.

4 Conclusion

We have refuted the conjecture by Carpi and D'Alessandro and the conjecture by Rystsov and have shown that some arbitrarily large synchronizing automata do not satisfy the k-Extension property for each $k < 2$. However, this does not mean that extension methods can not be applied to prove the Černý conjecture or quadratic upper bounds for the minimum length of reset words in the general case. In terms of the notation of the algorithm EA, our examples only show that extension methods with $C_e = Q$ can be hardly successful.

This suggests to relax the k-Extension property to the following *k-Local-Extension* property: there are subsets $C_s, C_e \subseteq Q$ and words $v_s, v_e \in \Sigma^*$ such that

$$|C_s.v_s| = 1, \quad C_e.v_e^{-1} = Q, \quad C_s \subseteq C_e,$$
$$|v_s| \le k + kn(|C_s| - 2), \quad |v_e| \le kn(n - |C_e|)$$

and each proper subset S of C_e is kn-extendable in C_e, i.e.

$$|S.v^{-1} \cap C_e| > |S \cap C_e|$$

for some word v with $|v| \le kn$.

If an n-state synchronizing automaton $\mathscr{A}$ satisfies the k-Local-Extension property, then it has a reset word of length at most

$$kn(n - |C_e|) + k + kn(|C_s| - 2) + (|C_e| - |C_s|)kn = k(n-1)^2.$$

In particular, if $k = 1$, then the Černý conjecture holds true for $\mathscr{A}$. Similarly, the k-Balance property can be relaxed to the following *k-Local-Balance* property: each proper subset S of C_e admits a collection of words $v_1, v_2 \dots v_r$ such that $|v_i| < kn$ and

$$\sum_{i=1}^{r} [S.v_i^{-1}] = r\frac{|S|}{|Q|}[Q].$$

The k-Local-Balance property implies the "main part" of the $(k+1)$-Local-Extension property, i.e. each proper subset S of C_e can be extended in C_e by using a word v of length at most $(k+1)n$. One can easily prove this fact using techniques from [1,2]. In fact, to prove a quadratic upper bound for the minimum length of reset words in one-cluster automata, the authors of [1,2] have actually proved 2-Local-Extension property, using the 1-Local-Balance property as an auxiliary statement.

It is easy to see that the 1-Local-Extension property holds true for the automaton $\mathscr{A}(n-k-1,k)$ with

$$C_s = \{q_0, q_1\}, \qquad v_s = ba$$
$$C_e = \{q_0, q_1, \dots, q_m\}, \qquad v_e = a.$$

Moreover, by running the algorithm EA on this input we get as a result the reset word $a(ba^m)^{m-1}ba = v$ of length $m^2 + 2 = (n-k-1)^2 + 2$. Using Lemma 3, one can easily prove that this word is a reset word of minimum length for the automaton. It is also easy to see that $\mathscr{A}(n-k-1,k)$ with the same C_s, v_s, C_e and v_e satisfies the k-Local-Balance property. Thus, our examples, which have been designed to demonstrate that "global" versions of some synchronization properties fail, do satisfy slightly relaxed versions of the same properties. We hope that these relaxed version may be useful for further work in the direction towards proving a quadratic upper bound for the minimum length of reset words for arbitrary synchronizing automata.

The author acknowledges support from the Federal Education Agency of Russia, grant 2.1.1/3537, and from the Russian Foundation for Basic Research, grants 09-01-12142 and 10-01-00793. The author is extremely grateful to the anonymous referees for their detailed remarks and useful suggestions.

References

1. Béal, M., Berlinkov, M., Perrin, D.: A quadratic upper bound on the size of a synchronizing word in one-cluster automata (submitted)
2. Béal, M., Perrin, D.: A quadratic upper bound on the size of a synchronizing word in one-cluster automata. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 81–90. Springer, Heidelberg (2009)
3. Carpi, A., D'Alessandro, F.: The synchronization problem for strongly transitive automata. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 240–251. Springer, Heidelberg (2008)
4. Carpi, A., D'Alessandro, F.: Strongly transitive automata and the Černý conjecture. Acta Informatica 46, 591–607 (2009)
5. Černý, J.: Poznámka k homogénnym eksperimentom s konečnými automatami. Matematicko-fyzikalny Časopis Slovensk. Akad. Vied. 14(3), 208–216 (1964) (in Slovak)
6. Dubuc, L.: Sur les automates circulaires et la conjecture de Černý. RAIRO Inform. Théor. Appl. 32, 21–34 (1998) (in French)
7. Eppstein, D.: Reset sequences for monotonic automata. SIAM J. Comput. 19, 500–510 (1990)
8. Frankl, P.: An extremal problem for two families of sets. Eur. J. Comb. 3, 125–127 (1982)
9. Kari, J.: A counter example to a conjecture concerning synchronizing words in finite automata. EATCS Bull. 73, 146 (2001)
10. Kari, J.: Synchronizing finite automata on Eulerian digraphs. Theoret. Comput. Sci. 295, 223–232 (2003)
11. Pin, J.-E.: Le problème de la synchronization et la conjecture de Cerny, Thèse de 3ème cycle. Université de Paris 6 (1978)
12. Pin, J.-E.: On two combinatorial problems arising from automata theory. Ann. Discrete Math. 17, 535–548 (1983)
13. Rystsov, I.: Quasioptimal bound for the length of reset words for regular automata. Acta Cybernetica 12, 145–152 (1995)
14. Rystsov, I.: Exact linear bound for the length of reset words in commutative automata. Publ. Math. Debrecen 48, 405–411 (1996)
15. Rystsov, I.: Reset words for commutative and solvable automata. Theor. Comput. Sci. 172, 273–279 (1997)
16. Rystsov, I: Reset words for automata with simple idempotents, Kibernetika i Sistemnyj Analiz No. 3, 32–39 (2000) (in Russian; English translation: Cybernetics and System Analysis 36, 339–344 (2000))
17. Steinberg, B.: The averaging trick and the Černý conjecture, arXiv:0910.0410 (2009)
18. Trahtman, A.: An efficient algorithm finds noticeable trends and examples concerning the Černý conjecture. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 789–800. Springer, Heidelberg (2006)
19. Trahtman, A.: The Černý conjecture for aperiodic automata. Discrete Math. Theor. Comput. Sci. 9(2), 3–10 (2007)
20. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)
21. Volkov, M.V.: Synchronizing automata preserving a chain of partial orders. Theoret. Comput. Sci. 410, 2992–2998 (2009)

Linking Algebraic Observational Equivalence and Bisimulation

Mouhebeddine Berrima[1] and Narjes Ben Rajeb[2]

[1] LIP2, Faculté des Sciences de Tunis
berrima.mouheb@gmail.com
[2] LIP2, Institut National des Sciences Appliquées et de Technologie, Tunis
narjes.benrajeb@wanadoo.tn

Abstract. Observability concepts allow to focus on the observable behaviour of a system while abstracting internal details of implementation. In this context, formal verification techniques use behavioural equivalence notions formalizing the idea of indistinguishability of system states. In this paper, we investigate the relation between two behavioural equivalences: the algebraic observational equivalence in the framework of observational algebras with many hidden sorts and automata bisimulation. For that purpose, we propose a transformation of an observational algebra into an infinite deterministic automaton. Consequently, we obtain a subclass of deterministic automata, equivalent to (infinite) Mealy automata, which we call Observational Algebra Automata (OAA). We use a categorical setting to show the equivalence between bisimulation on OAA and algebraic observational equivalence. Therefore we extend the hidden algebras result concerning observational equivalence and bisimulation coincidence to the non-monadic case.

1 Introduction

The increasing importance of programming has led to the development of formal methods of specification and verification. Classically, a program -or more generally a system- is correct if it is proved that it satisfies all the properties enounced in its specification. However, it has appeared that this is not necessary: it is sufficient that the system satisfies only *observable* properties which determine its external behaviour. Based on this viewpoint, different notions of behavioural equivalence have emerged, formalizing the idea of indistinguishability of system states or objects. In formal approaches, there are two important notions: algebraic observational equivalence in the context of "behavioural algebraic specification" (Hidden Algebra [10], coherent hidden algebra [8], Coalgebra [25], Observational Logic [11], etc...), and bisimulation in the context of process algebras (π-calculus [20], CCS [19], CSP [12], etc ...).

Algebraic observational equivalence, that have been introduced by H. Reichel [22], is the key notion on which is based the observational algebraic frameworks to build observational semantics. These frameworks rely on the observational algebras which introduce the notion of non observable elements (also called states or objects). In literature, one can distinguish two main definitions of observational equivalence. One is defined between elements of an observational algebra (e.g [10,11]), through experiments with observable results (w.r.t a set of observable sorts). These experiments are formalized by a notion of *context*, which is a term containing a distinguished variable to be replaced by some objects. Thus, two objects are observationally equivalent (or equal) if they cannot be distinguished by a context of observable result. Other approaches define observational equivalence between algebras (e.g [5,21,28]). The basic idea is that two algebras are considered to be observationally equivalent if they cannot be distinguished by a set of observations (also w.r.t a set of observable sorts). These observations are formalized by equations between terms.

Bisimulation is a rich concept which appears in various areas of theoretical computer science. It was introduced by R. Milner in CCS language [19]. Bisimulation of automata (or transition systems) have been considered after the work of R. Milner [19]. Many variants of automata have been proposed with a suitable definition of bisimulation (e.g [15,6,7,27]). In all models, the bisimulation expresses the equivalence behaviour of states in response to a sequence of actions. Intuitively, two states are bisimilar if each action starting from one state can be done by the other and leads again to bisimilar states.

As related works, some links were observed between bisimulation and observational equivalence in some contexts which consider one hidden sort; hidden algebra [17], coalgebra framework [26] and object-oriented algebraic specification [13]. Moreover, in [16] the author showed that the bisimilarity defined by means of open maps [14] coincides with behavioural equivalence between regular and standard algebras.

In this paper, we are interested in studying the relation between bisimulation and the observational equivalence following observational logic framework [11], that generalizes hidden algebra by dropping the monadicity requirement which stipulates that operations may take at most one hidden argument. For hidden algebra, it is already known that observational equivalence and bisimulation coincide [17]. Moreover, in [4] the authors have argued the coincidence between observational equivalence following [11] and bisimulation, but they have required a restriction forbidding some operations (called *direct observers*) with more than one hidden argument. In this work, we don't require any restriction on the non-monadicity property: all operations defined in the framework of observational logic are allowed. Our work can be considered also as related work to [10] (Section 3.4) in which the authors showed that models of anemic signature [1] correspond to labeled transition systems, and consequently have proved the coincidence between the observational equivalence and bisimulation. We propose an explicit construction which transforms a given observational algebra into an infinite deterministic automaton. This defines a subclass of infinite deterministic automata which we call "Observational Algebra Automata" (OAA). Regardless of infiniteness, our automaton is). equivalent to Mealy automaton [18]. The idea

[1] An anemic signature is a hidden signature with no hidden constants and with just two sorts, h hidden and v visible.

is to correspond a state to each non observable element and a sequence of transitions to each context. We formalize this transformation in categorical setting via a functor. This is useful since it allows us in further work, to recognize the connection between observational algebras and automata theory, moreover many results about observational equivalence can be translated across this functor to give results about bisimulation. Finally, we show the coincidence between observational equivalence (between elements and between algebras) and bisimulation. The main conceptual advance of this paper is to extend the hidden algebra result concerning observational equivalence and bisimulation coincidence to the non-monadic case. To the best of our knowledge, no study on the relation between the notion of observational equivalence between algebras and bismulation has yet been done in literature. The interest of such work is to link proof techniques based on both notions.

The paper is organized as follows: Section 1 introduces the main concepts related to algebraic observational equivalence. Section 2 presents our transformation of observational algebra into an OAA and defines the bisimulation. Section 3 formalizes this transformation using a functor and gives the main ideas for proving the equivalence between bisimulation and observational equivalence. The paper concludes in Section 6. We suppose the reader is familiar with algebraic specifications [9] and category theory [1].

2 Algebraic Observational Equivalence

In this section, we introduce the main definitions and concepts related to the notion of algebraic observational equivalence following [11].

An Σ-algebra $A = ((A_s)_{s\in S}, (op^A)_{op\in OP})$ over a signature $\Sigma = (S, OP)$, with a set S of sorts and a set OP of operation symbols, is a family $(A_s)_{s\in S}$ of sets A_s for all $s \in S$ called supports, and a family $(op^A)_{op\in OP}$ of functions $op^A : A_{s_1} \times A_{s_2} \times \ldots \times A_{s_n} \to A_s$ if op has the profile $op : s_1, s_2, \ldots, s_n \to s$. A particular algebra is the term algebra $T(\Sigma, X)$. An interpretation in an algebra A is a function I_α from $T(\Sigma, X)$ to A defined w.r.t a valuation $\alpha : X \to A$. A Σ-congruence relation on a Σ-algebra A, denoted by $\simeq_A$, is a family $(\simeq_s)_{s\in S}$ of equivalence relations which is compatible with the operations of A. A quotient of an Σ-algebra A by a Σ-congruence $\simeq$ is defined by $A/\simeq = \{[a] \mid a \in A\}$ where $[a] = \{a' \in A \mid a \simeq a'\}$. We use the following notation: let $a_i \in A_{s_i}$ for $i = 1..n$, we write $a_{1..n}$ instead of $a_1, \ldots, a_n$. Similarly, we write $f(a_{1..n})$ instead of $f(a_1), \ldots, f(a_n)$ for some symbol function f.

Observational algebraic specifications are algebraic specifications equipped with an observation technique [2], allowing to specify the observed terms.

Definition 1 (Observational signature). *An observational signature Σ_{Obs} is defined by (S_{state}, S_{Obs}, OP) where S_{state} is the set of non-observable sorts, S_{Obs} is the set of observable sorts and OP is the set of operations specified by their profiles. We also denote an observational signature by $\Sigma_{Obs} = (S, OP)$ where $S = S_{Obs} \cup S_{State}$.*

An algebra over an observational signature Σ_{Obs} is called observational Σ_{Obs}-algebra. Given an observational signature $\Sigma_{Obs} = (S, OP)$, we use meta-variables A, B, C to range over Σ_{Obs}-algebras.

Definition 2 (Observational specification). *An observational specification is a pair (Σ_{Obs}, Ax) such that Σ_{Obs} is an observational signature and Ax is a set of equations.*

Example 1. The following is a simple observational specification for sets, with *in* as membership, *add* placing a data to a set, U gives the union of two sets, and & giving the intersection of two sets.

Spec SET	
Sorts {Set, Data, Bool }	**Axioms :** Var N, N' : Int Var X X' : Set
Observable sorts {Bool, Data}	$in(N, empty) = false$
Operations :	$in(N, add(N', X)) = (N == N')or(in(N, X))$
$a, b, c : \rightarrow Data$	$in(N, U(X, X') = (in(N, X))or(in(N, X'))$
$empty : \rightarrow Set$	$in(N, U(X, X') = (in(N, X))and(in(N, X'))$
$true : \rightarrow Bool$	
$false : \rightarrow Bool$	
$in : Data \times Set \rightarrow Bool$	
$add : Data \times Set \rightarrow Set$	
$U : Set \times Set \rightarrow Set$	
$\& : Set \times Set \rightarrow Set$	

Definition 3 (Observer). *An observer is a pair (op, i) where $op \in OP$ has the profile $op : s_1, \ldots, s_i, \ldots, s_n \rightarrow s$ and i is an argument position of non observable sort. An observer (op, i) is direct if $s \in S_{Obs}$ otherwise it is indirect ($s \in S_{State}$).*

In the following, we denote by OP_{Obs} the set of all observers, by OP_{Obs-D} the set of direct observers and by OP_{Obs-I} the set of indirect observers.

Definition 4 (Context). *A context $c[\diamond_s]$ is a non ground term $c \in T(\Sigma_{Obs}, OP)$ containing only one occurrence of a distinguished variable called context variable and denoted by $\diamond_s$ (where s is the sort of $\diamond$). The arity of a context $c[\diamond_s]$ is $s \rightarrow s'$ where s' is result sort of c and s is its argument sort. The application of a context $c[\diamond_s]$ of an arity $s \rightarrow s'$ to a term t of sort s, denoted par c[t], is the term obtained by substituting $\diamond_s$ by t. An empty context is a context which is a context variable.*

Definition 5 (Observable context). *An observable context is a context c of arity $s \rightarrow s'$ where $s' \in S_{Obs}$. We denote by $C(\Sigma_{Obs})_{s \rightarrow S_{Obs}}$ the set of observable contexts with argument sort s.*

Elements which can not be distinguished by experimentations with observable result are considered observationally equivalent (or equal). This equivalence expresses an indistinguishability relation between non observable elements.

Definition 6 (Observational Σ_{Obs}-equivalence). *[11] Let Σ_{Obs} be an observational signature. For any Σ_{Obs}-algebra A the observational Σ_{Obs}-equivalence* *(or equality) on A, denoted by $\approx_A$, is defined by: For all $s \in S$, two elements $a, b \in A_s$ are observationally equivalent (equal) w.r.t. Σ_{Obs}, i.e. $a \approx_A b$, if and only if for all observable contexts $c \in C(\Sigma_{Obs})_s \to S_{Obs}$ and for all valuations $\alpha, \beta : X \cup \{\diamond_s\} \to A$ with $\alpha(x) = \beta(x)$ if $x \in X, \alpha(\diamond_s) = a, \beta(\diamond_s) = b$, we have $I_\alpha(c) = I_\beta(c)$. Obviously, if s is an observable sort, then for all $a, b \in A, a \approx_A b$ is equivalent to $a = b$.*

As mentioned in [11], $\approx_A$ is an equivalence relation on A, but in general not Σ_{Obs}-congruence.

3 Observational Algebra Automata (OAA)

The idea of modeling an observational algebra by an automaton, is to consider non observable elements as states of the automaton and observable elements as the output alphabet. Moreover, transitions represent contexts built from indirect observers, and the results of output functions represent the results of applying observable contexts. Note that such a modeling gives rise to our definition of an infinite deterministic automaton. Regardless of infiniteness, our automaton is equivalent to Mealy automaton [18].

Definition 7 (Observational Algebra Automata). *Let A be an observational Σ_{Obs}-algebra defined over $\Sigma_{Obs} = (S_{State}, S_{Obs}, OP)$ and $a_j \in A_{s_j}$ for $j = 1..n$. The observational Σ_{Obs}-algebra automata (simply written Σ_{Obs}-OAA) associated to A, denoted by $\mathcal{A}_A$, is defined by the 5-tuple $(I_A, O_A, E_A, \delta_A, \lambda_A)$ where:*

- *I_A is a set of elements of the form $op^A(a_{1..j-1}, \diamond_{s_j}, a_{j+1..n})$ where $(op, j) \in OP_{Obs}$.*
- *$O_A = A_{S_{Obs}}$, where $A_{S_{Obs}} = (A_s)_{s \in S_{Obs}}$.*
- *$E_A = A_{S_{State}}$, where $A_{S_{State}} = (A_s)_{s \in S_{State}}$.*
- *$\lambda_A : E_A \times I_A \to O_A \cup \{\epsilon\}$ is defined by: for each a of sort s_j and $i \in I_A$ with $i =_{def} op^A(a_{1..j-1}, \diamond_{s_j}, a_{j+1..n})$;*

$$\lambda_A(a, i) = \begin{cases} a' & \text{if } op^A(a_{1..j-1}, a, a_{j+1..n}) = a', \\ \epsilon & \text{otherwise.} \end{cases}$$

- *$\delta_A : E_A \times I_A \to E_A$ is defined by: for each a of sort s_j and $i \in I_A$ with $i =_{def} op^A(a_{1..j-1}, \diamond_{s_j}, a_{j+1..n})$;*

$$\delta_A(a, i) = \begin{cases} a' & \text{if } op^A(a_{1..j-1}, a, a_{j+1..n}) = a', \\ a & \text{otherwise.} \end{cases}$$

Due to the infinity of elements of observational algebras, the sets I_A, E_A and O_A are infinite. Therefore, we can consider observational algebra automata among the infinite-branching automata. Moreover, according to above definition, we remark that the input alphabet relies on the corresponding algebra, then we derive that each OAA has its own input alphabet.

Example 2. To illustrate the construction of an OAA, consider the SET specification of example 1. Consider the observational algebra A representing an implementation of $Data$ using characters, $Bool$ using strings and Set using a infinite lists of $Data$: thus, $empty$ is the empty list, $add(d, s)$ placing a data d at the front of list s, $U(s_1, s_2)$ appending s_2 at the end of s_1, and $\&(s_1, s_2)$ giving a list containing each element in s_1 and also in s_2. Note that in this example the output alphabet is finite. The OAA corresponding to algebra A is defined as follows:

– $O_A = \{true, false, a, b, c\}$
– $E_A = \{e_0, e_1, e_2, e_3, e_4, e_5, ...\}$ with $e_0 = empty$, $e_1 = add(a, empty)$, $e_2 = add(b, empty)$, $e_3 = add(c, empty)$, $e_4 = add(a, add(b, empty))$ and $e_5 = add(b, add(a, empty))$.
– $I_A = \{i_0, i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8, i_9, i_{10}, ...\}$ with $i_0 = in(a, \diamond_{Set})$, $i_1 = in(b, \diamond_{Set})$, $i_2 = in(c, \diamond_{Set})$, $i_3 = add(a, \diamond_{Set})$, $i_4 = add(b, \diamond_{Set})$, $i_5 = add(c, \diamond_{Set})$, $i_6 = U(e_1, \diamond_{Set})$, $i_7 = U(e_2, \diamond_{Set})$, $i_8 = U(\diamond_{Set}, e_2)$, $i_9 = U(\diamond_{Set}, e_1)$ and $i_{10} = \&(\diamond_{Set}, e_4)$.
– $\lambda(e_0, i_0) = false$, $\lambda(e_1, i_0) = true$, $\lambda(e_1, i_3) = \epsilon$, etc...
– $\delta(e_0, i_3) = e_1$, $\delta(e_0, i_4) = e_2$, $\delta(e_0, i_5) = e_3$, $\delta(e_1, i_8) = e_4$, $\delta(e_1, i_7) = e_5$, $\delta(e_2, i_6) = e_4$, $\delta(e_2, i_9) = e_5$, etc....

Bisimulation on OAA is a S_{State}-sorted relation that takes into account the difference of input alphabets.

Definition 8 (Bisimulation). *A bisimulation between two Σ_{Obs}-OAA $\mathcal{A}_A = (I_A, O_{\Sigma_{Obs}}, E_A, \delta_A, \lambda_A)$ and $\mathcal{A}_B = (I_B, O_{\Sigma_{Obs}}, E_B, \delta_B, \lambda_B)$ is a S_{State}-sorted relation $R \subseteq E_A \times E_B$ such that there exists a symmetric relation $R_I \subseteq I_A \times I_B$ defined by $(i, i') \in R_I$ if $f(i) = i'$ for some function $f : I_A \to I_B$, and for all $a \in E_A, b \in E_B$: if $(a, b) \in R$ then:*

1. *$\forall\ i \in I_A$ and $i' \in I_B$: $\lambda_A(a, i) = \lambda_B(b, i')$ whenever $(i, i') \in R_I$.*
2. *$\forall\ i \in I_A$: if $\delta_A(a, i) = a'$, then there exist $b' \in E_B$ and $i' \in I_B$ such that $\delta_B(b, i') = b'$, $(a', b') \in R$ and $(i, i') \in R_I$.*
3. *$\forall\ i \in I_B$: if $\delta_B(b, i) = b'$, then there exist a' in E_A and $i' \in I_A$ such that $\delta_A(a, i') = a'$, $(a', b') \in R$ and $(i, i') \in R_I$.*

A bisimulation relation between $\mathcal{A}_A$ and itself is called a bisimulation on $\mathcal{A}_A$. Union and composition of two bisimulations are bisimulations. We write $a \sim_{\mathcal{A}_A} a'$ whenever there exists a bisimulation R on $\mathcal{A}_A$ with $(a, a') \in R$. This relation is called bisimilarity relation and it is the union of all bisimulations; so it is the greatest bisimulation. It is easy to see that bisimilarity is an equivalence relation. Two OAA $\mathcal{A}_A = (I_A, O_{\Sigma_{Obs}}, E_A, \delta_A, \lambda_A)$ and $\mathcal{A}_B = (I_B, O_{\Sigma_{Obs}}, E_B, \delta_B, \lambda_B)$ are said to be bisimilar, denoted by $\mathcal{A}_A \sim \mathcal{A}_B$, if exists a bisimulation relation R such that for each $a \in E_A$ exists $b \in E_B$ with $(a, b) \in R$, and for each $b \in E_B$ exists $a \in E_A$ with $(a, b) \in R$, i.e the projections $p_1 : R \to E_A$ and $p_1 : R \to E_B$ are surjective.

It is easy to verify that, if $\mathcal{A}_A$ and $\mathcal{A}_B$ have the same input alphabet, then by choosing f the identity function, the bismulation between two OOA becomes equivalent to the usual definition of bisimulation in classical automata theory.

4 Relating Observational Equivalence and Bisimulation

The aim of this section is to formally link observational equivalence on observational algebras with bisimulation on associated OAA. For this purpose, we define a functor from the category of observational algebras to the category of OAA. Functoriality proves that the construction has been properly formalized and it can be surprisingly helpful in practice.

From now on, we assume that the observational Σ_{Obs}-algebras have common observable elements; in typical applications, observable elements might include natural numbers, booleans, character strings, etc... However, there could be multiple representations for observable elements with translations among them, but our assumption can easily be relaxed to cover such cases.

4.1 The Transformation Functor

A homomorphism between two observational algebras is a function that preserves algebraic structure. It is defined as usual:

Definition 9 (Σ_{Obs}-homomorphism). *Let A, B two Σ_{Obs}-algebras. A Σ_{Obs}-homomorphism $h : A \rightarrow B$ is a S-sorted function such that $h_s(op^A(a_1, \dots, a_n)) = op^B(h_{s_1}(a_1), \dots, h_{s_n}(a_n))$ for $op \in OP$ and $a_i \in A_{s_i}$ when op has the profile $op : s_1, \dots, s_n \rightarrow s$.*

Given an observational signature Σ_{Obs}, we consider the category of Σ_{Obs}-algebras, denoted by $Alg(\Sigma_{Obs})$, where objects are Σ_{Obs}-algebras and morphisms are Σ_{Obs}-homomorphisms. The identity functions are Σ_{Obs}-homomorphisms and the functional composition of two Σ_{Obs}-homomorphisms is a Σ_{Obs}-homomorphism.

Since we deal with Σ_{Obs}-algebras having the same observable elements, the Σ_{Obs}-homomorphisms are identity functions on observable elements, i.e. for each Σ_{Obs}-homomorphism $h : A \rightarrow B$, $h_s : A_s \rightarrow B_s$ are identity functions for all $s \in S_{Obs}$. Hence, we deal with OAA of Σ_{Obs}-algebras having the same output alphabet, that we denote by $O_{\Sigma_{Obs}}$.

A homomorphism between OAA is a function that preserves transitions and outputs. Let $\mathcal{A}_A = (I_A, O_{\Sigma Obs}, E_A, \delta_A, \lambda_A)$ and $\mathcal{A}_B = (I_B, O_{\Sigma Obs}, E_B, \delta_B, \lambda_B)$ be two OAA. We mean by a function from $\mathcal{A}_A$ to $\mathcal{A}_B$ a function $f : E_A \cup I_A \rightarrow E_B \cup I_B$ such that for each $e \in E_A \cup I_A$:

$$f(e) = \begin{cases} e' \in E_B & \text{if } e \in E_A, \\ e' \in I_B & \text{if } e \in I_A. \end{cases}$$

Definition 10 (OAA-homomorphism). *Let Σ_{Obs} be an observational signature, $\mathcal{A}_A = (I_A, O_{\Sigma Obs}, E_A, \delta_A, \lambda_A)$ and $\mathcal{A}_B = (I_B, O_{\Sigma Obs}, E_B, \delta_B, \lambda_B)$ be two Σ_{Obs}-OAA. A Σ_{Obs}-OAA-homomorphism from $\mathcal{A}_A$ to $\mathcal{A}_B$ is a function $f : \mathcal{A}_A \rightarrow \mathcal{A}_B$ such that for all $e \in E_A$ and $i \in I_A$ the following hold:*

- $f(\delta_A(e, i)) = \delta_B(f(e), f(i))$,
- $\lambda_A(e, i) = \lambda_B(f(e), f(i))$.

Proposition 1. *Let $f : \mathcal{A}_A \rightarrow \mathcal{A}_B$ and $g : \mathcal{A}_B \rightarrow \mathcal{A}_C$ be two OAA-homomorphisms. Then the composition $g \circ f : \mathcal{A}_A \rightarrow \mathcal{A}_C$ is an OAA-homomorphism.*

For any Σ_{Obs}-OOA $\mathcal{A}_A = (I_A, O_{\Sigma Obs}, E_A, \delta_A, \lambda_A)$, we have the automaton identity function $id_{\mathcal{A}_A} : E_A \cup I_A \rightarrow E_A \cup I_A$. It is immediate that this indeed defines an OAA-homomorphism. Then, the Σ_{Obs}-OAA with Σ_{Obs}-OAA-homomorphisms form a category, denoted by $OAA(\Sigma_{Obs})$.

The mapping from observational algebras structures to OAA structures is realized by a functorial function associating to each observational Σ_{Obs}-algebra its corresponding Σ_{Obs}-OAA, and to each Σ_{Obs}-homomorphism its corresponding Σ_{Obs}-OAA-homomorphism.

Definition 11 (Transformation functor $\mathcal{F}_{\Sigma_{Obs}}$). *Let Σ_{Obs} be an observational signature, $\mathcal{A}_A = (I_A, O_{\Sigma Obs}, E_A, \delta_A, \lambda_A)$ and $\mathcal{A}_B = (I_B, O_{\Sigma Obs}, E_B, \delta_B, \lambda_B)$ be two Σ_{Obs}-OOA. The functor $\mathcal{F}_{\Sigma_{Obs}} : Alg(\Sigma_{Obs}) \rightarrow OAA(\Sigma_{Obs})$ is defined by: For each $A \in Alg(\Sigma_{Obs})$; $\mathcal{F}_{\Sigma_{Obs}}(A) =_{def} \mathcal{A}_A$, For each Σ_{Obs}-homomorphism $h : A \rightarrow B$; $\mathcal{F}_{\Sigma_{Obs}}(h) =_{def} f : \mathcal{A}_A \rightarrow \mathcal{A}_B$ defined by: $f(a) = b$ if $h(a) = b$ for all $a \in E_A$,*
$f(op^A(a_{1..i-1}, \diamond_{s_i}, ai+1..n)) = op^B(h(a_{1..i-1}), \diamond_{s_i}, h(a_{i+1..n}))$ for all $op^A(a_{1..i-1}, \diamond_{s_i}, a_{i+1..n}) \in I_A$.

It easy to verify that the mapping from I_A to I_B is induced by the Σ_{Obs}-homomorphism. We omit the subscript Σ_{Obs} whenever there is no confusion.

Proposition 2. *The functor $\mathcal{F}$ is well-defined.*

The functor $\mathcal{F}$ establishes a one-to-one correspondence between Σ_{Obs}-homomorphisms and OAA-homomorphisms. This interesting property is expressed in category theory by the following notion:

Lemma 1. *The functor $\mathcal{F}$ is full and faithful.*

4.2 Observational Equivalence versus Bisimulation

Our main contribution is to prove observational equivalence and bisimulation coincidence. We first state and prove some preliminary results. The next lemma shows the strong connection between OAA-bisimulation and OAA-homomorphism. This connection is already known in coalgebraic setting [26] and for this reason, the coalgebras homomorphism are sometimes called functional bisimulation.

Lemma 2. *Let Σ_{Obs} be an observational signature, $\mathcal{A}_A, \mathcal{A}_B \in OAA(\Sigma_{Obs})$ and $f : \mathcal{A}_A \rightarrow \mathbf{A}_B$ be any function. The following are equivalent:*

1. *f is an OAA-homomorphism.*
2. *For all $a \in E_A$ and $i \in I_A$,*
   (a) If $\delta_A(a, i) = a'$ for some $a' \in E_A$, then $\delta_B(f(a), f(i)) = f(a')$.
   (b) If $\lambda_A(a, i) = o$ for some $o \in O_{\Sigma_{Obs}}$, then $\lambda_B(f(a), f(i)) = o$.

(c) If $\delta_B(f(a), f(i)) = b$ for some $b \in E_B$, then there exists $a' \in E_A$ such that $\delta_A(a,i) = a'$ and $f(a') = b$.
(d) If $\lambda_B(f(a), f(i)) = o$ for some $o \in O_{\Sigma_{Obs}}$, then $\lambda_A(a,i) = o$.

Proof. Immediate from the observation that the first condition of the definition of an OAA-homomorphism (Definition 10) is equivalent to clauses (a) and (c), and the second condition is equivalent to clauses (b) and (d). □

Let $f : \mathcal{A}_A \to \mathcal{A}_B$ be an OAA-homomorphism. The kernel $K(f)$ and the graph $G(f)$ of f are defined as follows:

$$K(f) = \{(a, a') \in E_A \times E_A \mid f(a) = f(a')\},$$

$$G(f) = \{(a, b) \in E_A \times E_B \mid f(a) = b\}.$$

Proposition 3. *Let Σ_{Obs} be an observational signature, $\mathcal{A}_A, \mathcal{A}_B \in OAA(\Sigma_{Obs})$ and $f : \mathcal{A}_A \to \mathcal{A}_B$ be any function, then we have:*
1. f is an OAA-homomorphism iff $G(f)$ is bisimulation between $\mathcal{A}_A$ and $\mathcal{A}_B$.
2. If f is an OAA-homomorphism then $K(f)$ is a bisimulation on $\mathcal{A}_A$.

Proof. By choosing $(i, i') \in R_I$ if $f(i) = i'$ with $i \in I_A$ and $i' \in I_B$, the statement 1 becomes immediate by the Definition 8 and Lemma 2. The statement 2 follows from the observation that $K(f) = G(f) \circ G(f)^{-1}$ and the fact that a bisimulation is symmetric and the composition of two bisimulations is again a bisimulation. □

Since we are interested to observational equivalence between non-observable elements, we consider a Σ_{Obs}-congruence as a family $\simeq = (\simeq_s)_{s \in S_{State}}$ of equivalence relations. The next result shows the correspondence between a bisimulation on OAA and a Σ_{Obs}-congruence on its corresponding observational algebra.

Theorem 1. *Let Σ_{Obs} be an observational signature, $A \in Alg(\Sigma_{Obs})$, $a, b \in A$ be two non-observable elements, and $\mathcal{A}_A \in OAA(\Sigma_{Obs})$ such that $\mathcal{A}_A = (I_A, O_{\Sigma_{Obs}}, E_A, \delta_A, \lambda_A)$ and $\mathcal{F}(A) =_{def} \mathcal{A}_A$. We have $a \simeq_A b$ iff there exists a bisimulation $R \subseteq E_A \times E_A$ such that $(a, b) \in R$.*

Proof. In this proof, we use the notion of the kernel which describes the notion of congruence. Let A be an observational algebra and assume that there exists a Σ_{Obs}-congruence $\simeq$ on A. We describe this congruence by $K(f)$ with $f : A \to A/\simeq$ is mapping quotient. It is easy to see that f is a Σ_{Obs}-homomorphism. Then we have $\mathcal{F}(f) : \mathcal{F}(A) \to \mathcal{F}(A/\simeq)$ is an OAA-homomorphism. Thus, by the Proposition 3, we conclude that $K(\mathcal{F}(f))$ is a bisimulation. The converse is straightforward since $\mathcal{F}$ is full and faithful. □

We deduce that every Σ_{Obs}-congruence on an observational algebra gives rise to a bisimulation on the corresponding OAA, and vice versa.

In [24], has been proved for non-monadic signature that observational equivalence, when it is a congruence, is the greatest congruence. Then, it follows that observational equivalence, when it is a congruence, coincides with the bisimilarity.

Corollary 1. *Let Σ_{Obs} be an observational signature, $A \in Alg(\Sigma_{Obs})$ and $\mathcal{A}_A \in OAA(\Sigma_{Obs})$ such that $\mathcal{F}(A) =_{def} \mathcal{A}_A$. If $\approx_A$ is a Σ_{Obs}-congruence, then we have $\approx_A = \sim_{\mathcal{A}_A}$.*

Proof. Immediate from the Theorem 1 and the observation that observational equivalence is the greatest Σ_{Obs}-congruence, and bisimilarity is the greatest bisimulation. □

Example 3. Consider again the algebra A and its corresponding OAA defined in Example 2. We have the states e_4 and e_5 are bisimilar. These states correspond respectively to the elements $U^A(add^A(a, empty), add^A(b, empty))$ and $U^A(add^A(b, empty), add^A(a, empty))$ which are observationally equivalent. This means that two lists are observationally equivalent iff contain exactly the same data, without regard to order or number of occurrences.

Our second main contribution, is to link bisimulation and observational equivalence between algebras which have been widely studied in the frameworks of many-sorted algebras. As mentioned earlier, one considers two algebras to be observationally equivalent if they cannot be distinguished by a predefined set of observations given by an appropriate set of terms. Several definitions have been proposed, e.g [5,21,28]. We use the definition of [5] that is a generalization of the notion of observational equivalence between algebras.

Definition 12. *Let A, B two observational Σ_{Obs}-algebras. Then A and B are called observationally equivalent w.r.t S_{Obs} and S_{In}, denoted by $A \equiv_{S_{Obs}, S_{In}} B$, if there exists a S-sorted family Y_{In} of sets $(Y_{In})_s$ of variables of sort s with $(Y_{In})_s = \emptyset$ for all $s \notin S_{In}$, $(Y_{In})_s \neq \emptyset$ for all $s \in S_{In}$ and if there exist two valuations $\alpha 1 : Y_{In} \rightarrow A$ and $\beta 1 : Y_{In} \rightarrow B$ with surjective mappings $\alpha 1_s : (Y_{In})_s \rightarrow A_s$ and $\beta 1_s : (Y_{In})_s \rightarrow B_s$ for all $s \in S_{In}$ such that for all terms $t, r \in T(\Sigma_{Obs}, Y_{In})_s$ of observable sort $s \in S_{Obs}$ the following holds: $I_{\alpha 1}(t) = I_{\alpha 1}(r)$ iff $I_{\beta 1}(t) = I_{\beta 1}(r)$.*

Given an observational signature $\Sigma_{Obs} = (S, OP)$, it can be shown that observational equivalence between algebras, when $S_{In} = S$, corresponds to the bisimilarity between their corresponding OAA. For the proof, we will need to the next lemma which expresses the bisimilarity between two OAA in terms of OAA-homomorphisms.

Lemma 3. *Let Σ_{Obs} be an observational signature and $\mathcal{A}_A, \mathcal{A}_B, \mathcal{A}_C \in OAA(\Sigma_{Obs})$. If there exist $f : \mathcal{A}_C \rightarrow \mathcal{A}_A$ and $g : \mathcal{A}_C \rightarrow \mathcal{A}_B$ two surjective OAA-homomorphisms, then $\mathcal{A}_A \sim \mathcal{A}_B$.*

We are now ready to state the result concerning observational equivalence between algebras and bisimulation coincidence.

Theorem 2. *Let Σ_{Obs} be an observational signature, $A, B \in Alg(\Sigma_{Obs})$ and $\mathcal{A}_A$, $\mathcal{A}_B \in OAA(\Sigma_{Obs})$ such that $\mathcal{F}(A) =_{def} \mathcal{A}_A$ and $\mathcal{F}(B) =_{def} \mathcal{A}_B$. We have $A \equiv_{S_{Obs}, S} B$ iff $\mathcal{A}_A \sim \mathcal{A}_B$.*

5 Conclusion

In this work, our aim was to study the link between two concepts of behavioural equivalence: bisimulation and algebraic observational equivalence. We proposed a transformation of observational algebras into infinite deterministic automata. One can consider these automata as infinite Mealy automata. Therefore, we characterized a subclass of deterministic automata which we called Observational Algebra Automata (OAA). The formulation of the transformation of observational algebras to OOA in the categorical setting via a functor, allowed us to establish elegant connection between two fields of research, observational algebraic specification and automata theory. This connection is useful (in further works) for discovering and exploiting relations between the two fields. Based on the transformation functor and elementary constructions, we have proved that bisimulation plays the role of observational equivalence between non-observable elements. This result can be viewed as a generalization of the result of [17] for hidden algebra. We have also proved the coincidence between observational equivalence between algebras and bisimulation. If such a comparison appears natural, to our knowledge, no proof has been formally written using automata.

It would be also interesting to extend this work by considering partial observational equivalence [5] and other observation techniques based on operations or terms [2]. Another future work would be to develop proof techniques combining those for automata theory and those for frameworks related to observational algebras as Observational Logic [11], COL [3], etc...

References

1. Asperti, A., Longo, G.: Categories, types, and structures: an introduction to category theory for the working computer scientist. MIT Press, Cambridge (1991)
2. Bernot, G., Bidoit, M., Knapik, T.: Behavioural Approaches to Algebraic Specifications: A Comparative Study. Acta Inf. 31(7), 651–671 (1994)
3. Bidoit, M., Hennicker, R.: Constructor-Based Observational Logic. Journal of Logic and Algebraic Programming 67(1-2), 3–51 (2006)
4. Bidoit, M., Hennicker, R., Kurz, A.: Observational logic, constructor-based logic, and their duality. Theoretical Computer Science 298(3), 471–501 (2003)
5. Bidoit, M., Hennicker, R., Wirsing, M.: Behavioural and abstractor specifications. Science of Computer Programming 25(2-3), 149–186 (1995)
6. Buchholz, P.: Bisimulation relations for weighted automata. Theoretical Computer Science 393(1-3), 109–123 (2008)
7. Caucal, D.: Bisimulation of Context-Free Grammars and of Pushdown Automata. CSLI Lecture Notes 53, 85–106 (1995)
8. Diaconescu, R., Futatsugi, K.: Behavioural Coherence in Object-Oriented Algebraic Specification. In: Calude, C.S., Stefanescu, G. (eds.) Journal of Universal Computer Science, vol. 6(1), pp. 74–96 (2000)
9. Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification I. Springer, New Work (1985)
10. Goguen, J., Malcolm, G.: A hidden agenda. Theoretical Computer Science 245(1), 55–101 (2000)

11. Hennicker, R., Bidoit, M.: Observational logic. In: Haeberer, A.M. (ed.) AMAST 1998. LNCS, vol. 1548, pp. 263–277. Springer, Heidelberg (1998)
12. Hoare, C.A.R.: Communicating sequential processes. Prentice-Hall, Englewood Cliffs (1985)
13. Jacobs, B.: Objects and classes, co-algebraically, Object orientation with parallelism and persistence. Kluwer Academic Publishers, Norwell (1996)
14. Joyal, A., Nielsen, M., Winskel, G.: Bisimulation and open maps. In: LICS: IEEE Symposium on Logic in Computer Science, Montreal, Canada, pp. 418–427 (2003)
15. Kumar, R., Zhou, C., Basu, S.: Finite Bisimulation of Reactive Untimed Infinite State Systems Modeled as Automata with Variables. In: Proc. of 2006 American Control Conference, Minneapolis, MN, pp. 6057–6062 (2006)
16. Lasota, S.: Open maps as a bridge between algebraic observational equivalence and bisimilarity. In: Parisi-Presicce, F. (ed.) WADT 1997. LNCS, vol. 1376, pp. 285–299. Springer, Heidelberg (1998)
17. Malcolm, G.: Behavioural Equivalence, Bisimulation, and Minimal Realisation. In: Selected papers from the 11th Workshop on Specification of Abstract Data Types, vol. 1130(3), pp. 359–378. Springer, Heidelberg (1996)
18. Mealy, G.H.: A Method to Synthesizing Sequential Circuits. Bell Systems Technical Journal 34 (1955)
19. Milner, R.: A Calculus of Communicating Systems. Springer, Secaucus (1982)
20. Milner, R.: Communicating and mobile systems: the π-calculus. Cambridge University Press, New York (1999)
21. Nivela, M.P., Orejas, F.: Initial behavior semantics for algebraic specifications. In: Recent Trends in Data Type Specification, pp. 184–207. Springer, Heidelberg (1988)
22. Reichel, H.: Behavioural equivalence — a unifying concept for initial and final specification methods. In: Proc. of the 3rd. Hungarian Computer Science Conference, Akadémia kiadó, pp. 27–39 (1981)
23. Reichel, H.: Initial computability, algebraic specifications, and partial algebras. Oxford University Press Inc., New York (1987)
24. Rosu, G., Goguen, J.: Hidden Congruent Deduction. In: Selected Papers from Automated Deduction in Classical and Non-Classical Logics, pp. 251–266. Springer, Heidelberg (2000)
25. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249(1), 3–80 (2000)
26. Rutten, J.J.M.M.: A calculus of transition systems (towards universal coalgebra), Thechnical report, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands (1995)
27. Rutten, J.J.M.M.: Coalgebra, concurrency, and control, Thechnical report, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands (1999)
28. Sannella, D., Wirsing, M.: A Kernel Language for Algebraic Specification and Implementation - Extended Abstract. In: The Proc. of the 1983 International FCT-Conference on Fundamentals of Computation Theory, pp. 413–427. Springer, Heidelberg (1983)

Undecidability and Hierarchy Results for Parallel Communicating Finite Automata

Henning Bordihn[1], Martin Kutrib[2], and Andreas Malcher[2]

[1] Institut für Informatik, Universität Potsdam,
August-Bebel-Straße 89, 14482 Potsdam, Germany
`henning@cs.uni-potsdam.de`

[2] Institut für Informatik, Universität Giessen
Arndtstraße 2, 35392 Giessen, Germany
{`kutrib,malcher`}`@informatik.uni-giessen.de`

Abstract. Parallel communicating finite automata (PCFAs) are systems of several finite state automata which process a common input string in a parallel way and are able to communicate by sending their states upon request. We consider deterministic and nondeterministic variants and distinguish four working modes. It is known that these systems in the most general mode are as powerful as one-way multi-head finite automata. It is additionally known that the number of heads corresponds to the number of automata in PCFAs in a constructive way. Thus, undecidability results as well as results on the hierarchies induced by the number of heads carry over from multi-head finite automata to PCFAs in the most general mode. Here, we complement these undecidability and hierarchy results also for the remaining working modes. In particular, we show that classical decidability questions are not semi-decidable for any type of PCFAs under consideration. Moreover, it is proven that the number of automata in the system induces infinite hierarchies for deterministic and nondeterministic PCFAs in three working modes.

1 Introduction

In the beginning of computer science, all understanding of computing was based on the von Neumann architecture. Thus, processes have been viewed to be strictly sequential, and the theoretical models used for describing and analyzing them were mainly devices working in a sequential way. Since the late 1960-ies also parallel and distributed processes have gained more and more importance. Correspondingly, the theory of formal languages and automata developed a series of devices reflecting features such as parallelism, cooperation, distribution and concurrency.

One of the most intuitive approaches towards such devices might have been the development of suitable extensions of the most fundamental sequential machine model, namely that of finite state automata. Multi-head finite automata [18] are in some sense the simplest model of cooperating finite state automata, which are provided with a fixed number of reading heads. At any computational step each head processes some input symbol, and the global state is updated according to these pieces of information and the current state. Thus, we have some model with one finite state control, and the cooperation between the finite state control and the single components is the reading of the input and positioning the heads. This model has been generalized to multi-head two-way finite automata [10] and multi-head pushdown automata [8]. Further instances of systems of cooperating sequential automata appear in the literature in many facets, see, for example, [2,3,6,7,12]. Another approach is to consider systems of cooperating formal grammars [5].

Here, we will deal with parallel communicating finite automata (PCFAs) which were introduced in [17]. In this model, the input is read and processed in parallel by several finite automata. The communication between automata is defined in such a way that an automaton can request the current state from another automaton. Similarly to parallel communicating systems of formal grammars, the system can work in *returning* or *non-returning* mode. In the former case each automaton which sends its current state is reset to its initial state after this communication step. In the latter case the state of the sending automaton is not changed. We also distinguish between *centralized* systems where only one designated automaton, called *master*, can request information from other automata, and *non-centralized* systems where every automaton is allowed to communicate with others. Altogether we obtain four different working modes. One fundamental result shown in [17] is that nondeterministic (deterministic) non-centralized systems working in the non-returning mode are equally powerful as one-way multi-head nondeterministic (deterministic) finite automata. Recently, it has been shown in [4] that the returning and non-returning working modes coincide for nondeterministic non-centralized systems, and in [1] that this result is also valid in the deterministic case. Moreover, the authors proved in [1] that nondeterminism is strictly more powerful than determinism for all the four working modes, and that deterministic centralized returning systems are not weaker than deterministic centralized non-returning ones. Also some incomparability results with respect to certain classes of the Chomsky hierarchy and the family of Church-Rosser languages have been obtained there.

Due to the constructive proofs of the equivalences between one-way multi-head finite automata and non-centralized PCFAs, any decidability or undecidability result obtained for the one model is equally valid for the other one. Thus, it is known that classical decidability questions such as emptiness, universality, inclusion, equivalence, finiteness and infiniteness are undecidable for non-centralized PCFAs. But only little is known for the centralized variants. In [16] it has been shown that universality, inclusion and equivalence of centralized PCFAs are undecidable only for the nondeterministic and non-returning case and if at least five components are involved. The decidability status of the problems for nondeterministic centralized non-returning PCFAs with two, three or four components as well as the status of the emptiness and finiteness problems are formulated as open problems. Furthermore, no decidability results have been obtained so far for centralized PCFAs which are deterministic *or* returning, and are left as open problems in [16] as well.

In this paper, we answer most of these open problems: We prove that emptiness, universality, inclusion, equivalence, finiteness and infiniteness are *not* semi-decidable[1] for centralized PCFAs which are deterministic, nondeterministic, returning, or non-returning. In most cases, the results are obtained for systems with at least two components; only the non-semi-decidability of emptiness, finiteness and infiniteness for the centralized returning cases are based on systems with at least three components. The novelty in the proofs is that they rely on the concept of valid and invalid computations of one-way cellular automata (OCAs).

Another fundamental property of one-way multi-head finite automata is that the number of reading heads induces infinite strict hierarchies, both in the deterministic and nondeterministic cases. As it is known that the number of heads in one-way multi-head automata corresponds to the number of components needed in PCFAs, these hierarchy results carry over to the non-centralized variants. In this paper, we prove infinite strict inclusion hierarchies also for deterministic and nondeterministic centralized PCFAs working in the non-returning mode.

2 Preliminaries and Definitions

We denote the powerset of a set S by 2^S. The empty word is denoted by λ, the reversal of a word w by w^R, and for the length of w we write $|w|$. We use $\subseteq$ for *inclusions* and $\subset$ for *strict inclusions*.

Next we turn to the definition of the devices in question, which have been introduced in [17]. A parallel communicating finite automata system of degree k is a device of k finite automata working in parallel, synchronized according to a universal clock, on a common one-way read-only input tape. The k automata communicate by states, that is, when some automaton enters a query state q_i it is set to the current state of automaton A_i. Concerning the next state of the sender A_i, we distinguish two modes. In *non-returning* mode the sender remains in its current state, whereas in *returning* mode the sender is set to its initial state. Moreover, we distinguish whether all automata are allowed to request communications, or whether there is just one master allowed to request communications. The latter types are called *centralized*.

One of the fundamental results obtained in [17] is the characterization of the computational power of (unrestricted) parallel communicating finite automata systems by multi-head finite automata. Due to this relation, we present a formal definition of language acceptance that suits to the definition given in [19] for one-way multi-head finite automata. To this end, we provide tape inscriptions which are input words followed by an endmarker. Whenever the transition function of (at least) one of the single automata is undefined the whole systems halts. Whether the input is accepted or rejected depends on the states of the automata

[1] A decidability problem is said to be *semi-decidable* if the set of all instances for which the answer is "yes" is recursively enumerable.

having undefined transitions. The input is accepted if at least one of them is in an accepting state. Following [1], the formal definition is as follows.

A *nondeterministic parallel communicating finite automata system of degree k* (PCFA(k)) is a construct $\mathcal{A} = \langle \Sigma, A_1, A_2, \ldots, A_k, Q, \lhd \rangle$, where Σ is the set of *input symbols*, each $A_i = \langle S_i, \Sigma, \delta_i, s_{0,i}, F_i \rangle$, $1 \leq i \leq k$, is a *nondeterministic finite automaton* with state set S_i, initial state $s_{0,i} \in S_i$, set of accepting states $F_i \subseteq S_i$, and transition function $\delta_i : S_i \times (\Sigma \cup \{\lambda, \lhd\}) \to 2^{S_i}$, $Q = \{q_1, q_2, \ldots, q_k\}$ is the set of *query states*, and $\lhd \notin \Sigma$ is the *end-of-input symbol*.

The automata $A_1, A_2, \ldots, A_k$ are called *components* of the system $\mathcal{A}$. A *configuration* $(s_1, x_1, s_2, x_2, \ldots, s_k, x_k)$ of $\mathcal{A}$ represents the current states s_i as well as the still unread parts x_i of the tape inscription of all components $1 \leq i \leq k$. System $\mathcal{A}$ starts with all of its components scanning the first square of the tape in their initial states. For input word $w \in \Sigma^*$, the initial configuration is $(s_{0,1}, w\lhd, s_{0,2}, w\lhd, \ldots, s_{0,k}, w\lhd)$.

Basically, a computation of $\mathcal{A}$ is a sequence of configurations beginning with an initial configuration and ending with a halting configuration. Each step can consist of two phases. In a first phase, all components are in non-query states and perform an ordinary (non-communicating) step independently. The second phase is the communication phase during which components in query states receive the requested states as long as the sender is not in a query state itself. This process is repeated until all requests are resolved, if possible. If the requests are cyclic, no successor configuration exists. As mentioned above, we distinguish *non-returning* communication, that is, the sender remains in its current state, and *returning* communication, that is, the sender is reset to its initial state. For the first phase, we define the successor configuration relation $\vdash$ by $(s_1, a_1y_1, s_2, a_2y_2, \ldots, s_k, a_ky_k) \vdash (p_1, z_1, p_2, z_2, \ldots, p_k, z_k)$, if $Q \cap \{s_1, s_2, \ldots, s_k\} = \emptyset$, $a_i \in \Sigma \cup \{\lambda, \lhd\}$, $p_i \in \delta_i(s_i, a_i)$, and $z_i = \lhd$ for $a_i = \lhd$ and $z_i = y_i$ otherwise, $1 \leq i \leq k$. For non-returning communication in the second phase, we set $(s_1, x_1, s_2, x_2, \ldots, s_k, x_k) \vdash (p_1, x_1, p_2, x_2, \ldots, p_k, x_k)$, if, for all $1 \leq i \leq k$ such that $s_i = q_j$ and $s_j \notin Q$, we have $p_i = s_j$, and $p_r = s_r$ for all the other r, $1 \leq r \leq k$. Alternatively, for returning communication in the second phase, we set $(s_1, x_1, s_2, x_2, \ldots, s_k, x_k) \vdash (p_1, x_1, p_2, x_2, \ldots, p_k, x_k)$, if, for all $1 \leq i \leq k$ such that $s_i = q_j$ and $s_j \notin Q$, we have $p_i = s_j$, $p_j = s_{0,j}$, and $p_r = s_r$ for all the other r, $1 \leq r \leq k$.

A computation *halts* when the successor configuration is not defined for the current situation. In particular, this may happen when cyclic communication requests appear, or when the transition function of one component is not defined. (We regard the transition function as undefined whenever it maps to the empty set.) The language $L(\mathcal{A})$ accepted by a PCFA(k) $\mathcal{A}$ is precisely the set of words w such that there is some computation beginning with $w\lhd$ on the input tape and halting with at least one component having an undefined transition function and being in an accepting state. Let $\vdash^*$ designate the reflexive and transitive closure of the successor configuration relation $\vdash$ and set $L(\mathcal{A}) = \{\, w \in \Sigma^* \mid (s_{0,1}, w\lhd, s_{0,2}, w\lhd, \ldots, s_{0,k}, w\lhd) \vdash^* (p_1, a_1y_1, p_2, a_2y_2, \ldots, p_k, a_ky_k)$, such that $p_i \in F_i$ and $\delta_i(p_i, a_i)$ is undefined for some $1 \leq i \leq k \,\}$.

If all components A_i are deterministic finite automata, that is, for all $s \in S_i$ the transition function $\delta_i(s, a)$ maps to a set of at most one state and is undefined for all $a \in \Sigma$, whenever $\delta_i(s, \lambda)$ is defined, then the whole system is called *deterministic*, and we add the prefix D to denote it. The absence or presence of an R in the type of the system denotes whether it works in *non-returning* or *returning* mode. Finally, if there is just one component, say A_1, that is allowed to query for states, that is, $S_i \cap Q = \emptyset$, for $2 \le i \le k$, then the system is said to be *centralized*. In this case, we refer to A_1 as the *master component* and add a C to the notation of the type of the system. Whenever the degree is missing we mean systems of arbitrary degree. The *family of languages accepted* by devices of type X (with degree k) is denoted by $\mathscr{L}(X)$ ($\mathscr{L}(X(k))$). $\mathscr{L}(\text{NFA}(k))$ ($\mathscr{L}(\text{DFA}(k))$) are the families of languages accepted by nondeterministic (deterministic) multi-head finite automata with k heads.

3 Decidability Questions

The decidability status of the emptiness, universality, inclusion, equivalence, finiteness and infiniteness problems shall be settled. Due to the effective coincidence of the language classes $\mathscr{L}(\text{PCFA}(k))$ and $\mathscr{L}(\text{RPCFA}(k))$ with $\mathscr{L}(\text{NFA}(k))$ as well as of $\mathscr{L}(\text{DPCFA}(k))$ and $\mathscr{L}(\text{DRPCFA}(k))$ with $\mathscr{L}(\text{DFA}(k))$, for any $k \ge 1$, we know that the aforementioned questions are undecidable for nondeterministic and deterministic non-centralized PCFAs with at least two components.

Thus, it is left to investigate the decidability questions with respect to the centralized variants. From [16] we only know that universality, inclusion and equivalence are undecidable for CPCFA(k)s with $k \ge 5$. We complement these results by proving that all the decidability questions listed above are not semi-decidable for CPCFA(k)s and DCPCFA(k)s, for $k \ge 2$, and for RCPCFA(k)s and DRCPCFA(k)s, for $k \ge 3$. Furthermore, the non-semi-decidability of universality, inclusion and equivalence can be shown for RCPCFA(k)s with degree $k \ge 2$.

To prove our results we use the technique of valid computations which is described, e.g., in [9]. The goal is to prove undecidability results in a certain model. Now, the main idea is to effectively construct in the investigated model a set of strings which represent histories of accepting computations of a given Turing machine. Then, certain properties such as, e.g., emptiness or finiteness of the constructed set can be related to properties of the language accepted by the Turing machine. Since due to Rice's Theorem almost all decidability questions for Turing machines are undecidable, also undecidability results for the investigated model can be derived.

Since it is not clear yet in which way a DCPCFA or DRCPCFA could accept the set of valid computations of a Turing machine, we are considering here the valid computations of *one-way cellular automata* which is a parallel computational model (see, e.g., [13,14]). The same way as the valid computations of a Turing machine help to reduce decidability questions to decidability questions of Turing machines, we can reduce decidability questions for DCPCFAs and DRCPCFAs to those of one-way cellular automata. Since it is shown in [15] that for one-way cellular automata the questions of emptiness, universality, inclusion, equivalence, finiteness and infiniteness are not semi-decidable, we obtain the non-semi-decidability of all these questions for DCPCFAs or DRCPCFAs as well.

A one-way cellular automaton (OCA) consists of many identical deterministic finite automata, called *cells*, which are arranged in a line. The next state of a cell depends on the current state of the cell itself and the current state of its neighboring cell to the left. The transition rule is applied synchronously to each cell at the same time. The input is given to the OCA where each input symbol is written into one cell. The input is bounded by some boundary symbols. An OCA accepts an input when the rightmost cell enters an accepting state.

More formally, an OCA is a system $\mathcal{M} = \langle S, \#, T, \delta, F\rangle$, where $S \neq \emptyset$ is the finite set of cell states, $\# \notin S$ is the boundary state, $T \subseteq S$ is the input alphabet, $F \subseteq S$ is the set of accepting cell states and $\delta : (S \cup \{\#\}) \times S \to S$ is the local transition function. A configuration of an OCA at some time step $t \geq 0$ is a description of its global state. Formally, for any input w, it is a mapping $c_{t,w} : \{1, 2, \ldots, n\} \to S$ for $n \geq 1$. If there is no danger of confusion, we will omit the subscript w in that notation. The initial configuration at time 0 on the input $w = x_1x_2 \ldots x_n$ is defined by $c_{0,w}(i) = x_i$, $1 \leq i \leq n$. During a computation the OCA steps through a sequence of configurations whereby successor configurations are computed according to the global transition function Δ: Let c_t, $t \geq 0$, be a configuration, then its successor configuration is defined as follows:

$$\begin{aligned} c_{t+1} = \Delta(c_t) \iff & \\ & c_{t+1}(1) = \delta(\#, c_t(1)) \\ & c_{t+1}(i) = \delta(c_t(i-1), c_t(i)), i \in \{2, \ldots, n\} \end{aligned}$$

Thus, Δ is induced by δ. An input string w is accepted by an OCA if at some time step i during its computation the rightmost cell enters a state from the set of accepting states $F \subseteq S$. Without loss of generality and for technical reasons, one can assume that any accepting computation has at least three steps. The language accepted by some OCA $\mathcal{M}$ is defined as the set of all accepted input strings and is denoted by $L(\mathcal{M})$.

Let $\mathcal{M} = \langle S, \#, T, \delta, F\rangle$ be an OCA. The successor configuration c_{t+1} of c_t on some input w of length n is computed in all cells in parallel by applying the transition rule δ to each cell and its left neighbor simultaneously. We now want to write down the computation of the successor configuration in a sequential way. Thus, we are computing c_{t+1} cell by cell from left to right instead of computing c_{t+1} in one step from c_t. That is, we are concerned with subconfigurations of the form $c_{t+1}(1) \ldots c_{t+1}(i)c_t(i+1) \ldots c_t(n)$. For technical reasons, in $c_{t+1}(i)$ we have to store both the successor state, which is assumed in time step $t+1$ by cell i, and additionally its former state. In this way, the computation of the successor configuration of $\mathcal{M}$ can be written as a sequence of n subconfigurations. Thus, a configuration c_{t+1} of $\mathcal{M}$ can be represented by $w^{(t+1)} = w_1^{(t+1)} \ldots w_n^{(t+1)}$, where, for $1 \leq i \leq n$, $w_i^{(t+1)}$ is contained in $\#S^*(S \times S)S^*$ with $w_i^{(t+1)} = \#c_{t+1}(1) \ldots c_{t+1}(i-1)(c_{t+1}(i), c_t(i))c_t(i+1) \ldots c_t(n)$.

The set of valid computations VALC($\mathcal{M}$) is now defined to be the set of words of the form $w^{(0)}w^{(1)}\cdots w^{(m)}$, where $m \geq 3$, $w^{(t)} \in (\#S^*(S \times S)S^*)^+$ are configurations of $\mathcal{M}$, $1 \leq t \leq m$, $w^{(0)}$ is an initial configuration carrying the input having the form $\#(T')^+$, where T' is a primed copy of the input alphabet T with $T' \cap S = \emptyset$, $w^{(m)}$ is an accepting configuration, that is, $w^{(m)}$ is contained in $(\#S^*(S \times S)S^*)^*\#S^*(F \times S)$, and $w^{(t+1)}$ is the successor configuration of $w^{(t)}$, for $0 \leq t \leq m-1$.

For constructing the DCPCFAs and DRCPCFAs which are accepting the set VALC($\mathcal{M}$), we need an additional technical transformation of the input alphabet. Let $S' = S \cup T'$ and $A = \{\#\} \cup S' \cup S'^2$ be the alphabet over which VALC($\mathcal{M}$) is defined. We consider the mapping $f : A^+ \to (A \times A)^+$ which is defined for words of length at least two by $f(x_1x_2\ldots x_n) = [x_1,x_2][x_2,x_3]\ldots[x_{n-1},x_n]$. From now on we consider VALC($\mathcal{M}$) $\subseteq (A \times A)^+$ to be the set of valid computations to which f has been applied. The set of *invalid computations* INVALC($\mathcal{M}$) is then the complement of VALC($\mathcal{M}$) with respect to the alphabet $A \times A$.

Lemma 1. *Let $\mathcal{M}$ be an OCA. Then the sets* VALC($\mathcal{M}$) *and* INVALC($\mathcal{M}$) *belong to* $\mathscr{L}$(DCPCFA(2)) *as well as to* $\mathscr{L}$(DRCPCFA(3)).

Proof. (Sketch) We first describe the construction for accepting the set VALC($\mathcal{M}$) in the centralized and returning mode with three components. Let us call them the master, component$_1$ and component$_2$. The latter component reads one input symbol in every time step and is used as a DFA which checks the correct format of the input string. In particular, it has to be checked whether for every two subsequent input symbols $[x_1,x_2]$ and $[x_3,x_4]$ holds that $x_2 = x_3$ and that no primed symbols appear to the right of the first 'unprimed' symbol. Obviously, this can be realized by some DFA. It is additionally checked whether the input starts with an initial configuration and ends with an accepting configuration. Finally, when component$_2$ reaches the endmarker, it stores in its state whether or not the input format was correct.

The master and component$_1$ are used to verify that from every subconfiguration the correct successor subconfiguration is computed. Due to the checking of component$_2$ we may assume that the input is correctly formatted. The principal behavior of component$_1$ is to read one input symbol, to store it in its state, and then to wait as long as it is queried by the master. After the master requested communication, component$_1$ is reset to its initial state, reads the next input symbol, stores it in its state, and waits to be queried again. In the beginning, master and component$_1$ are at the first symbol of $f(w^{(0)})$. Here, the master performs λ-transitions and queries component$_1$ until it obtains the first input symbol from the set $\{\#\} \times S$. Then, the master knows that component$_1$ is at the beginning of the next subconfiguration and the master starts to check the correct computation of the successor subconfiguration. This is realized as follows. The master queries component$_1$ and gets the information about the current symbol of the next subconfiguration. Subsequently, component$_1$ returns to its initial state. In the next time step, component$_1$ reads the next input symbol and the master checks with the help of the next input symbol from the first subconfiguration whether the information from component$_1$ is a part of the correct successor subconfiguration.

If the checking was successful, then the master queries again component$_1$ and tries to check the next part of the successor subconfiguration. This behavior is iterated as long as the checks between subconfiguration and successor subconfiguration are successful. Otherwise, some error has occurred and the master assumes a certain state *rej'* which forces the master to move to the right end and to assume a non-accepting state *rej*.[2] Since all subconfigurations have the same length, which is due to the fact that we consider valid computations of OCAs, we observe that after having checked that $w_{i+1}^{(t+1)}$ is the correct successor of $w_i^{(t+1)}$, the master is at the beginning of $w_{i+1}^{(t+1)}$ and component$_1$ is at the beginning of $w_{i+2}^{(t+1)}$. Thus, the next check can be started subsequently. The successor relation between $w_n^{(t+1)}$ and $w_1^{(t+2)}$ is checked in the same manner.

If an accepting configuration has been checked correctly, the master assumes some state *acc'*, moves to the right end and finally queries component$_2$. If the input was correctly formatted, an accepting state is assumed and the input is accepted. Otherwise, the input is rejected.

To accept the set INVALC($\mathcal{M}$) we use the identical construction, but we define *rej* to be an accepting state as well as the states of component$_2$ which say that the input was not correctly formatted.

In the centralized non-returning mode, the construction is nearly identical. We use again the alphabet T' to bring component$_1$ in position at the beginning of the second subconfiguration. Now, the construction to check the correct computation of the successor subconfiguration is the same as in the returning mode. Since we are here in the non-returning mode, component$_1$ is not reset to its initial state after being queried. Thus, component$_1$ can be additionally used to check the correct formatting of the input. Acceptance and rejection of the input is defined analogously. Altogether, we need only two components in contrast to the returning mode. The considerations to accept INVALC($\mathcal{M}$) are similar. □

Theorem 2. *Emptiness, finiteness, infiniteness, universality, inclusion, equivalence, regularity and context-freeness are not semi-decidable for* DRCPCFA(k)*s and* RCPCFA(k)*s of degree* $k \geq 3$ *and* DCPCFA(k))*s and* CPCFA(k)*s of degree* $k \geq 2$.

Proof. Let $\mathcal{M}$ be an OCA. By Lemma 1 we can construct DCPCFA(2))s as well as DRCPCFA(3)s accepting VALC($\mathcal{M}$) and INVALC($\mathcal{M}$), respectively. It is easy to observe that VALC($\mathcal{M}$) $= \emptyset$ if and only if $L(\mathcal{M}) = \emptyset$ as well as VALC($\mathcal{M}$) is finite if and only if $L(\mathcal{M})$ is finite. Since emptiness, finiteness and infiniteness are not semi-decidable for OCAs [15], the questions are not semi-decidable for DCPCFA(2))s and DRCPCFA(3)s as well.

Since INVALC($\mathcal{M}$) $= (A \times A)^*$ if and only if VALC($\mathcal{M}$) $= \emptyset$, we obtain that universality is not semi-decidable for DCPCFA(2))s and DRCPCFA(3)s as well. This implies immediately that also the questions of inclusion and equivalence are not semi-decidable.

[2] The moving to the right end of the input in the state *rej'* is part of the construction in order to simplify the proof with respect to the set INVALC($\mathcal{M}$). It is dispensable here.

By standard pumping arguments, it can be shown that $\mathrm{VALC}(\mathcal{M})$ is regular (context-free) if and only if $L(\mathcal{M}) = \emptyset$. This implies that the questions of regularity and context-freeness are not semi-decidable.

The results for DCPCFAs and DRCPCFAs of larger degrees as well as for their nondeterministic variants follow immediately. □

Similar to the proof of Theorem 2, the following lemma helps to obtain non-semi-decidability results.

Lemma 3. *Let $\mathcal{M}$ be an OCA. Then the set* $\mathrm{INVALC}(\mathcal{M})$ *belongs to the language class* $\mathscr{L}(\mathrm{RCPCFA}(2))$.

Theorem 4. *Universality, inclusion, and equivalence are not semi-decidable for* $\mathrm{RCPCFA}(k)$*s of degree* $k \geq 2$.

4 Degree Hierarchy

In this section we investigate the question for which variants of PCFAs $k+1$ components give rise to more computational power than k do. We shall find infinite strict hierarchies induced by the degree of parallel communicating finite automata in six cases, namely for PCFAs, RPCFAs, CPCFAs and their deterministic variants.

Since $\mathscr{L}(\mathrm{DRPCFA}(k)) = \mathscr{L}(\mathrm{DPCFA}(k))$ ($\mathscr{L}(\mathrm{RPCFA}(k)) = \mathscr{L}(\mathrm{PCFA}(k))$) coincides, for $k \geq 1$, with the family of languages accepted by deterministic (nondeterministic) one-way k-head finite automata, we inherit the degree hierarchies immediately from the head hierarchies shown in [19]. In order to prove a strict hierarchy dependent on the number of components for the non-returning centralized types of devices, we show that there are languages accepted by the weaker devices of degree $k+1$, that is by some $\mathrm{DCPCFA}(k+1)$, but cannot be accepted by the stronger device of degree k, that is by any $\mathrm{CPCFA}(k)$. To this end, for $k \geq 1$ we define the witness language

$$L_k = \{\, v_1\$v_2\$\cdots\$v_k\$^m u v_k\$\cdots\$v_2\$v_1 \mid m \geq 1,\\ u \in \{a,b\}^*, v_i \in \{a,b\}^+,\ \text{for } 1 \leq i \leq k \,\}.$$

First we construct a deterministic non-returning centralized system of degree $k+1$ accepting L_k.

Lemma 5. *Let $k \geq 1$ be a constant. Then the language L_k belongs to the family* $\mathscr{L}(\mathrm{DCPCFA}(k+1))$.

Proof. (Sketch) For easier writing we divide the input into two halves, namely the halves to the left and to the right of the block $\$^m$, and write left v_i and right v_i for the subwords of the first and second half. The principal idea of the construction is that the master and each of the k non-master components is used to compare one pair of matching subwords. To this end, the non-master components move to their right subwords, while the master waits until the first component has reached the right v_1. Then the master queries the first component and receives information about the current symbol of the right v_1, which is matched against the current symbol read be the master itself. After comparing the subwords v_i the master continues by comparing the next subword v_{i+1} by querying the component $i+1$. The crucial point is to let the components move in such a way that they are at the beginning of their right subwords exactly when the master starts to compare that subword. We start to construct the master and the first non-master component, and subsequently the remaining components. Then we explain how these constructions fit together in order to form the desired DCPCFA$(k+1)$.

The first non-master component moves to the right while it checks whether all subwords v_i are non-empty. When it passes the $(2k-1)$st border between subwords it starts to represent the symbol read by its state. The component counts the borders between subwords passed through up to $2k-1$. Whenever it moves on the first symbol of a subword it is in some unprimed state. All other symbols a and b are read in a primed state except for the symbols of the right subword v_1. So, it is ensured that every subword has at least one symbol. While reading the right v_1 the first component stores the symbol read in its state. Finally, it loops on the endmarker.

Next we turn to the master. The master waits until the first component has reached the right v_1, that is, has entered a state $s_{x,1}$ for the first time. Then it continues to query the first component and matches the received information about the current symbol of the right v_1 against the symbol currently read by itself. If the endmarker is matched against the \$ between the left v_1 and the left v_2, the comparison of the v_1 was successful and the master starts to compare the next subwords. Similarly, the other subwords are compared by querying the components 2 to k. Finally, if the comparison of the v_k was successful, the master enters the accepting state and blocks the computation accepting.

Next, we construct the remaining non-master components. Basically, component $2 \leq i \leq k$ passes through the left subwords $v_1, v_2, \ldots, v_i$, whereby it waits one time step on every symbol of $v_1\$$ and $v_i\$$, and two time steps on every symbol of $v_2\$v_3\$\cdots\$v_{i-1}\$$. Then it starts to store every symbol read in its state. Finally, it loops on the endmarker.

Finally, we give evidence that the master and the components form the desired DCPCFA$(k+1)$. When the comparison of the subwords v_1 and v_1' starts, the first component has ensured that it has passed through $2k-1$ non-empty subwords before. If the comparison is successful, the structure of the input is correct and, moreover, v_1 matches v_1'. So, the input is of the form $v_1\$v_2\$\cdots\$v_k\$^m v_k'\$\cdots\$v_2'\$v_1$.

The second component is delayed by $|v_1\$v_2\$|$ time steps against the first one. So, when the comparison of the v_1 is finished, it reads the first symbol of the input suffix $\hat{v}\$v_1$, where $\hat{v} \in \{a,b,\$\}^*$ and $|\hat{v}| = |v_2|$, while the master starts to read v_2. After a successful comparison of $\hat{v}$ and v_2 it has been verified that the input is of the form $v_1\$v_2\$\cdots\$v_k\$^m v_k'\$\cdots\$uv_2\$v_1$, where $u \in \{a,b\}^*$.

Proceeding inductively, the ith component is delayed by $|v_{i-1}\$v_i\$|$ time steps against the $(i-1)$st one. So, it reads the first symbol of the input suffix $\hat{v}\$uv_{i-1}\$\cdots\$v_1$, where $\hat{v} \in \{a,b,\$\}^*$ and $|\hat{v}\$u| = |v_i|+1$, while the master starts to read v_i. After a successful comparison it has been verified that the input is of the form $v_1\$\cdots\$v_{i-1}\$v_i\$\cdots\$v_k\$^m v'_k\$\cdots\$uv_i\$v_{i-1}\$\cdots\$v_1$. Setting $i=k$ verifies that the input accepted belongs to L_k. □

Finally, by proving that L_k cannot be accepted by a CPCFA(k) we obtain the infinite strict degree hierarchies. The proof of the following theorem is omitted due to space restrictions.

Theorem 6. *Let* $X \in \{\text{PCFA}, \text{DPCFA}, \text{RPCFA}, \text{DRPCFA}, \text{CPCFA}, \text{DCPCFA}\}$ *and* $k \geq 1$ *be a constant. Then the family* $\mathscr{L}(X(k))$ *is properly included in* $\mathscr{L}(X(k+1))$.

5 Concluding Remarks

In this paper, hierarchy and undecidability results for parallel communicating finite automata systems are proved. It is left open whether the number of components induces infinite hierarchies also for centralized systems working in the returning mode. Another open question is whether there are reasonable subclasses of PCFAs for which some problems treated here become decidable. For the stateless variant of the closely related multihead automata [11], some decidability results have been obtained.

References

1. Bordihn, H., Kutrib, M., Malcher, A.: On the computational capacity of parallel communicating finite automata. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 146–157. Springer, Heidelberg (2008)
2. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30, 323–342 (1983)
3. Buda, A.: Multiprocessor automata. Inform. Process. Lett. 25, 257–261 (1987)
4. Choudhary, A., Krithivasan, K., Mitrana, V.: Returning and non-returning parallel communicating finite automata are equivalent. RAIRO Inform. Théor. 41, 137–145 (2007)
5. Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G.: Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Gordon and Breach, Yverdon (1984)
6. Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V., Vaszil, G.: Parallel communicating pushdown automata systems. Int. J. Found. Comput. Sci. 11, 633–650 (2000)
7. Ďuriš, P., Jurdziński, T., Kutyłowski, M., Loryś, K.: Power of cooperation and multihead finite systems. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 896–907. Springer, Heidelberg (1998)
8. Harrison, M.A., Ibarra, O.H.: Multi-tape and multi-head pushdown automata. Inform. Control 13, 433–470 (1968)
9. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Language, and Computation. Addison-Wesley, Reading (1979)

10. Ibarra, O.H.: On two-way multihead automata. J. Comput. System Sci. 7, 28–36 (1973)
11. Ibarra, O.H., Karhumäki, J., Okhotin, A.: On stateless multihead automata: Hierarchies and the emptiness problem. Theoret. Comp. Sci. 411, 581–593 (2010)
12. Klemm, R.: Systems of communicating finite state machines as a distributed alternative to finite state machines. Phd thesis, Pennsylvania State University (1996)
13. Kutrib, M.: Cellular automata – a computational point of view. In: New Developments in Formal Languages and Applications, pp. 183–227. Springer, Heidelberg (2008)
14. Kutrib, M.: Cellular automata and language theory. In: Encyclopedia of Complexity and System Science, pp. 800–823. Springer, Heidelberg (2009)
15. Malcher, A.: Descriptional complexity of cellular automata and decidability questions. J. Autom., Lang. Comb. 7, 549–560 (2002)
16. Martín-Vide, C., Mitrana, V.: Some undecidable problems for parallel communicating finite automata systems. Inform. Process. Lett. 77, 239–245 (2001)
17. Martín-Vide, C., Mateescu, A., Mitrana, V.: Parallel finite automata systems communicating by states. Int. J. Found. Comput. Sci. 13, 733–749 (2002)
18. Rosenberg, A.L.: On multi-head finite automata. IBM J. Res. Dev. 10, 388–394 (1966)
19. Yao, A.C., Rivest, R.L.: $k+1$ heads are better than k. J. ACM 25, 337–340 (1978)

Inclusion Problems for Patterns with a Bounded Number of Variables

Joachim Bremer and Dominik D. Freydenberger*

Institut für Informatik, Goethe Universität, Frankfurt am Main, Germany
bremer@cs.uni-frankfurt.de, freydenberger@em.uni-frankfurt.de

Abstract. We study the inclusion problems for pattern languages that are generated by patterns with a bounded number of variables. This continues the work by Freydenberger and Reidenbach (Information and Computation 208 (2010)) by showing that restricting the inclusion problem to significantly more restricted classes of patterns preserves undecidability, at least for comparatively large bounds. For smaller bounds, we prove the existence of classes of patterns with complicated inclusion relations, and an open inclusion problem, that are related to the Collatz Conjecture. In addition to this, we give the first proof of the undecidability of the inclusion problem for NE-pattern languages that, in contrast to previous proofs, does not rely on the inclusion problem for E-pattern languages, and proves the undecidability of the inclusion problem for NE-pattern languages over binary and ternary alphabets.

1 Introduction

Patterns – finite strings that consist of *variables* and *terminals* – are compact and natural devices for the definition of formal languages. A pattern generates a word by a *substitution* of the variables with arbitrary strings of terminals from a fixed alphabet Σ (where all occurrences of a variable in the pattern must be replaced with the same word), and its language is the set of all words that can be obtained under substitutions. In a more formal manner, the language of a pattern can be understood as the set of all images under *terminal-preserving* morphisms; i. e., morphisms that map variables to terminal strings, and each terminal to itself. For example, the pattern $\alpha = x_1 x_1\, \mathtt{a}\,\mathtt{b}\, x_2$ (where x_1 and x_2 are variables, and $\mathtt{a}$ and $\mathtt{b}$ are terminals) generates the language of all words that have a prefix that consists of a square, followed by the word $\mathtt{a}\,\mathtt{b}$.

The study of patterns in strings goes back to Thue [20] and is a central topic of combinatorics on words (cf. the survey by Choffrut and Karhumäki [3]), while the investigation of pattern languages was initiated by Angluin [1]. Angluin's definition of pattern languages permits only the use of *nonerasing* substitutions (hence, this class of pattern languages is called *NE-pattern languages*). Later, Shinohara [19] introduced *E-pattern languages* (E for 'erasing' or 'extended'), were erasing substitutions are permitted.

* Corresponding author.

This small difference in the definitions leads to immense differences in the properties of these two classes. For example, while the equivalence problem for NE-pattern languages is trivially decidable, the equivalence problem for E-pattern languages is a hard open problem. Although both classes were first introduced in the context of *inductive inference* (which deals with the problem of learning patterns for given sets of strings, for a survey see Ng and Shinohara [15]), they have been widely studied in Formal Language Theory (cf. the surveys by Mitrana [13], Salomaa [18]). Due to their compact definition, patterns or their languages occur in numerous prominent areas of computer science and discrete mathematics, including *unavoidable patterns* (cf. Jiang et al. [8]), *practical regular expressions* (cf. Câmpeanu et al. [2]), or *word equations* and the *positive theory of concatenation* (cf. Choffrut and Karhumäki [3]).

One of the most notable results on pattern languages is the proof of the undecidability of the inclusion problem by Jiang et al. [9], a problem that was open for a long time and is of vital importance for the inductive inference of pattern languages. Unfortunately, this proof heavily depends on the availability of an unbounded number of terminals, which might be considered impractical, as pattern languages are mostly used in settings with fixed (or at least bounded) alphabets. But as shown by Freydenberger and Reidenbach [6], undecidability holds even if the terminal alphabet is bounded. As the proof by Jiang et al. and its modification by Freydenberger and Reidenbach require the number of variables of the involved patterns to be unbounded, we consider it a natural question whether the inclusion problems remain undecidable even if bounds are imposed on the number of variables in the pattern; especially as bounding the number of variables changes the complexity of the membership problem from NP-complete to P (cf. Ibarra et al. [7]). Similar restrictions have been studied in the theory of concatenation (cf. Durnev [4]).

Apart from potential uses in inductive inference or other areas, and the search for an approach that could provide the leverage needed to solve the equivalence problem for E-pattern languages, our main motivation for deeper research into the inclusion problems is the question how strongly patterns and their languages are connected. All known cases of (non-trivial) decidability of the inclusion problem for various classes of patterns rely on the fact that for these classes, inclusion is characterized by the existence of a terminal-preserving morphism mapping one pattern to the other. This is a purely syntactical condition that, although NP-complete (cf. Ehrenfeucht and Rozenberg [5]), can be straightforwardly verified. Finding cases of inclusion that are not covered by this condition, but still decidable, could uncover (or rule out) previously unknown phenomena, and be of immediate use for related areas of research.

Our results can be summarized as follows: We show that the inclusion problems for E- and NE-patterns with a bounded (but large) number of variables are indeed undecidable. For smaller bounds, we prove the existence of classes of patterns with complicated inclusion relations, and an open inclusion problem. Some of these inclusions can simulate iterations of the Collatz function, while others could (in principle) be used to settle an important part of the famous

Collatz Conjecture. In contrast to the aforementioned previous proofs, our proof of the undecidability of the inclusion problem for NE-pattern languages is not obtained through a reduction of the inclusion problem for E-pattern languages. Apart from the technical innovation, this allows to prove the undecidability of the inclusion problem for NE-pattern languages over binary and ternary alphabets, which was left open by Freydenberger and Reidenbach.

2 Preliminaries

2.1 Basic Definitions and Pattern Languages

Let $\mathbb{N}_1 := \{1, 2, 3, \ldots\}$ and $\mathbb{N}_0 := \mathbb{N}_1 \cup \{0\}$. The function div denotes the integer division, and mod its remainder. The symbols $\subseteq$, $\subset$, $\supseteq$ and $\supset$ refer to subset, proper subset, superset and proper superset relation, respectively. The symbol $\setminus$ denotes the set difference, and $\emptyset$ the empty set.

For an arbitrary alphabet A, a *string* (over A) is a finite sequence of symbols from A, and λ stands for the *empty string*. The symbol A^+ denotes the set of all nonempty strings over A, and $A^* := A^+ \cup \{\lambda\}$. 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 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 $w = u_1 v u_2$. If $u_1 = \lambda$ (or $u_2 = \lambda$), then v is a *prefix* of w (or a *suffix*, respectively).

For any alphabet A, a *language* L (over A) is a set of strings over A, i. e. $L \subseteq A^*$. A language L is *empty* if $L = \emptyset$; otherwise, it is *nonempty.*

The notation $|K|$ stands for the size of a set K or the length of a string K; the term $|w|_a$ refers to the number of occurrences of the symbol a in the string w. For any $w \in \Sigma^*$ and any $n \in \mathbb{N}_0$, w^n denotes the *n-fold concatenation of w*, with $w^0 := \lambda$. Furthermore, we use $\cdot$ and the regular operations $*$ and $+$ on sets and strings in the usual way.

For any alphabets A, B, a *morphism* is a function $h : A^* \to B^*$ that satisfies $h(vw) = h(v)h(w)$ for all $v, w \in A^*$. A morphism $h : A^* \to B^*$ is said to be *nonerasing* if $h(a) \neq \lambda$ for all $a \in A$. For any string $w \in C^*$, where $C \subseteq A$ and $|w|_a \geq 1$ for every $a \in C$, the morphism $h : A^* \to B^*$ is called a *renaming (of w)* if $h : C^* \to B^*$ is injective and $|h(a)| = 1$ for every $a \in C$.

Let Σ be a (finite or infinite) alphabet of so-called *terminals* and X an infinite set of *variables* with $\Sigma \cap X = \emptyset$. We normally assume $\{\mathtt{a}, \mathtt{b}, \ldots\} \subseteq \Sigma$ and $\{x_1, x_2, x_3 \ldots\} \subseteq X$. A *pattern* is a string over $\Sigma \cup X$, a *terminal-free pattern* is a string over X and a *terminal-string* is a string over Σ. For any pattern α, we refer to the set of variables in α as $\operatorname{var}(\alpha)$. The set of all patterns over $\Sigma \cup X$ is denoted by Pat_Σ; the set of all terminal-free patterns is denoted by $\mathrm{Pat}_{\mathrm{tf}}$. For every $n \geq 0$, let $\mathrm{Pat}_{n,\Sigma}$ denote the set of all patterns over Σ that contain at most n variables; that is, $\mathrm{Pat}_{n,\Sigma} := \{\alpha \in \mathrm{Pat}_\Sigma \mid |\operatorname{var}(\alpha)| \leq n\}$.

A morphism $\sigma : (\Sigma \cup X)^* \to (\Sigma \cup X)^*$ is called *terminal-preserving* if $\sigma(a) = a$ for every $a \in \Sigma$. A terminal-preserving morphism $\sigma : (\Sigma \cup X)^* \to \Sigma^*$ is called a *substitution.* The *E-pattern language* $L_{\mathrm{E},\Sigma}(\alpha)$ of α is given by

$$L_{\mathrm{E},\Sigma}(\alpha) := \{\sigma(\alpha) \mid \sigma : (\Sigma \cup X)^* \to \Sigma^* \text{ is a substitution}\},$$

and the *NE-pattern language* $L_{\mathrm{NE},\Sigma}(\alpha)$ of a pattern $\alpha \in \mathrm{Pat}_\Sigma$ is given by

$$L_{\mathrm{NE},\Sigma}(\alpha) := \{\sigma(\alpha) \mid \sigma : (\Sigma \cup X)^* \to \Sigma^* \text{ is a nonerasing substitution}\}.$$

If the intended meaning is clear, we write $L(\alpha)$ instead of $L_{\mathrm{E},\Sigma}(\alpha)$ or $L_{\mathrm{NE},\Sigma}(\alpha)$ for any $\alpha \in \mathrm{Pat}_\Sigma$. Furthermore, let ePAT_Σ denote the class of all E-pattern languages over Σ, and nePAT_Σ the class of all NE-pattern languages over Σ. Likewise, we define $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$ as the class of all $L_{\mathrm{E},\Sigma}(\alpha)$ with $\alpha \in \mathrm{Pat}_{\mathrm{tf}}$, and, for any $n \geq 0$, $\mathrm{ePAT}_{n,\Sigma}$ as the class of all $L_{\mathrm{E},\Sigma}(\alpha)$ with $\alpha \in \mathrm{Pat}_{n,\Sigma}$. The classes $\mathrm{nePAT}_{\mathrm{tf},\Sigma}$ and $\mathrm{nePAT}_{n,\Sigma}$ are defined accordingly. Let P_1, P_2 be two classes of patterns, and $\mathrm{PAT}_1, \mathrm{PAT}_2$ be the corresponding classes of pattern languages (either the class of all E-pattern languages or the class of all NE-pattern languages over some alphabet Σ that are generated by patterns from P_1 or P_2). We say that the *inclusion problem for* PAT_1 *in* PAT_2 *is decidable* if there exists a total computable function χ such that, for every pair of patterns $\alpha \in P_1$ and $\beta \in P_2$, χ decides on whether or not $L(\alpha) \subseteq L(\beta)$. If no such function exists, this inclusion problem is *undecidable*. If both classes of pattern languages are the same class $\mathrm{PAT}_{\star,\Sigma}$, we simple refer to the *inclusion problem of* $\mathrm{PAT}_{\star,\Sigma}$.

The concepts contained in the following two Sections are a vital part of our considerations.

2.2 A Universal Turing Machine

Let U be the universal Turing machine $U_{15,2}$ with 2 symbols and 15 states described by Neary and Woods [14]. This machine has the state set $Q = \{q_1, \ldots, q_{15}\}$ and operates on the tape alphabet $\Gamma = \{0, 1\}$ (where 0 is the blank symbol). In order to discuss configurations of U, we adopt the following conventions. The tape content of any configuration of U is characterized by the two infinite sequences $t_L = (t_{L,n})_{n\geq 0}$ and $t_R = (t_{R,n})_{n\geq 0}$ over Γ. Here, t_L describes the content of what we shall call the *left side of the tape*, the infinite word that starts at the position of the machine's head and extends to the left. Likewise, t_R describes the *right side of the tape*, the infinite word that starts immediately to the right of the head and extends to the right.

Next, we define the function $\mathrm{e} : \Gamma \to \mathbb{N}_0$ as $\mathrm{e}(0){:=}0$ and $\mathrm{e}(1){:=}1$, and extend this to an encoding of infinite sequences $t = (t_n)_{n\geq 0}$ over Γ by $\mathrm{e}(t){:=} \sum_{i=0}^{\infty} 2^i \,\mathrm{e}(t_i)$. As we consider only configurations where all but finitely many cells of the tape consist of the blank symbol 0 (which is encoded as 0), $\mathrm{e}(t)$ is always finite and well-defined. Note that for every side t of the tape, $\mathrm{e}(t) \bmod 2$ returns the encoding of the symbol that is closest to the head (the symbol under the head for t_L, and the symbol to the right of the head for t_R). Furthermore, each side can be lengthened or shortened by multiplying or dividing (respectively) its encoding $\mathrm{e}(t)$ by 2. The encodings $\mathrm{enc}_{\mathrm{E}}$ and $\mathrm{enc}_{\mathrm{NE}}$ of configurations of U are defined by

$$\mathrm{enc}_{\mathrm{E}}(q_i, t_L, t_R){:=}0\, 0^{\mathrm{e}(t_R)}\#0\, 0^{\mathrm{e}(t_L)}\#0^i,$$
$$\mathrm{enc}_{\mathrm{NE}}(q_i, t_L, t_R){:=}0^7\, 0^{\mathrm{e}(t_R)}\#0^7\, 0^{\mathrm{e}(t_L)}\#0^{i+6},$$

for every configuration (q_i, t_L, t_R). Note that both functions are almost identical; the only difference is that $\mathrm{enc}_{\mathrm{NE}}$ adds six additional occurrences of 0 to each of the three continuous blocks of 0.

We extend each of these encodings to an encoding of finite sequences of configurations $C = (C_i)_{i=1}^n$ by $\mathrm{enc}(C){:=}\#\#\,\mathrm{enc}(C_1)\#\#\ldots\#\#\,\mathrm{enc}(C_n)\#\#$ for $\mathrm{enc} = \mathrm{enc}_{\mathrm{E}}$ or $\mathrm{enc} = \mathrm{enc}_{\mathrm{NE}}$. Let I be any configuration of U. A *valid computation from* I is a finite sequence $C = (C_i)_{i=1}^n$ (with $n \geq 2$) of configurations of U such that $C_1 = I$, C_n is a halting configuration, and C_{i+1} is a valid successor configuration of C_i for every i with $1 \leq i < n$. We adopt the convention that any possible configuration where both tape sides have a finite value under e is a valid successor configuration of a halting configuration. This extended definition of succession does not change the acceptance behavior of U. Finally, let

$$\mathrm{VALC}_{\mathrm{E}}(I){:=}\{\mathrm{enc}_{\mathrm{E}}(C) \mid C \text{ is a valid computation from } I\},$$
$$\mathrm{VALC}_{\mathrm{NE}}(I){:=}\{\mathrm{enc}_{\mathrm{NE}}(C) \mid C \text{ is a valid computation from } I\}.$$

Each of the two sets is nonempty if and only if U accepts the input of the initial configuration I, and can thus be used to decide the halting problem of U. As U is universal, there can be no recursive function that, on input I, decides whether $\mathrm{VALC}_{\mathrm{E}}(I)$ is empty or not (the same holds for $\mathrm{VALC}_{\mathrm{NE}}(I)$).

2.3 Collatz Iterations

The Collatz function $\mathcal{C} : \mathbb{N}_1 \to \mathbb{N}_1$ is defined by $\mathcal{C}(n){:=}\frac{1}{2}n$ if n is even, and $\mathcal{C}(n) := 3n + 1$ if n is odd. For any $i \geq 0$ and any $n \geq 1$, let $\mathcal{C}^0(n){:=}n$ and $\mathcal{C}^{i+1}(n){:=}\mathcal{C}(\mathcal{C}^i(n))$. A number n *leads* $\mathcal{C}$ *into a cycle* if there are i, j with $1 \leq i < j$ and $\mathcal{C}^i(n) = \mathcal{C}^j(n)$. The cycle is *non-trivial* if $\mathcal{C}^k(n) \neq 1$ for every $k \geq 0$; otherwise, it is the *trivial cycle*.

The Collatz Conjecture states that every natural number leads $\mathcal{C}$ into the trivial cycle $4, 2, 1$. Regardless of the considerable effort spent on this problem (see the bibliographies by Lagarias [10,11]), the conjecture remains unsolved, as the iterated function often behaves rather unpredictably. For this reason, iterations of the Collatz function have been studied in the research of small Turing machines. Margenstern [12] conjectures that every class of Turing machines (as characterized by the number of states and symbols) that contains a machine that is able to simulate the iteration of the Collatz function, also contains a machine that has an undecidable halting problem.

Similar to the definition of $\mathrm{VALC}_{\mathrm{E}}(I)$ and $\mathrm{VALC}_{\mathrm{NE}}(I)$, we encode those iterations of the Collatz function that lead to the number 1 (and thus, to the trivial cycle) in languages over the alphabet $\{0, \#\}$. For every $N \in \mathbb{N}_1$, let

$$\mathrm{TRIV}_{\mathrm{E}}(N){:=}\{\#0^{\mathcal{C}^0(N)}\#0^{\mathcal{C}^1(N)}\#\ldots\#0^{\mathcal{C}^n(N)}\# \mid n \geq 1, \mathcal{C}^n(N) = 1\},$$
$$\mathrm{TRIV}_{\mathrm{NE}}(N){:=}\{\#0^{6+\mathcal{C}^0(N)}\#0^{6+\mathcal{C}^1(N)}\#\ldots\#0^{6+\mathcal{C}^n(N)}\# \mid n \geq 1, \mathcal{C}^n(N) = 1\}.$$

By definition, $\mathrm{TRIV}_{\mathrm{E}}(N)$ (and $\mathrm{TRIV}_{\mathrm{NE}}(N)$) are empty if and only if N does not lead $\mathcal{C}$ into the trivial cycle. As we shall see, our constructions are able to express an even stronger problem, the question whether there are any numbers that lead $\mathcal{C}$ to a non-trivial cycle. We define $\mathrm{NTCC_E}$ as the set of all strings

$$\#0^{\mathcal{C}^0(N)}\#0^{\mathcal{C}^1(N)}\#\ldots\#0^{\mathcal{C}^n(N)}\#,$$

where $n, N \geq 1$, $\mathcal{C}^i(N) \neq 1$ for all $i \in \{0, \ldots, n\}$, and $\mathcal{C}^j(N) = \mathcal{C}^n(N)$ for some $j < n$. Analogously, $\mathrm{NTCC_{NE}}$ is defined to be the set of all strings

$$\#0^{6+\mathcal{C}^0(N)}\#0^{6+\mathcal{C}^1(N)}\#\ldots\#0^{6+\mathcal{C}^n(N)}\#,$$

with the same restrictions on n and N. Obviously, both sets are nonempty if and only if there exist non-trivial cycles in the iteration of $\mathcal{C}$. This is one of the two possible cases that would disprove the Collatz Conjecture, the other being the existence of a number N with $\mathcal{C}^i(N) \neq \mathcal{C}^j(N)$ for all $i \neq j$.

3 Main Results

In this section, we study the inclusion problems of various classes of pattern languages generated by patterns with a bounded number of variables.

As shown by Jiang et al. [9], the *general inclusion problem for pattern languages* is undecidable, both in the case of E- and NE-patterns:

Theorem 1 (Jiang et al. [9]). *Let $Z \in \{\mathrm{E}, \mathrm{NE}\}$. There is no total computable function χ_Z which, for every alphabet Σ and for every pair of patterns $\alpha, \beta \in \mathrm{Pat}_\Sigma$, decides on whether or not $L_{Z,\Sigma}(\alpha) \subseteq L_{Z,\Sigma}(\beta)$.*

The proof for the E-case uses an involved construction that relies heavily on the unboundedness of the terminal alphabet Σ. For the NE-case, Jiang et al. give a complicated reduction of the inclusion problem for ePAT_Σ to the inclusion problem for $\mathrm{nePAT}_{\Sigma_2}$, where Σ_2 is an alphabet with two additional terminals. As shown by Freydenberger and Reidenbach [6], the inclusion problem remains undecidable for most cases of a fixed terminal alphabet:

Theorem 2 (Freydenberger and Reidenbach [6]). *Let Σ be a finite alphabet. If $|\Sigma| \geq 2$, the inclusion problem of ePAT_Σ is undecidable. If $|\Sigma| \geq 4$, the inclusion problem of nePAT_Σ is undecidable.*

The proof for the E-case consists of a major modification of the construction for the general inclusion problem for E-pattern languages, and relies on the presence of an unbounded number of variables in one of the patterns. The NE-case of the result follows from the same reduction as in the proof of Theorem 1 (thus, the difference in $|\Sigma|$), and also relies on an unbounded number of variables.

As patterns with an arbitrarily large number of variables might seem somewhat artificial for many applications, we consider it natural to bound this number in order to gain decidability of (or at least further insights on) the inclusion of pattern languages. We begin our considerations with an observation from two classical papers on pattern languages:

Theorem 3 (Angluin [1], Jiang et al. [8]). *The inclusion problem for* nePAT_Σ *in* $\mathrm{nePAT}_{1,\Sigma}$ *and the inclusion problem for* ePAT_Σ *in* $\mathrm{ePAT}_{1,\Sigma}$ *are decidable.*

The proofs for both cases of this theorem rely on the following sufficient condition for inclusion of pattern languages:

Theorem 4 (Jiang et al. [8], Angluin [1]). *Let Σ be an alphabet and $\alpha, \beta \in \mathrm{Pat}_\Sigma$. If there is a terminal-preserving morphism $\phi : (\Sigma \cup X)^* \to (\Sigma \cup X)^*$ with $\phi(\beta) = \alpha$, then $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$. If ϕ is also nonerasing, then $L_{\mathrm{NE},\Sigma}(\alpha) \subseteq L_{\mathrm{NE},\Sigma}(\beta)$.*

In fact, the proofs of both parts of Theorem 3 show that, for every alphabet Σ and all patterns $\alpha \in \mathrm{Pat}_\Sigma$, $\beta \in \mathrm{Pat}_{1,\Sigma}$, $L(\alpha) \subseteq L(\beta)$ holds *if and only if* there is a terminal-preserving (and, in the NE-case, nonerasing) morphism ϕ with $\phi(\beta) = \alpha$. As the existence of such a morphism is a decidable property (although in general NP-complete, cf. Ehrenfeucht and Rozenberg [5]), the respective inclusion problems for these classes are decidable.

There are numerous other classes of pattern languages where this condition is not only sufficient, but characteristic; e. g. the terminal-free E-pattern languages (cf. Jiang et al. [9]), some of their generalizations (cf. Ohlebusch and Ukkonen [16]), and pattern languages over infinite alphabets (cf. Freydenberger and Reidenbach [6]). As far as we know, all non-trivial decidability results for pattern languages over non-unary alphabets rely on this property[1]. Contrariwise, the existence of patterns where inclusion is not characterized by the existence of an appropriate morphism between them is a necessary condition for an undecidable inclusion problem for this class.

The same phenomenon as in Theorem 3 does not occur if we swap the bounds. For the nonerasing case, this is illustrated by the following example:

Example 1 (Reidenbach [17], Example 3.2). Let $\Sigma = \{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$ with $n \geq 2$, and consider the pattern $\alpha_n := x\,\mathtt{a}_1\,x\,\mathtt{a}_2\,x \ldots x\,\mathtt{a}_n\,x$, $\beta := xyyz$. Then there is no terminal-preserving morphism ϕ with $\phi(\beta) = \alpha_n$, but every word from $L_{\mathrm{NE},\Sigma}(\alpha_n)$ contains an inner square. Thus, $L_{\mathrm{NE},\Sigma}(\alpha_n) \subseteq L_{\mathrm{NE},\Sigma}(\beta)$. ◇

Thus, regardless of the size of $|\Sigma|$, even the inclusion problem of $\mathrm{nePAT}_{1,\Sigma}$ in $\mathrm{nePAT}_{3,\Sigma}$ is too complex to be characterized by the existence of a nonerasing terminal-preserving morphism between the patterns.

A similar phenomenon can be observed for E-pattern languages:

Proposition 1. *For every finite alphabet Σ with $|\Sigma| \geq 2$, there are patterns $\alpha \in \mathrm{Pat}_{1,\Sigma}$ and $\beta \in \mathrm{Pat}_{2|\Sigma|+2,\Sigma}$ such that $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$, but there is no terminal-preserving morphism $\phi : (\Sigma \cup X)^* \to (\Sigma \cup X)^*$ with $\phi(\beta) = \alpha$.*

The proof for Proposition 1 is omitted due to space constraints.

The proof also shows that, if Σ has an odd number of letters, the bound on the number of variables in the second class of patterns can be lowered to $2|\Sigma|$.

[1] Non-trivial meaning that the involved classes are neither finite, nor restricted in some artificial way that leads to trivial decidability.

We do not know whether this lower bound is strict, or if there are patterns $\alpha \in \mathrm{Pat}_{1,\Sigma}$, $\beta \in \mathrm{Pat}_{n,\Sigma}$ with $n < 2|\Sigma|$ such that $L_{\mathrm{E},\Sigma}(\alpha) \subseteq L_{\mathrm{E},\Sigma}(\beta)$, but there is no terminal-preserving morphism mapping β to α.

For $|\Sigma| = 2$, according to Proposition 1, the inclusion of $\mathrm{ePAT}_{1,\Sigma}$ in $\mathrm{ePAT}_{6,\Sigma}$ is not characterized by the existence of such a morphism. As this bound (and the bound on NE-patterns from Example 1) are the lowest known bounds for 'morphism-free' inclusion, we want to emphasize the following problem:

Open Problem 1 *Let $|\Sigma| = 2$. Is the inclusion problem of $\mathrm{ePAT}_{1,\Sigma}$ in $\mathrm{ePAT}_{6,\Sigma}$ decidable? Is the inclusion problem of $\mathrm{nePAT}_{1,\Sigma}$ in $\mathrm{nePAT}_{3,\Sigma}$ decidable?*

In principle, both inclusion problems might be undecidable; but comparing these bounds to the ones in the following results, this seems somewhat improbable, and suggests that if these problems are undecidable, the proof would need to be far more complicated than the proofs in the present paper. On the other hand, these classes are promising candidates for classes of pattern languages where the inclusion is decidable, but not characterized by the existence of an appropriate morphism.

As evidenced by our first two main theorems, bounding the number of variables preserves the undecidability of the inclusion problem:

Theorem 5. *Let $|\Sigma| = 2$. The following problems are undecidable:*

1. *The inclusion problem of $\mathrm{ePAT}_{3,\Sigma}$ in $\mathrm{ePAT}_{2854,\Sigma}$,*
2. *the inclusion problem of $\mathrm{ePAT}_{2,\Sigma}$ in $\mathrm{ePAT}_{2860,\Sigma}$.*

Theorem 6. *Let $|\Sigma| = 2$. The following problems are undecidable:*

1. *The inclusion problem of $\mathrm{nePAT}_{3,\Sigma}$ in $\mathrm{nePAT}_{3541,\Sigma}$,*
2. *the inclusion problem of $\mathrm{nePAT}_{2,\Sigma}$ in $\mathrm{nePAT}_{3549,\Sigma}$.*

Note that the cases of all larger (finite) alphabets are handled in Section 4.1. The bounds presented in these two theorems are not optimal. Through additional effort and some encoding tricks, it is possible to reduce each bound on the number of variables in the second pattern by a few hundred variables. As the resulting number would still be far away from the bounds presented in the theorems further down in this section, we felt that these optimizations would only add additional complexity to the proofs, without providing deeper insight, and decided to give only the less optimal bounds present above.

The proofs for both theorems use the same basic approach as the proofs of the E-case in Theorems 1 and 2. We show that, for a given configuration I of U, one can effectively construct patterns α, β in the appropriate classes of patterns such that $L(\alpha) \subseteq L(\beta)$ if and only if U halts after starting in I. As this would decide the halting problem of the universal Turing machine U, the inclusion problems must be undecidable.

For the E-case, we show this using a nontrivial but comparatively straightforward modification of the proof for the E-case of Theorem 2. As this construction is still very complicated, a brief sketch can be found in Section 3.1, while the full construction is omitted due to space constraints.

For the NE-case, we show that a comparable construction can be realized with NE-patterns. This observation is less obvious than it might appear and requires extensive modifications to the E-construction. As previous results on the non-decidability of the inclusion problem for NE-patterns rely on an involved construction from [9], we consider the construction used for our proof of Theorem 6 a significant technical breakthrough; especially as this result (together with its extension following from the modification in Section 4.1) allows us to solve Open Problem 1 in [6], concluding that the inclusion problem for NE-patterns over binary and ternary alphabets is undecidable. Some remarks on the construction are sketched in Section 3.2, while the full construction is omitted.

Although encoding the correct operation of a Turing machine (or any similar device) in patterns requires a considerable amount of variables, the simple structure of iterating the Collatz function $\mathcal{C}$ can be expressed in a more compact form. With far smaller bounds, we are able to obtain the following two results using the same constructions as for the proof of Theorems 5 and 6:

Theorem 7. *Let Σ be a binary alphabet. Every algorithm that decides the inclusion problem of* $\text{ePAT}_{2,\Sigma}$ *in* $\text{ePAT}_{74,\Sigma}$ *can be converted into an algorithm that, for every $N \in \mathbb{N}_1$, decides whether N leads $\mathcal{C}$ into the trivial cycle.*

Theorem 8. *Let Σ be a binary alphabet. Every algorithm that decides the inclusion problem of* $\text{nePAT}_{2,\Sigma}$ *in* $\text{nePAT}_{145,\Sigma}$ *can be converted into an algorithm that, for every $N \in \mathbb{N}_1$, decides whether N leads $\mathcal{C}$ into the trivial cycle.*

The proofs are sketched in Sections 3.1 and 3.2. As mentioned in Section 2.3, this demonstrates that, even for these far tighter bounds, the inclusion problems are able to express comparatively complicated sets. Moreover, a slight modification of the result allows us to state the following far stronger results:

Theorem 9. *Let Σ be a binary alphabet. Every algorithm that decides the inclusion problem for* $\text{ePAT}_{4,\Sigma}$ *in* $\text{ePAT}_{80,\Sigma}$ *can be used to decide whether any number $N \geq 1$ leads $\mathcal{C}$ into a non-trivial cycle.*

Theorem 10. *Let Σ be a binary alphabet. Every algorithm that decides the inclusion problem for* $\text{nePAT}_{4,\Sigma}$ *in* $\text{nePAT}_{153,\Sigma}$ *can be used to decide whether any number $N \geq 1$ leads $\mathcal{C}$ into a non-trivial cycle.*

The proofs are sketched in Sections 3.1 and 3.2. These two results need to be interpreted very carefully. Of course, the existence of non-trivial cycles is trivially decidable (by a constant predicate); but these results are stronger than mere decidability, as the patterns are constructed effectively. Thus, deciding the inclusion of any of the two pairs of patterns defined in the proofs would allow us to prove the existence of a counterexample to the Collatz Conjecture, or to rule out the existence of one important class of counterexamples, and thus solve 'one half' of the Collatz Conjecture. More pragmatically, we think that these results give reason to suspect that the inclusion problems of these classes of pattern languages are probably not solvable (even if effectively, then not efficiently), and definitely very complicated.

3.1 Sketch of the Construction for E-Patterns

As the construction is rather involved, we only give a basic sketch, and omit the full technical details. In each of the proofs, our goal is to decide the emptiness of a set V, which is one of $\mathrm{TRIV}_{\mathrm{E}}(N)$ (for some $N \geq 1$), $\mathrm{NTCC}_{\mathrm{E}}$, or $\mathrm{VALC}_{\mathrm{E}}(I)$ (for some configuration I). For this, we construct two patterns α and β such that $L_{\mathrm{E},\Sigma}(\alpha) \setminus L_{\mathrm{E},\Sigma}(\beta) \neq \emptyset$ if and only if $V \neq \emptyset$. The pattern α contains two subpatterns α_1 and α_2, where α_2 is a terminal-free pattern with $\mathrm{var}(\alpha_2) \subseteq \mathrm{var}(\alpha_1) \cup \{y\}$, and y is a variable that occurs exactly once in α_2, but does not occur in α_1.

Glossing over details (and ignoring the technical role of α_2), the main goal is to define β in such a way that, for every substitution σ, $\sigma(\alpha) \in L_{\mathrm{E},\Sigma}(\beta)$ if and only if $\sigma(\alpha_1) \in \mathrm{V}$. More explicitly, the subpattern α_1 generates a set of possible strings, and β encodes a disjunction of predicates on strings that describe the complement of V through all possible errors. If one of these errors occurs in $\sigma(\alpha_1)$, we can construct a substitution τ with $\tau(\beta) = \sigma(\alpha)$. If $\mathrm{V} = \emptyset$, every $\sigma(\alpha)$ belongs to $L_{\mathrm{E},\Sigma}(\beta)$. Otherwise, any element of V can be used to construct a word $\sigma(\alpha) \notin L_{\mathrm{E},\Sigma}(\beta)$. The proof of Theorem 2 in [6] can be interpreted as a special case of this construction, using $\alpha_1 := x$ and $\alpha_2 := y$. Through our modification, we are able to exert more control on the elements of $L_{\mathrm{E},\Sigma}(\alpha_1)$, and use this to define required repetitions, prefixes or suffixes for all $\sigma(\alpha_1)$ with $\sigma(\alpha) \notin L_{\mathrm{E},\Sigma}(\beta)$. The variables in $\mathrm{var}(\alpha_2) \setminus \{y\}$ are even further restricted, and can only be mapped to 0^*.

3.2 Sketch of the Construction for NE-Patterns

The detailed definition of the construction, and the associated proofs, are omitted due to space constraints. Describing the NE-construction on the same level of detail as the E-construction, both appear to be identical, including the presence and the role of subpatterns α_1 and α_2 in α. But as evidenced in the full proof, the peculiarities of NE-patterns require considerable additional technical effort. For example, the E-construction heavily depends on being able to map most variables in β to the empty word; dealing with these 'superfluous' variables is the largest difficulty for the modification. In order to overcome this problem, the pattern α contains long terminal-strings, which makes it possible to map every variable in β to at least one terminal. These terminal-strings complicate one of the main proofs for the E-construction, as we have to ensure that these terminal-strings do not prevent a necessary mapping, while not allowing any unintended mappings. The E-construction uses a set of variables x_i of which, under some preconditions, all but one have to be mapped to the empty word. That variable is then used to enforce certain decompositions of β in a way that allows us to encode the predicates in a system of word equations. In the NE-construction, we use a more complicated prefix-construction to obtain a set of variables, which (again under some preconditions) all but one have to be mapped to the terminal 0, while the single remaining variable has to be mapped to the terminal #. Some minor changes make sure that the number of different variables in β does not increase too much in comparison to the E-construction

– this is one reason for the different definitions of the encoding sets for the erasing and the nonerasing case in Sections 2.2 and 2.3. On the other hand, the modifications of the construction and the use of nonerasing substitutions make the implementation of the extensions in Section 4 simpler than for the erasing case.

4 Extensions of the Main Theorems

In this section, we extend the main theorems of the previous section to larger alphabets (Section 4.1), and show that all patterns from the second class can be replaced with terminal-free patterns (Section 4.2).

4.1 Larger Alphabets

As mentioned in Lemma 5 in [6], the construction for E-patterns can be adapted to all finite alphabets $|\Sigma|$ with $|\Sigma| \geq 3$. This modification is comparatively straightforward, but would require $2(|\Sigma|-2)$ additional predicates, and increase the number of required variables in β by $|\Sigma| - 2$ for each predicate. With additional effort, both constructions can be adapted to arbitrarily large alphabets:

Theorem 11. *Let Σ be a finite alphabet with $|\Sigma| \geq 3$. The following problems are undecidable:*

1. *The inclusion problem of* $\mathrm{ePAT}_{2,\Sigma}$ *in* $\mathrm{ePAT}_{2882,\Sigma}$*,*
2. *the inclusion problem of* $\mathrm{nePAT}_{2,\Sigma}$ *in* $\mathrm{nePAT}_{3563,\Sigma}$*.*

The required modifications and the proof of their correctness for the E- and the NE-construction are omitted. Using the same modifications to the constructions, the remaining cases from Theorems 5 and 6 and Theorems 7 to 10 can also be adapted to ternary (or larger) alphabets, using only 22 additional variables.

4.2 Inclusion in $\mathbf{ePAT_{tf,\Sigma}}$ or $\mathbf{nePAT_{tf,\Sigma}}$

Both constructions can also be adapted to use terminal-free patterns β:

Theorem 12. *Let $|\Sigma| = 2$. The following problems are undecidable:*

1. *The inclusion problem of* $\mathrm{ePAT}_{2,\Sigma}$ *in* $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$*,*
2. *the inclusion problem of* $\mathrm{nePAT}_{2,\Sigma}$ *in* $\mathrm{nePAT}_{\mathrm{tf},\Sigma}$*.*

Again, the technical details are omitted for space reasons. Note that the number of variables in the patterns from $\mathrm{Pat}_{\mathrm{tf}}$ remains bounded. Although one might expect that this result could be modified to show that the open inclusion problem for $\mathrm{nePAT}_{\mathrm{tf},\Sigma}$ is undecidable, we consider this doubtful, as the modified NE-construction relies heavily on the terminal symbols in α. Furthermore, although it is considerably easier to modify the NE-construction, the fact that the inclusion problem for $\mathrm{ePAT}_{\mathrm{tf},\Sigma}$ is decidable casts further doubt on that expectation. As in Section 4.1, all other results that are based on one of the two constructions can be adapted as well.

References

1. Angluin, D.: Finding patterns common to a set of strings. In: Proc. 11th Annual ACM Symposium on Theory of Computing, pp. 130–141 (1979)
2. Câmpeanu, C., Salomaa, K., Yu, S.: A formal study of practical regular expressions. Int. J. Found. Comput. Sci. 14, 1007–1018 (2003)
3. Choffrut, C., Karhumäki, J.: Combinatorics of words. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, ch. 6, vol. 1, pp. 329–438. Springer, Heidelberg (1997)
4. Durnev, V.G.: Undecidability of the positive $\forall\exists^3$-theory of a free semigroup. Siberian Math. J. 36(5), 917–929 (1995)
5. Ehrenfeucht, A., Rozenberg, G.: Finding a homomorphism between two words is NP-complete. Inform. Process. Lett. 9, 86–88 (1979)
6. Freydenberger, D.D., Reidenbach, D.: Bad news on decision problems for patterns. Inform. and Comput. 208(1), 83–96 (2010)
7. Ibarra, O.H., Pong, T.C., Sohn, S.M.: A note on parsing pattern languages. Pattern Recognit. Lett. 16(2), 179–182 (1995)
8. Jiang, T., Kinber, E., Salomaa, A., Salomaa, K., Yu, S.: Pattern languages with and without erasing. Int. J. Comput. Math. 50, 147–163 (1994)
9. Jiang, T., Salomaa, A., Salomaa, K., Yu, S.: Decision problems for patterns. J. Comput. Syst. Sci. 50, 53–63 (1995)
10. Lagarias, J.C.: The 3x+1 problem: An annotated bibliography (1963-1999) (August 2009), http://arxiv.org/abs/math/0309224
11. Lagarias, J.C.: The 3x+1 problem: An annotated bibliography, II (2000-2009) (August 2009), http://arxiv.org/abs/math/0608208
12. Margenstern, M.: Frontier between decidability and undecidability: a survey. Theor. Comput. Sci. 231(2), 217–251 (2000)
13. Mitrana, V.: Patterns and languages: An overview. Grammars 2(2), 149–173 (1999)
14. Neary, T., Woods, D.: Four small universal Turing machines. Fundam. Inform. 91(1), 123–144 (2009)
15. Ng, Y.K., Shinohara, T.: Developments from enquiries into the learnability of the pattern languages from positive data. Theor. Comput. Sci. 397, 150–165 (2008)
16. Ohlebusch, E., Ukkonen, E.: On the equivalence problem for E-pattern languages. Theor. Comput. Sci. 186, 231–248 (1997)
17. Reidenbach, D.: The Ambiguity of Morphisms in Free Monoids and its Impact on Algorithmic Properties of Pattern Languages. PhD thesis, Fachbereich Informatik, Technische Universität Kaiserslautern, Logos Verlag, Berlin (2006)
18. Salomaa, K.: Patterns. In: Martin-Vide, C., Mitrana, V., Păun, G. (eds.) Formal Languages and Applications. Studies in Fuzziness and Soft Computing, vol. 148, pp. 367–379. Springer, Heidelberg (2004)
19. Shinohara, T.: Polynomial time inference of pattern languages and its application. In: Proc. 7th IBM Symposium on Mathematical Foundations of Computer Science, pp. 191–209 (1982)
20. Thue, A.: Über unendliche Zeichenreihen. Kra. Vidensk. Selsk. Skrifter. I. Mat. Nat. Kl 7 (1906)

On the Average Number of States of Partial Derivative Automata*

Sabine Broda[1], António Machiavelo[2], Nelma Moreira[1], and Rogério Reis[1]

[1] DCC-FC & LIACC, Universidade do Porto
Rua do Campo Alegre 1021/1055, 4169-007 Porto, Portugal
[2] CMUP, Universidade do Porto
Rua do Campo Alegre 687, 4169-007 Porto, Portugal
sbb@ncc.up.pt, ajmachia@fc.up.pt, {nam,rvr}@ncc.up.pt

Abstract. The partial derivative automaton ($\mathcal{A}_{\text{pd}}$) is usually smaller than other non-deterministic finite automata constructed from a regular expression, and it can be seen as a quotient of the Glushkov automaton ($\mathcal{A}_{\text{pos}}$). By estimating the number of regular expressions that have ε as a partial derivative, we compute a lower bound of the average number of mergings of states in $\mathcal{A}_{\text{pos}}$ and describe its asymptotic behaviour. This depends on the alphabet size, k, and its limit, as k goes to infinity, is $\frac{1}{2}$. The lower bound corresponds exactly to consider the $\mathcal{A}_{\text{pd}}$ automaton for the marked version of the regular expression, i.e. where all its letters are made different. Experimental results suggest that the average number of states of this automaton, and of the $\mathcal{A}_{\text{pd}}$ automaton for the unmarked regular expression, are very close to each other.

1 Introduction

There are several well-known constructions to obtain non-deterministic finite automata from regular expressions. The worst case analysis of both the complexity of the conversion algorithms, and the size of the resulting automata, are well studied. However, for practical purposes, the average case analysis can provide a much more useful information. Recently, Nicaud [Nic09] presented an average case study of the size of the Glushkov automata, proving that, on average, the number of transitions is linear in the size of the expression. This analysis was carried out using the framework of analytic combinatorics.

Following the same approach, in this paper we focus on the partial derivative automaton ($\mathcal{A}_{\text{pd}}$), which was introduced by Antimirov [Ant96], and is a non-deterministic version of the Brzozowski automaton [Brz64]. In order to have an inductive definition of the set of states of $\mathcal{A}_{\text{pd}}$, we consider Mirkin's formulation of prebases. The equivalence of the two constructions, Mirkin's prebases, and sets of partial derivatives, was pointed out by Champarnaud and Ziadi [CZ01]. We briefly revisit Mirkin's algorithm, due to an inaccuracy in that presentation.

* This work was partially funded by Fundação para a Ciência e Tecnologia (FCT) and Program POSI, and by projects RESCUE (PTDC/EIA/65862/2006) and CANTE (PTDC/EIA-CCO/101904/2008).

In 2002, Champarnaud and Ziadi [CZ02] showed that the partial derivative automaton is a quotient of the Glushkov automaton. As such, the $\mathcal{A}_{\text{pd}}$ automaton can be obtained from the $\mathcal{A}_{\text{pos}}$ automaton by merging states. The number of states in $\mathcal{A}_{\text{pd}}$, which never exceeds the number of states in $\mathcal{A}_{\text{pos}}$, appears to be significantly smaller in practice. In this work we are particularly interested in measuring this difference. By estimating the number of regular expressions that have ε as a partial derivative, we compute a lower bound for the average number of mergings of states in $\mathcal{A}_{\text{pos}}$, and study its asymptotic behaviour. This behaviour depends on the alphabet size, k, and its limit, as k goes to infinity, is half the number of states in $\mathcal{A}_{\text{pos}}$. Our experimental results suggest that this lower bound is very close to the actual value.

2 Regular Expressions and Automata

In this section we briefly review some basic definitions about regular expressions and finite automata. For more details, we refer the reader to Kozen [Koz97] or Sakarovitch [Sak09].

Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be an *alphabet* (set of *letters*) of size k. A *word* w over Σ is any finite sequence of letters. The *empty word* is denoted by ε. Let $\Sigma^\star$ be the set of all words over Σ. A *language* over Σ is a subset of $\Sigma^\star$. The set R of *regular expressions* over Σ is defined by:

$$\alpha := \emptyset \mid \varepsilon \mid \sigma_1 \mid \cdots \mid \sigma_k \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid \alpha^\star \tag{1}$$

where the operator $\cdot$ (concatenation) is often omitted. The language $\mathcal{L}(\alpha)$ associated to α is inductively defined as follows: $\mathcal{L}(\emptyset) = \emptyset$, $\mathcal{L}(\varepsilon) = \{\varepsilon\}$, $\mathcal{L}(\sigma) = \{\sigma\}$ for $\sigma \in \Sigma$, $\mathcal{L}((\alpha + \beta)) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$, $\mathcal{L}((\alpha \cdot \beta)) = \mathcal{L}(\alpha) \cdot \mathcal{L}(\beta)$, and $\mathcal{L}(\alpha^\star) = \mathcal{L}(\alpha)^\star$. The *size* $|\alpha|$ of $\alpha \in \mathsf{R}$ is the number of symbols in α (parentheses not counted); the *alphabetic size* $|\alpha|_\Sigma$ is its number of letters. We define $\varepsilon(\alpha)$ by $\varepsilon(\alpha) = \varepsilon$ if $\varepsilon \in \mathcal{L}(\alpha)$, and $\varepsilon(\alpha) = \emptyset$ otherwise. If two regular expressions α and β are syntactically identical we write $\alpha \equiv \beta$. Two regular expressions α and β are *equivalent* if $\mathcal{L}(\alpha) = \mathcal{L}(\beta)$, and we write $\alpha = \beta$. With this interpretation, the algebraic structure $(\mathsf{R}, +, \cdot, \emptyset, \varepsilon)$ constitutes an idempotent semiring, and with the unary operator $\star$, a Kleene algebra.

A *non-deterministic automaton* (NFA) $\mathcal{A}$ is a quintuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is the alphabet, $\delta \subseteq Q \times \Sigma \times Q$ the transition relation, q_0 the initial state, and $F \subseteq Q$ the set of final states. The *size* of a NFA is $|Q| + |\delta|$. For $q \in Q$ and $\sigma \in \Sigma$, we denote the set $\{p \mid (q, \sigma, p) \in \delta\}$ by $\delta(q, \sigma)$, and we can extend this notation to $w \in \Sigma^\star$, and to $R \subseteq Q$. The *language* accepted by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^\star \mid \delta(q_0, w) \cap F \neq \emptyset\}$.

3 The Partial Derivative Automaton

Let $S \cup \{\beta\}$ be a set of regular expressions. Then $S \odot \beta = \{\alpha\beta \,|\, \alpha \in S\}$ if $\beta \notin \{\emptyset, \varepsilon\}$, $S \odot \emptyset = \emptyset$, and $S \odot \varepsilon = S$. Analogously, one defines $\beta \odot S$.

For a regular expression α and a letter $\sigma \in \Sigma$, the set $\partial_\sigma(\alpha)$ of *partial derivatives* of α w.r.t. σ is defined inductively as follows:

$$\begin{aligned}
\partial_\sigma(\emptyset) &= \partial_\sigma(\varepsilon) = \emptyset & \partial_\sigma(\alpha+\beta) &= \partial_\sigma(\alpha) \cup \partial_\sigma(\beta)\\
\partial_\sigma(\sigma') &= \begin{cases} \{\varepsilon\} & \text{if } \sigma' \equiv \sigma \\ \emptyset & \text{otherwise} \end{cases} & \partial_\sigma(\alpha\beta) &= \begin{cases} \partial_\sigma(\alpha) \odot \beta \cup \partial_\sigma(\beta) & \text{if } \varepsilon(\alpha) = \varepsilon \\ \partial_\sigma(\alpha) \odot \beta & \text{otherwise.} \end{cases}\\
\partial_\sigma(\alpha^\star) &= \partial_\sigma(\alpha) \odot \alpha^\star
\end{aligned}$$

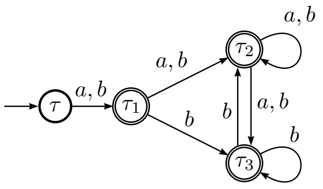
This definition can be extended to sets of regular expressions, to words, and to languages by: given $\alpha \in \mathsf{R}$ and $\sigma \in \Sigma$, $\partial_\sigma(S) = \cup_{\beta \in S}\partial_\sigma(\beta)$ for $S \subseteq \mathsf{R}$; $\partial_\varepsilon(\alpha) = \{\alpha\}$, $\partial_{w\sigma}(\alpha) = \partial_\sigma(\partial_w(\alpha))$ for $w \in \Sigma^\star$; and $\partial_L(\alpha) = \cup_{w \in L}\partial_w(\alpha)$ for $L \subseteq \Sigma^\star$. The *set of partial derivatives* of α, $\{\partial_w(\alpha) \mid w \in \Sigma^\star\}$, is denoted by $\mathrm{PD}(\alpha)$. The partial derivative automaton $\mathcal{A}_{\mathrm{pd}}(\alpha)$, introduced by Antimirov, is defined by

$$\mathcal{A}_{\mathrm{pd}}(\alpha) = (\mathrm{PD}(\alpha), \Sigma, \delta_{pd}, \alpha, \{q \in \mathrm{PD}(\alpha) \mid \varepsilon(q) = \varepsilon\}),$$

where $\delta_{pd}(q, \sigma) = \partial_\sigma(q)$, for all $q \in \mathrm{PD}(\alpha)$ and $\sigma \in \Sigma$.

Proposition 1 (Antimirov). $\mathcal{L}(\mathcal{A}_{\mathrm{pd}}(\alpha)) = \mathcal{L}(\alpha)$.

Example 1. Throughout the paper we will use the regular expression $\tau = (a + b)(a^\star + ba^\star + b^\star)^\star$, given by Ilie and Yu [IY03]. This example illustrates perfectly the purpose of our constructions in Section 5.1. For τ one has, $\mathrm{PD}(\tau) = \{\tau, \tau_1, \tau_2, \tau_3\}$, where $\tau_1 = (a^\star + ba^\star + b^\star)^\star$, $\tau_2 = a^\star\tau_1$ and $\tau_3 = b^\star\tau_1$. The corresponding automaton $\mathcal{A}_{\mathrm{pd}}(\tau)$ is the following:



3.1 Mirkin's Formulation

Champarnaud and Ziadi [CZ01] showed that partial derivatives and Mirkin's prebases [Mir66] lead to identical constructions of non-deterministic automata. In order to do this, they proposed a recursive algorithm for computing the Mirkin's prebases. However, that algorithm has an inaccuracy for the concatenation rule. Here, we give the corrected version of the algorithm.

Let α_0 be a regular expression. A set $\pi(\alpha_0) = \{\alpha_1, \ldots, \alpha_n\}$, where $\alpha_1, \ldots, \alpha_n$ are non-empty regular expressions, is called a *support* of α_0 if, for $i = 0, \ldots, n$, there are $\alpha_{il} \in \mathsf{R}$ ($l = 1, \ldots, k$), linear combinations of the elements in $\pi(\alpha_0)$, such that $\alpha_i = \sigma_1 \cdot \alpha_{i1} + \ldots + \sigma_k \cdot \alpha_{ik} + \varepsilon(\alpha_i)$, where, as above, $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is the considered alphabet. If $\pi(\alpha)$ is a support of α, then the set $\pi(\alpha) \cup \{\alpha\}$ is called a *prebase* of α.

Proposition 2 (Mirkin/Champarnaud&Ziadi[1]). *Let α be a regular expression. Then the set $\pi(\alpha)$, inductively defined by*

$$\begin{aligned} \pi(\emptyset) &= \emptyset & \pi(\alpha+\beta) &= \pi(\alpha)\cup\pi(\beta) \\ \pi(\varepsilon) &= \emptyset & \pi(\alpha\beta) &= \pi(\alpha)\odot\beta\cup\pi(\beta) \\ \pi(\sigma) &= \{\varepsilon\} & \pi(\alpha^\star) &= \pi(\alpha)\odot\alpha^\star, \end{aligned}$$

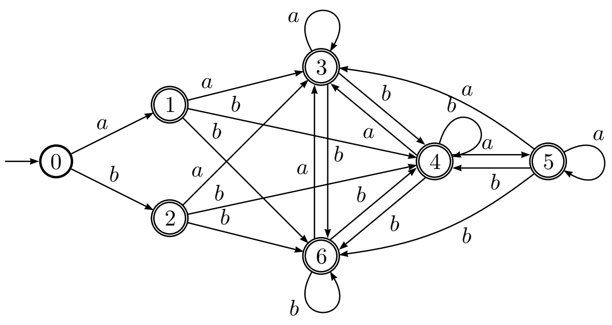
is a support of α.

In his original paper Mirkin showed that $|\pi(\alpha)| \leq |\alpha|_\Sigma$. Furthermore, Champarnaud and Ziadi established that $\mathrm{PD}(\alpha) = \pi(\alpha)\cup\{\alpha\}$. Thus $|\mathrm{PD}(\alpha)| \leq |\alpha|_\Sigma + 1$.

3.2 The Glushkov Automaton

To review the definition of the Glushkov automaton, let $\mathrm{Pos}(\alpha) = \{1, 2, \ldots, |\alpha|_\Sigma\}$ be the set of positions for $\alpha \in \mathsf{R}$, and let $\mathrm{Pos}_0(\alpha) = \mathrm{Pos}(\alpha)\cup\{0\}$. We consider the expression $\overline{\alpha}$ obtained by marking each letter with its position in α, i.e. $\mathcal{L}(\overline{\alpha}) \in \overline{\Sigma}^\star$ where $\overline{\Sigma} = \{\sigma^i \mid \sigma\in\Sigma, 1\leq i\leq|\alpha|_\Sigma\}$. The same notation is used to remove the markings, i.e., $\overline{\overline{\alpha}} = \alpha$. For $\alpha\in\mathsf{R}$ and $i\in\mathrm{Pos}(\alpha)$, let $\mathsf{first}(\alpha) = \{i \mid \exists w\in\overline{\Sigma}^\star, \sigma^i w\in\mathcal{L}(\overline{\alpha})\}$, $\mathsf{last}(\alpha) = \{i \mid \exists w\in\overline{\Sigma}^\star, w\sigma^i\in\mathcal{L}(\overline{\alpha})\}$ and $\mathsf{follow}(\alpha, i) = \{j \mid \exists u, v\in\overline{\Sigma}^\star, u\sigma^i\sigma^j v\in\mathcal{L}(\overline{\alpha})\}$. The *Glushkov automaton* for α is $\mathcal{A}_{\mathrm{pos}}(\alpha) = (\mathrm{Pos}_0(\alpha), \Sigma, \delta_{\mathrm{pos}}, 0, F)$, with $\delta_{\mathrm{pos}} = \{(0, \overline{\sigma^j}, j) \mid j\in\mathsf{first}(\alpha)\}\cup\{(i, \overline{\sigma^j}, j) \mid j\in\mathsf{follow}(\alpha, i)\}$ and $F = \mathsf{last}(\alpha)\cup\{0\}$ if $\varepsilon(\alpha) = \varepsilon$, and $F = \mathsf{last}(\alpha)$, otherwise. We note that the number of states of $\mathcal{A}_{\mathrm{pos}}(\alpha)$ is exactly $|\alpha|_\Sigma + 1$. Champarnaud and Ziadi [CZ02] showed that the partial derivative automaton is a quotient of the Glushkov automaton. The right-invariant equivalence relation used in showing that the $\mathcal{A}_{\mathrm{pd}}$ is a quotient of $\mathcal{A}_{\mathrm{pos}}$ relates the sets first and last with (multi-)sets of partial derivatives w.r.t a letter.

Example 2. The Glushkov automaton for τ, $\mathcal{A}_{\mathrm{pos}}(\tau)$, is the following:



[1] The rule for concatenation in [CZ01] is $\pi(\alpha\beta) = \pi(\alpha)\odot\beta\cup\varepsilon(\alpha)\odot\pi(\beta)$, which, e.g., produces $\pi(ab) = \{b\}$. But, $\mathrm{PD}(ab) = \{ab, b, \varepsilon\}$, thus $\pi(ab)\supseteq\{b,\varepsilon\}$.

4 Generating Functions and Analytic Methods

A *combinatorial class* C is a set of objects on which a non-negative integer function (size) $|\cdot|$ is defined, and such that for each $n \geq 0$, the number of objects of size n, c_n, is finite. The *generating function* $C(z)$ of C is the formal power series

$$C(z) = \sum_{c \in \mathsf{C}} z^{|c|} = \sum_{n=0}^{\infty} c_n z^n.$$

The symbolic method (Flajolet and Sedgewick [FS08]) is a framework that allows the construction of a combinatorial class C in terms of simpler ones, $\mathsf{B}_1,\ldots,\mathsf{B}_n$, by means of specific operations, and such that the generating function $C(z)$ of C is a function of the generating functions $B_i(z)$ of B_i, for $1 \leq i \leq n$. For example, given two disjoint combinatorial classes A and B, with generating functions $A(z)$ and $B(z)$, respectively, the union $A \cup B$ is a combinatorial class whose generating function is $A(z)+B(z)$. Other usual admissible operations are the cartesian product and the Kleene closure.

Usually multivariate generating functions are used in order to obtain estimates about the asymptotic behaviour of various parameters associated to combinatorial classes. Here, however, we consider *cost generating functions*, as Nicaud did. Given $f : \mathsf{C} \to \mathbb{N}$, the *cost generating function* $F(z)$ of C associated to f is $F(z) = \sum_{c\in\mathsf{C}} f(c)z^{|c|} = \sum_{n\geq 0} f_n z^n$, with $f_n = \sum_{c\in\mathsf{C},|c|=n} f(c)$. With $[z^n]F(z)$ denoting the coefficient of z^n, the average value of f for the uniform distribution on the elements of size n of C is, obviously,

$$\mu_n(\mathsf{C}, f) = \frac{[z^n]F(z)}{[z^n]C(z)}.$$

For the regular expressions given in (1), but without $\emptyset$, an average case analysis of different descriptional measures, including the number of letters or the size of its Glushkov automaton, has been presented by Nicaud. In particular, it was shown that, for the generating function for regular expressions, $R_k(z)$, which satisfies

$$R_k(z) = \frac{1 - z - \sqrt{\Delta_k(z)}}{4z}, \text{ where } \Delta_k(z) = 1 - 2z - (7+8k)z^2, \tag{2}$$

one has

$$[z^n]R_k(z) \sim \frac{\sqrt{2(1-\rho_k)}}{8\rho_k\sqrt{\pi}}\rho_k^n n^{-3/2}, \text{ where } \rho_k = \frac{1}{1+\sqrt{8k+8}}. \tag{3}$$

Here $[z^n]R_k(z)$ is the number of regular expressions α with $|\alpha| = n$.

Nicaud also showed that the cost generating function for the number of letters in an element $\alpha \in \mathsf{R}$ is

$$L_k(z) = \frac{kz}{\sqrt{\Delta_k(z)}}, \tag{4}$$

and satisfies

$$[z^n]L_k(z) \sim \frac{k\rho_k}{\sqrt{\pi(2-2\rho_k)}}\rho_k^{-n}n^{-1/2}. \tag{5}$$

From this he deduced that

$$\frac{[z^n]L_k(z)}{[z^n]R_k(z)} \sim \frac{4k\rho_k^2}{1-\rho_k}n. \tag{6}$$

For $k = 2$ this results in approximately $0.277n$ (and not $0.408n$, as stated by Nicaud), and it is easy to see that

$$\lim_{k\to\infty} \frac{4k\rho_k^2}{1-\rho_k} \nearrow \frac{1}{2}. \tag{7}$$

This means that the average number of letters in a regular expression grows to about half its size, for large alphabets. In particular, for $k = 10, 100, 1000$ we have $\frac{4k\rho_k^2}{1-\rho_k} = 0.467, 0.485, 0.494$ respectively.

4.1 Analytic Asymptotics

Generating functions can be seen as complex analytic functions, and the study of their behaviour around their dominant singularities gives us access to the asymptotic form of their coefficients. We refer the reader to Flajolet and Sedgewick for an extensive study on this topic. Here we only state the propositions and lemmas used in this paper. Let $R > 1$ and $0 < \phi < \pi/2$ be two real numbers, the *domain* $\Delta(\phi, R)$ at $z = \xi$ is $\Delta(\phi, R) = \{z \in \mathbb{C} \mid |z| < R,\ z \neq \xi, \text{ and } |Arg(z-\xi)| > \phi\}$, where $Arg(z)$ denotes the argument of $z \in \mathbb{C}$. A *domain* is a Δ-*domain* at ξ if it is a $\Delta(\phi, R)$ at ξ for some R and ϕ. The generating functions we consider have always a unique dominant singularity, and satisfy one of the two conditions of the following proposition, given by Nicaud.

Proposition 3. *Let $f(z)$ be a function that is analytic in some Δ-domain at $\rho \in \mathbb{R}^+$. If at the intersection of a neighborhood of ρ and its Δ-domain,*

1. $f(z) = a - b\sqrt{1-z/\rho} + o\left(\sqrt{1-z/\rho}\right)$, *with $a, b \in \mathbb{R}$, $b \neq 0$, then* $[z^n]f(z) \sim \frac{b}{2\sqrt{\pi}}\rho^{-n}n^{-3/2}$.
2. $f(z) = \frac{a}{\sqrt{1-z/\rho}} + o\left(\frac{1}{\sqrt{1-z/\rho}}\right)$, *with $a \in \mathbb{R}$, and $a \neq 0$, then* $[z^n]f(z) \sim \frac{a}{\sqrt{\pi}}\rho^{-n}n^{-1/2}$.

The following straightforward lemma was used though out our analytic computations.

Lemma 1. *If $f(z)$ is an entire function with $\lim_{z\to\rho} f(z) = a$ and $r \in \mathbb{R}$, then*

$$f(z)(1-z/\rho)^r = a(1-z/\rho)^r + o((1-z/\rho)^r).$$

5 The Average Number of State Mergings

5.1 Regular Expressions with ε as a Partial Derivative

Since $\mathcal{A}_{\mathrm{pd}}(\alpha)$ is a quotient of the Glushkov automaton, we know that it has at most $|\alpha|_\Sigma + 1$ states. But this upper bound is reached if and only if, in every step, during the computation of $\pi(\alpha)$, all unions are disjoint. There are however two cases in which this clearly does not happen. Whenever $\varepsilon \in \pi(\beta) \cap \pi(\gamma)$,

$$|\pi(\beta + \gamma)| = |\pi(\beta) \cup \pi(\gamma)| \leq |\pi(\beta)| + |\pi(\gamma)| - 1, \tag{8}$$

and also

$$\begin{aligned} |\pi(\beta\gamma^\star)| = |\pi(\beta) \odot \gamma^\star \cup \pi(\gamma^\star)| &= |\pi(\beta) \odot \gamma^\star \cup \pi(\gamma) \odot \gamma^\star| \\ &\leq |\pi(\beta)| + |\pi(\gamma)| - 1. \end{aligned} \tag{9}$$

In this section we will estimate the number of non-disjoint unions formed during the computation of $\pi(\alpha)$, that are due to either one of the above cases. This corresponds to the merging of states in the Glushkov automaton. Notice that there might be additional mergings resulting from other identical elements in the support of the regular expressions. Therefore our estimation is only a lower bound of the actual number of state mergings, that turns out to be surprisingly tight as shown in Section 6.

Example 3. In order to illustrate the effect of ε being a partial derivative of a subexpression, we consider the marked version of τ, $\overline{\tau} = (a_1 + b_2)(a_3^\star + b_4 a_5^\star + b_6^\star)^\star$. In $\mathcal{A}_{\mathrm{pos}}(\tau)$ each position corresponds to a state. Now, note that, for instance, one has

$$\pi(a_1 + a_2) = \pi(a_1) \cup \pi(a_2) = \{\varepsilon\} \cup \{\varepsilon\} = \{\varepsilon\},$$

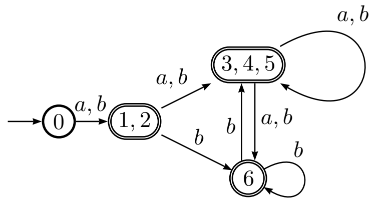
and

$$\pi(b_4 a_5^\star) = \pi(b_4) \odot a_5^\star \cup \pi(a_5) \odot a_5^\star = \{\varepsilon\} \odot a_5^\star \cup \{\varepsilon\} \odot a_5^\star.$$

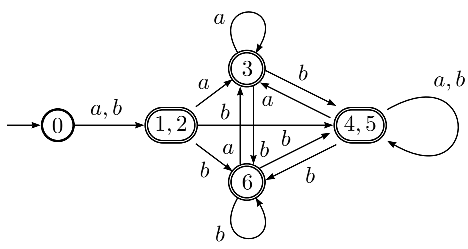
These two cases originate the mergings of states 1 and 2, as well as 4 and 5 of $\mathcal{A}_{\mathrm{pos}}(\tau)$. There is another state merging that is due to neither of the above cases. In fact,

$$\pi(a_3^\star + b_4 a_5^\star) = \pi(a_3) \odot a_3^\star \cup \pi(b_4) \odot a_5^\star = \{\varepsilon\} \odot a_3^\star \cup \{\varepsilon\} \odot a_5^\star.$$

Since the $\mathcal{A}_{\mathrm{pd}}(\tau)$ is computed for the unmarked τ, there is also the merging of states 3 and 4. This is not the case for $\mathcal{A}_{\mathrm{pd}}(\overline{\tau})$, as can be seen in the following diagrams:



$\mathcal{A}_{\mathrm{pd}}(\tau)$



$\mathcal{A}_{\mathrm{pd}}(\overline{\tau})$

As this example suggests, the lower bound for the number of mergings of states that is computed in this paper is precisely the number of mergings that arise when obtaining $\mathcal{A}_{\mathrm{pd}}(\overline{\alpha})$ from $\mathcal{A}_{\mathrm{pos}}(\alpha)$.

From now on, α will denote regular expressions given in (1), but without $\emptyset$, and its generating function, $R_k(z)$ is given by (2). As mentioned, the number of mergings for an expression α depends on the number of subexpressions with ε in its support. We will estimate this number first. The grammar

$$\alpha_\varepsilon := \sigma \in \Sigma \mid \alpha_\varepsilon + \alpha \mid \alpha_{\overline{\varepsilon}} + \alpha_\varepsilon \mid \alpha \cdot \alpha_\varepsilon \mid \alpha_\varepsilon \cdot \varepsilon$$

generates the set of regular expressions for which $\varepsilon \in \pi(\alpha_\varepsilon)$, that is denoted by R_ε. The remaining regular expressions, that are not generated by this grammar, are denoted by $\alpha_{\overline{\varepsilon}}$.
The generating function for R_ε, $R_{\varepsilon,k}(z)$, satisfies

$$\begin{aligned} R_{\varepsilon,k}(z) = kz + zR_{\varepsilon,k}(z)R_k(z) + z\left(R_k(z) - R_{\varepsilon,k}(z)\right)R_{\varepsilon,k}(z) \\ +zR_k(z)R_{\varepsilon,k}(z) + z^2R_{\varepsilon,k}(z), \end{aligned}$$

which is equivalent to

$$zR_{\varepsilon,k}(z)^2 - \left(z^2 + 3zR_k(z) - 1\right)R_{\varepsilon,k}(z) - kz = 0,$$

and from which one gets

$$R_{\varepsilon,k}(z) = \frac{\left(z^2 + 3zR_k(z) - 1\right) + \sqrt{\left(z^2 + 3zR_k(z) - 1\right)^2 + 4kz^2}}{2z}. \tag{10}$$

One has

$$8zR_{\varepsilon,k}(z) = -b(z) - 3\sqrt{\Delta_k(z)} + \sqrt{a_k(z) + 6b(z)\sqrt{\Delta_k(z)} + 9\Delta_k(z)}, \tag{11}$$

with

$$\begin{aligned} a_k(z) &= 16z^4 - 24z^3 + (64k+1)z^2 + 6z + 1 \\ b(z) &= -4z^2 + 3z + 1. \end{aligned} \tag{12}$$

and $\Delta_k(z)$ as in (2). Using the binomial theorem, Lemma 1 and Proposition 3, one gets

$$[z^n]R_{\varepsilon,k}(z) \sim \frac{3}{16\sqrt{\pi}}\left(1 - \frac{b(\rho_k)}{\sqrt{a_k(\rho_k)}}\right)\sqrt{2(1-\rho_k)}\,\rho_k^{-(n+1)}n^{-3/2}. \tag{13}$$

Therefore

$$\frac{[z^n]R_{\varepsilon,k}(z)}{[z^n]R_k(z)} \sim \frac{3}{2}\left(1 - \frac{b(\rho_k)}{\sqrt{a_k(\rho_k)}}\right). \tag{14}$$

Note that $\lim_{k\to\infty} \rho_k = 0$, $\lim_{k\to\infty} a_k(\rho_k) = 9$ and $\lim_{k\to\infty} b(\rho_k) = 1$, and so the asymptotic ratio of regular expressions with ε on their derivatives approaches 1 as $k \to \infty$.

5.2 The Generating Function of Mergings

Let $i(\alpha)$ be the number of non-disjoint unions appearing, due to (8) or (9), during the computation of $\pi(\alpha)$, $\alpha \in \mathsf{R}$. These correspond to state mergings of Glushkov automata. Splitting the regular expressions into the disjoint classes α_ε and $\alpha_{\overline{\varepsilon}}$, $i(\alpha)$ verifies

$$\begin{aligned}
i(\varepsilon) &= 0\\
i(\sigma) &= 0\\
i(\alpha_\varepsilon + \alpha_\varepsilon) &= i(\alpha_\varepsilon) + i(\alpha_\varepsilon) + 1\\
i(\alpha_\varepsilon + \alpha_{\overline{\varepsilon}}) &= i(\alpha_\varepsilon) + i(\alpha_{\overline{\varepsilon}})\\
i(\alpha_{\overline{\varepsilon}} + \alpha) &= i(\alpha_{\overline{\varepsilon}}) + i(\alpha)\\
i(\alpha_\varepsilon \cdot \alpha_\varepsilon^\star) &= i(\alpha_\varepsilon) + i(\alpha_\varepsilon) + 1\\
i(\alpha_\varepsilon \cdot \overline{\alpha_\varepsilon^\star}) &= i(\alpha_\varepsilon) + i(\overline{\alpha_\varepsilon^\star})\\
i(\alpha_{\overline{\varepsilon}} \cdot \alpha) &= i(\alpha_{\overline{\varepsilon}}) + i(\alpha)\\
i(\alpha^\star) &= i(\alpha),
\end{aligned}$$

where $\overline{\alpha_\varepsilon^\star}$ denotes regular expressions that are not of the form $\alpha_\varepsilon^\star$. Clearly, the generating function for these expressions is $R_k(z) - zR_{\varepsilon,k}(z)$.

The cost generating function of the mergings, $I_k(z)$, can now be obtained from these equations by adding the contributions of each single one of them. These contributions can be computed as here exemplified for the contribution of the regular expressions of the form $(\alpha_\varepsilon + \alpha_\varepsilon)$:

$$\begin{aligned}
\sum_{(\alpha_\varepsilon+\alpha_\varepsilon)} i(\alpha_\varepsilon + \alpha_\varepsilon) z^{|(\alpha_\varepsilon+\alpha_\varepsilon)|} &= z\sum_{\alpha_\varepsilon}\sum_{\alpha_\varepsilon}(i(\alpha_\varepsilon) + i(\alpha_\varepsilon) + 1) z^{|\alpha_\varepsilon|} z^{|\alpha_\varepsilon|}\\
&= z\sum_{\alpha_\varepsilon}\sum_{\alpha_\varepsilon}(i(\alpha_\varepsilon) + i(\alpha_\varepsilon)) z^{|\alpha_\varepsilon|} z^{|\alpha_\varepsilon|}\\
&\qquad + z\sum_{\alpha_\varepsilon}\sum_{\alpha_\varepsilon} z^{|\alpha_\varepsilon|} z^{|\alpha_\varepsilon|}\\
&= 2zI_{\varepsilon,k}(z)R_{\varepsilon,k}(z) + zR_{\varepsilon,k}(z)^2,
\end{aligned}$$

where $I_{\varepsilon,k}(z)$ is the generating function for the mergings coming from α_ε.

Applying this technique to the remaining cases, we obtain

$$I_k(z) = \frac{(z + z^2)R_{\varepsilon,k}(z)^2}{\sqrt{\Delta_k(z)}}. \tag{15}$$

The asymptotic value of the coefficients of this generating function can now be computed using (11), and again Lemma 1 and Proposition 3, yielding

$$[z^n]I_k(z) \sim \frac{1+\rho_k}{64}\frac{\left(a_k(\rho_k) + b(\rho_k)^2 - 2b(\rho_k)\sqrt{a_k(\rho_k)}\right)}{\sqrt{\pi}\sqrt{2-2\rho_k}}\rho_k^{-(n+1)}n^{-1/2}. \tag{16}$$

Table 1. Accuracy of the approximation

$n =$	10	20	50	100	200	400
$k = 2$	1.34	1.14	1.05	1.03	1.01	1.01
$k = 3$	1.35	1.12	1.05	1.02	1.01	1.01
$k = 5$	1.38	1.12	1.04	1.02	1.01	1.01
$k = 10$	—	1.13	1.04	1.02	1.01	1.01
$k = 20$	—	—	1.04	1.02	1.01	1.01
$k = 50$	—	—	—	1.02	1.01	1.01
$k = 100$	—	—	—	—	1.01	1.04

Table 1 exhibits the ratio between the approximation given by this computation and the actual coefficients of the power series of $I_k(z)$, for several values of k and n.

From (3) and (16) one easily gets the following asymptotic estimate for the average number of mergings

$$\frac{[z^n]I_k(z)}{[z^n]R_k(z)} \sim \lambda_k\, n, \tag{17}$$

where $\lambda_k = \frac{(1+\rho_k)}{16(1-\rho_k)}\left(a_k(\rho_k) + b(\rho_k)^2 - 2b(\rho_k)\sqrt{a_k(\rho_k)}\right)$. Using again the fact that $\lim_{k\to\infty} \rho_k = 0$, $\lim_{k\to\infty} a_k(\rho_k) = 9$ and $\lim_{k\to\infty} b(\rho_k) = 1$, one gets that

$$\lim_{k\to\infty} \lambda_k = \frac{1}{4}.$$

This means that, for a regular expression of size n, the average number of state mergings is, asymptotically, about $\frac{n}{4}$.

In order to obtain a lower bound for the reduction in the number of states of the $\mathcal{A}_{\text{pd}}$ automaton, as compared to the ones of the $\mathcal{A}_{\text{pos}}$ automaton, it is enough to compare the number of mergings for an expression α with the number of letters in α. From (5) and (17) one gets

$$\frac{[z^n]I_k(z)}{[z^n]L_k(z)} \sim \frac{1-\rho_k}{4k\rho_k^2}\lambda_k. \tag{18}$$

It is easy to see that

$$\lim_{k\to\infty} \frac{1-\rho_k}{4k\rho_k^2}\lambda_k = \frac{1}{2}.$$

In other words, asymptotically, the average number of states of the $\mathcal{A}_{\text{pd}}$ automaton is about one half of the number of states of the $\mathcal{A}_{\text{pos}}$ automaton, and about one quarter of the size of the corresponding regular expression, by (7). As shown in Table 2 the actual values are close to these limits already for small alphabets.

6 Comparison with Experimental Results

In order to compare our estimates with the actual number of states in a $\mathcal{A}_{\text{pd}}$ automaton we ran some experiments. We used the FAdo library [AAA$^+$09, fad09],

Table 2. Experimental results for uniform random generated regular expressions

k	$\|\alpha\|$	$\|\alpha\|_\Sigma$	$\|\mathrm{Pos}_0\|$	$\|\delta_{\mathrm{pos}}\|$	$\|\mathrm{PD}\|$	$\|\delta_{\mathrm{pd}}\|$	$\|\overline{\mathrm{PD}}\|$	$\frac{\|\alpha\|_\Sigma - \|\mathrm{PD}\|}{\|\alpha\|_\Sigma}$	$\frac{\|\alpha\|_\Sigma}{\|\alpha\|}$	$\frac{[z^n]I(z)}{n\times[z^n]R(z)}$	$\frac{[z^n]I(z)}{[z^n]L(z)}$
2	1000	276	277	3345	187	1806	190	**0.323**	0.276	0.084	**0.304**
2	2000	553	554	7405	374	3951	380	**0.324**	0.277		
3	1000	318	319	2997	206	1564	208	**0.352**	0.318	0.107	**0.337**
3	2000	638	639	6561	410	3380	416	**0.357**	0.319		
5	1000	364	365	2663	223	1339	226	**0.387**	0.364	0.135	**0.372**
5	2000	728	729	5535	446	2768	451	**0.387**	0.364		
10	1000	405	406	2203	236	1079	238	**0.417**	0.405	0.168	**0.409**
10	2000	809	810	4616	471	2235	475	**0.418**	0.405		
20	1000	440	441	1842	245	875	246	**0.443**	0.44	0.192	**0.435**
20	2000	880	881	3735	489	1768	492	**0.444**	0.44		
30	1000	453	454	1676	247	796	248	**0.455**	0.453	0.203	**0.447**
30	2000	906	907	3380	496	1603	498	**0.453**	0.453		
50	1000	466	467	1516	250	718	251	**0.464**	0.466	0.214	**0.459**
50	2000	933	934	3065	499	1441	500	**0.465**	0.467		
100	—	—	—	—	—	—	—	—	—	0.225	**0.471**
1000	—	—	—	—	—	—	—	—	—	0.242	**0.491**

that includes algorithms for computing the Glushkov automaton and the partial derivatives automaton corresponding to a given regular expression. For the results to be statistically significant, regular expressions must be uniformly random generated. The FAdo library implements the method described by Mairson [Mai94] for the uniform random generation of context-free languages. The random generator has as input a grammar and the size of the words to be generated. To obtain regular expressions uniformly generated in the size of the syntactic tree (i.e. parentheses not counted), a prefix notation version of the grammar (1) was used. For each size, n, samples of 1000 regular expressions were generated. Table 2 presents the average values obtained for $n \in \{1000, 2000\}$ and $k \in \{2, 3, 5, 10, 20, 30, 50\}$, and the two last columns give the asymptotic ratios obtained in (17) and (18) for the corresponding values of k.

As can be seen from the columns with bold entries, the asymptotic averages obtained with the analytic methods are very close to the values obtained experimentally. In general, even for small values of n, the ratio of the number of states of $\mathcal{A}_{\mathrm{pd}}$ to the number of states of $\mathcal{A}_{\mathrm{pos}}$ coincide (within an error of less than 3%) with our (asymptotic) estimates. These results indicate that occurrences of ε in the set of partial derivatives are the main reason for a smaller number of states in the $\mathcal{A}_{\mathrm{pd}}$ automaton, when compared with the one in the $\mathcal{A}_{\mathrm{pos}}$ automaton. This is confirmed by comparing the column containing the number of states of $\mathcal{A}_{\mathrm{pd}}$ ($|\mathrm{PD}|$) with the one containing those of its marked version ($|\overline{\mathrm{PD}}|$).

7 Final Remarks

In this paper we studied, using analytic methods, the average number of states of partial derivative automata. We proved this number to be, on average, half the number of states when considering the Glushkov automata case. An approach similar to the one applied here can be used to estimate the average number of transitions of $\mathcal{A}_{\text{pd}}$. According to Table 2, this number also seems to be half the number of transitions of $\mathcal{A}_{\text{pos}}$. At first sight, one would expect that the use of alternative grammars for the generation of regular expressions, with less redundancy, such as the ones presented by Lee and Shallit [LS05], would lead to different results. However, experimental studies do not support this expectation, since they do not show significant differences from the results here presented.

References

[AAA$^+$09] Almeida, A., Almeida, M., Alves, J., Moreira, N., Reis, R.: FAdo and GUItar: tools for automata manipulation and visualization. In: Maneth, S. (ed.) CIAA 2009. LNCS, vol. 5642, pp. 65–74. Springer, Heidelberg (2009)

[Ant96] Antimirov, V.M.: Partial derivatives of regular expressions and finite automation constructions. Theoret. Comput. Sci. 155(2), 291–319 (1996)

[Brz64] Brzozowski, J.A.: Derivatives of regular expressions. JACM 11(4), 481–494 (1964)

[CZ01] Champarnaud, J.M., Ziadi, D.: From Mirkin's prebases to Antimirov's word partial derivatives. Fundam. Inform. 45(3), 195–205 (2001)

[CZ02] Champarnaud, J.M., Ziadi, D.: Canonical derivatives, partial derivatives and finite automaton constructions. Theoret. Comput. Sci. 289, 137–163 (2002)

[fad09] FAdo: tools for formal languages manipulation, `http://www.ncc.up.pt/FAdo` (access date: 1.12.2009)

[FS08] Flajolet, P., Sedgewick, R.: Analytic Combinatorics. CUP, Cambridge (2008)

[IY03] Ilie, L., Yu, S.: Follow automata. Inf. Comput. 186(1), 140–162 (2003)

[Koz97] Kozen, D.C.: Automata and Computability. In: Undergrad. Texts in Computer Science. Springer, Heidelberg (1997)

[LS05] Lee, J., Shallit, J.: Enumerating regular expressions and their languages. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 2–22. Springer, Heidelberg (2005)

[Mai94] Mairson, H.G.: Generating words in a context-free language uniformly at random. Information Processing Letters 49, 95–99 (1994)

[Mir66] Mirkin, B.G.: An algorithm for constructing a base in a language of regular expressions. Engineering Cybernetics 5, 51–57 (1966)

[Nic09] Nicaud, C.: On the average size of Glushkov's automata. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 626–637. Springer, Heidelberg (2009)

[Sak09] Sakarovitch, J.: Elements of Automata Theory. CUP, Cambridge (2009)

On the Hybrid Černý-Road Coloring Problem and Hamiltonian Paths*

Arturo Carpi[1] and Flavio D'Alessandro[2,3]

[1] Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, via Vanvitelli 1, 06123 Perugia, Italy
carpi@dmi.unipg.it

[2] Dipartimento di Matematica, Università di Roma "La Sapienza" Piazzale Aldo Moro 2, 00185 Roma, Italy
dalessan@mat.uniroma1.it

[3] Department of Mathematics, Boğaziçi University 34342 Bebek, Istanbul, Turkey

Abstract. The Hybrid Černý-Road coloring problem is investigated for graphs with Hamiltonian paths. We prove that if an aperiodic, strongly connected digraph of costant outdegree with n vertices has an Hamiltonian path, then it admits a synchronizing coloring with a reset word of length $2(n-2)(n-1)+1$. The proof is based upon some new results concerning locally strongly transitive automata.

Keywords: Černý conjecture, road coloring problem, synchronizing automaton, rational series.

1 Introduction

An important concept in Computer Science is that of *synchronizing automaton*. A deterministic automaton is called *synchronizing* if there exists an input-sequence, called *synchronizing* or *reset word*, such that the state attained by the automaton, when this sequence is read, does not depend on the initial state of the automaton itself. Two fundamental problems which have been intensively investigated in the last decades are based upon this concept: the *Černý conjecture* and the *Road coloring problem*.

The Černý conjecture [10] claims that a deterministic synchronizing n-state automaton has a reset word of length not larger than $(n-1)^2$. This conjecture has been shown to be true for several classes of automata (*cf.* [2,3,4,8,9,10,12,14,15,16,17,18,20,23]). The interested reader is refered to [23] for a historical survey of the Černý conjecture and to [7] for synchronizing unambiguous automata. In this theoretical setting, two results recently proven in [9] and [4] respectively, are relevant to us.

* This work was partially supported by MIUR project "Aspetti matematici e applicazioni emergenti degli automi e dei linguaggi formali" and by fundings "Facoltà di Scienze MM. FF. NN. 2008" of the University of Rome "La Sapienza".

In [9], the authors have introduced the notion of *local strong transitivity*. An n-state automaton $\mathcal{A}$ is said to be *locally strongly transitive* if it is equipped by a set W of k words and a set R of k distinct states such that, for all states s of $\mathcal{A}$ and all $r \in R$, there exists a word $w \in W$ taking the state s into r. The set W is called *independent* while R is called the *range* of W. The main result of [9] is that any synchronizing locally strongly transitive n-state automaton has a reset word of length not larger than $(k-1)(n+L)+\ell$, where k is the cardinality of an independent set W and L and ℓ denote respectively the maximal and the minimal length of the words of W.

In the case where all the states of the automaton are in the range, the automaton $\mathcal{A}$ is said to be *strongly transitive*. Strongly transitive automata have been studied in [8]. This notion is related with that of regular automata introduced in [18].

A remarkable example of locally strongly transitive automata is that of *1-cluster automata* introduced in [4]. An automaton is called 1-cluster if there exists a letter a such that the graph of the automaton has a unique cycle labelled by a power of a. Indeed, denoting by k the length of the cycle, one easily verifies that the words

$$a^{n-1}, a^{n-2}, \ldots, a^{n-k}$$

form an independent set of the automaton whose range is the set of vertices of the cycle. In [4] it is proven that every 1-cluster synchronizing n-state automaton has a reset word of length not larger than $2(n-1)(n-2)+1$.

The second problem we have mentioned above is the well-known *Road coloring problem*. This problem asks to determine whether any aperiodic and strongly connected digraph, with all vertices of the same outdegree (*AGW-graph*, for short) has a *synchronizing coloring*, that is, a labeling of its edges that turns it into a synchronizing deterministic automaton. The problem was formulated in the context of Symbolic Dynamics by Adler, Goodwyn and Weiss and it is explicitly stated in [1]. In 2007, Trahtman [21] has positively solved it. The solution by Trahtman has electrified the community of formal language theories and recently Volkov has raised in [22] the problem of evaluating, for any AGW-graph G, the minimal length of a reset word for a synchronizing coloring of G. This problem has been called *the Hybrid Černý–Road coloring problem.* It is worth to mention that Ananichev has found, for any $n \geq 2$, an AGW-graph of n vertices such that the length of the shortest reset word for any synchronizing coloring of the graph is $(n-1)(n-2)+1$ (see [22]). In [9], the authors have proven that, given an AGW-graph G of n vertices, without multiple edges, such that G has a simple cycle of prime length $p < n$, there exists a synchronizing coloring of G with a reset word of length $(2p-1)(n-1)$. Moreover, in the case $p = 2$, that is, if G contains a cycle of length 2, then, also in presence of multiple edges, there exists a synchronizing coloring with a reset word of length $5(n-1)$.

In this paper, we continue the investigation of the Hybrid Černý–Road coloring problem on a very natural class of digraphs, those having a *Hamiltonian path*. The main result of this paper states that any AGW-graph G of n vertices with a Hamiltonian path admits a synchronizing coloring with a reset word of length

$$2(n-1)(n-2)+1. \tag{1}$$

The proof of the theorem above is based upon some techniques on independent sets of words and on some results on synchronizing colorings of graphs.

A set K of states of an automaton $\mathcal{A}$ is *reducible* if there exists a word $w \in A^*$ taking all the states of K into a fixed state. The least congruence ρ of $\mathcal{A}$ such that any congruence class is reducible is called *stability*. This congruence, introduced in [11], plays a fundamental role in the solution [21] of the Road coloring problem. Even if $\mathcal{A}$ is not a synchronizing automaton, it is natural to ask for the minimal length of a word w taking all the states of a given stability class K into a single state. In Section 3, we prove that if $\mathcal{A}$ is a locally strongly transitive n-state automaton which is not synchronizing, then the minimal length of such a word w is at most

$$\left(\frac{k}{2}-1\right)(n+L-1)+L,$$

where k is the cardinality of any independent set W and L denotes the maximal length of the words of W. In the case where $\mathcal{A}$ is synchronizing, we obtain for the minimal length of a reset word of $\mathcal{A}$ the upper bound

$$(k-1)(n+L-1)+\ell \tag{2}$$

where k and L are defined as before and ℓ denotes the minimal length of the words of W. This bound refines the quoted bound of [9].

We close the introduction with the following remark. By using a more sophisticated and laborious technique similar to that of [19], the bound (2) can be lowered to $(k-1)(n+L-2)+\ell$. Moreover, a recent improvement [5] of the bound obtained in [4] should allow to slightly refine the bound (1).

2 Preliminaries

We assume that the reader is familiar with the theory of automata and rational series. In this section we shortly recall a vocabulary of few terms and we fix the corresponding notation used in the paper.

Let A be a finite alphabet and let A^* be the free monoid of words over the alphabet A. The identity of A^* is called the *empty word* and is denoted by ϵ. The *length* of a word $w \in A^*$ is the integer $|w|$ inductively defined by $|\epsilon| = 0$, $|wa| = |w| + 1$, $w \in A^*$, $a \in A$. For any positive integer n, we denote by $A^{<n}$ the set of words of length smaller than n.

For any finite set of words, W, we denote respectively by L_W and ℓ_W the maximal and minimal lengths of the words of W.

A finite automaton is a triple $\mathcal{A} = \langle Q, A, \delta \rangle$ where Q is a finite set of elements called *states* and δ is a map

$$\delta : Q \times A \longrightarrow Q.$$

The map δ is called the *transition function* of $\mathcal{A}$. The canonical extension of the map δ to the set $Q \times A^*$ is still denoted by δ.

If P is a subset of Q and u is a word of A^*, we denote by $\delta(P, u)$ and $\delta(P, u^{-1})$ the sets:

$$\delta(P, u) = \{\delta(s, u) \mid s \in P\}, \quad \delta(P, u^{-1}) = \{s \in Q \mid \delta(s, u) \in P\}.$$

With any automaton $\mathcal{A} = \langle Q, A, \delta \rangle$, we can associate a directed multigraph $G = (Q, E)$, where the multiplicity of the edge $(p, q) \in Q \times Q$ is given by $\mathrm{Card}(\{a \in A \mid \delta(p, a) = q\}$. If the automaton $\mathcal{A}$ is associated with G, we also say that $\mathcal{A}$ is a *coloring* of G. An automaton is *transitive* if the associated graph is strongly connected. If $n = \mathrm{Card}(Q)$, we will say that $\mathcal{A}$ is a n-state automaton. A *synchronizing* or *reset* word of $\mathcal{A}$ is any word $u \in A^*$ such that $\mathrm{Card}(\delta(Q, u)) = 1$. A *synchronizing* automaton is an automaton that has a reset word. The following conjecture has been raised in [10].

Černý Conjecture. *Each synchronizing n-state automaton has a reset word of length not larger than $(n-1)^2$.*

Let $\mathbb{K}$ be a field. We recall that a *formal power series* with coefficients in $\mathbb{K}$ and non-commuting variables in A is a mapping of the free monoid A^* into $\mathbb{K}$. According to Kleene-Schützenberger theorem on formal power series (see [6]), a series $\mathcal{S} : A^* \to \mathbb{K}$ is *rational* if there exists a triple (α, μ, β) where

- $\alpha \in \mathbb{K}^{1\times n}$, $\beta \in \mathbb{K}^{n\times 1}$ are a row vector and a column vector respectively,
- $\mu : A^* \to \mathbb{K}^{n\times n}$ is a morphism of the free monoid A^* in the multiplicative monoid $\mathbb{K}^{n\times n}$ of matrices with coefficients in $\mathbb{K}$,
- for every $u \in A^*$, $\mathcal{S}(u) = \alpha\mu(u)\beta$.

The triple (α, μ, β) is called a *linear representation* of $\mathcal{S}$ and the integer n is called its *dimension.* The minimal dimension of a linear representation of $\mathcal{S}$ is called the *dimension* of $\mathcal{S}$. Let $\mathcal{A} = \langle Q, A, \delta \rangle$ be any n-state automaton. One can associate with $\mathcal{A}$ a morphism

$$\varphi_{\mathcal{A}} : A^* \to \mathbb{Q}^{Q\times Q},$$

of the free monoid A^* in the multiplicative monoid $\mathbb{Q}^{Q\times Q}$ of matrices over the field $\mathbb{Q}$ of rational numbers, defined as: for any $u \in A^*$ and for any $s, t \in Q$,

$$\varphi_{\mathcal{A}}(u)_{st} = \begin{cases} 1 & \text{if } t = \delta(s, u) \\ 0 & \text{otherwise.} \end{cases}$$

Let R and K be subsets of Q and consider the rational series $\mathcal{S}$ with rational coefficients having the linear representation $(\alpha, \varphi_{\mathcal{A}}, \beta)$, where, for every $s \in Q$,

$$\alpha_s = \begin{cases} 1 & \text{if } s \in R, \\ 0 & \text{otherwise,} \end{cases} \qquad \beta_s = \begin{cases} 1 & \text{if } s \in K, \\ 0 & \text{otherwise.} \end{cases}$$

It is easily seen that, for any $u \in A^*$, one has

$$\mathcal{S}(u) = \mathrm{Card}(\delta(K, u^{-1}) \cap R). \tag{3}$$

We say that a linear representation (α, μ, β) is *uniform* if there exists a vector $\gamma \in \mathbb{K}^{n\times 1}$ such that $\mu(a)\gamma = \gamma$ for all $a \in A$. For instance, the linear representation $(\alpha, \varphi_{\mathcal{A}}, \beta)$ above is uniform, since $\varphi_{\mathcal{A}}(a)\gamma = \gamma$ for all $a \in A$, with $\gamma = {}^{\mathrm{t}}(1\,1\,\cdots\,1)$.

The following result refines a fundamental theorem by Moore and Conway on automata equivalence (see [6,13]) in the case of uniform linear representations. The proof is inspired to some ideas of [3,4].

Theorem 1. *Let $\mathcal{S}_1$, $\mathcal{S}_2 : A^* \to \mathbb{K}$ be two rational series with coefficients in $\mathbb{K}$, having uniform linear representations of dimension n_1 and n_2 respectively. If, for every $u \in A^*$ such that $|u| \le n_1 + n_2 - 2$, $\mathcal{S}_1(u) = \mathcal{S}_2(u)$, the series $\mathcal{S}_1$ and $\mathcal{S}_2$ are equal.*

Proof. It is sufficient to verify that the series $\mathcal{S} = \mathcal{S}_1 - \mathcal{S}_2$ has a linear representation of dimension $n = n_1 + n_2 - 1$. Indeed, in such a case, since $\mathcal{S}(u) = 0$ for all $u \in A^{<n}$, a classical result on rational formal power series (see [6]) ensures that $\mathcal{S}$ is the null series, that is, $\mathcal{S}_1 = \mathcal{S}_2$.

Let $(\alpha_i, \mu_i, \beta_i)$ be the uniform linear representations of $\mathcal{S}_i$ of dimension n_i, $i = 1, 2$. Then the series $\mathcal{S} = \mathcal{S}_1 - \mathcal{S}_2$ has the linear representation (α, μ, β) with

$$\alpha = (\alpha_1\ \alpha_2), \quad \beta = \begin{pmatrix} \beta_1 \\ -\beta_2 \end{pmatrix}, \quad \mu(a) = \begin{pmatrix} \mu_1(a) & 0 \\ 0 & \mu_2(a) \end{pmatrix}, \quad a \in A\,.$$

By a well known result on rational power series (see [6]), the dimension of $\mathcal{S}$ is upper bounded by the linear dimension of the space generated by the vectors $\alpha\mu(w)$, $w \in A^*$.

The matrices $\mu(a)$ have two common right eigenvectors $(\gamma_1\ 0)$ and $(0\ \gamma_2)$ both associated with the eigenvalue 1. Among the linear combinations of these eigenvectors, there is an eigenvector γ orthogonal to α. One obtains

$$\alpha\mu(w)\gamma = \alpha\gamma = 0\,, \quad w \in A^*\,.$$

This equation shows that the vectors $\alpha\mu(w)$ lie in a proper subspace of $\mathbb{K}^{n+1}$. Consequently, the dimension of $\mathcal{S}$ is not larger than n. The conclusion follows. $\square$

We end this section by introducing the important notion of stability [11]. Let $\mathcal{A} = \langle Q, A, \delta\rangle$ be an automaton. Given two states p, q of $\mathcal{A}$, we say that the pair (p, q) is *stable* if, for all $u \in A^*$, there exists $v \in A^*$ such that $\delta(p, uv) = \delta(q, uv)$. The set ρ of stable pairs is a congruence of the automaton $\mathcal{A}$, which is called *stability relation*. It is easily seen that an automaton is synchronizing if and only if the stability relation is the universal equivalence.

3 Independent Systems of Words

In this section, we will present some results that can be obtained by using some techniques on independent systems of words. We begin by recalling a definition introduced in [9].

Definition 1. *Let $\mathcal{A} = \langle Q, A, \delta \rangle$ be an automaton. A set of k words $W = \{w_0, \ldots, w_{k-1}\}$ is called* independent *if there exist k distinct states $q_0, \ldots, q_{k-1}$ of $\mathcal{A}$ such that, for all $s \in Q$,*

$$\{\delta(s, w_0), \ldots, \delta(s, w_{k-1})\} = \{q_0, \ldots, q_{k-1}\}.$$

The set $R = \{q_0, \ldots, q_{k-1}\}$ will be called the range *of W.*

An automaton is called *locally strongly transitive* if it has an independent set of words. The following useful properties can be derived from Definition 1 (see [9], Section 3).

Lemma 1. *Let $\mathcal{A}$ be an automaton and let W be an independent set of $\mathcal{A}$ with range R. Then, for every $u \in A^*$, the set uW is an independent set of $\mathcal{A}$ with range R.*

Proposition 1. *Let $\mathcal{A} = \langle Q, A, \delta \rangle$ be an automaton and consider an independent set W of $\mathcal{A}$ with range R. Then, for every subset P of R,*

$$\sum_{w \in W} \mathrm{Card}(\delta(P, w^{-1}) \cap R) = \mathrm{Card}(W)\, \mathrm{Card}(P).$$

A remarkable example of locally transitive automata is that of *1-cluster automata*, recently investigated in [4]. A n-state automaton is called 1-cluster if there exists a letter a such that the graph of the automaton has a unique cycle labelled by a power of a. Indeed, denoting by k the length of the cycle, one easily verifies that the words

$$a^{n-1}, a^{n-2}, \ldots, a^{n-k}$$

form an independent set of the automaton whose range is the set of vertices of the cycle. We recall that the following result has been proven in [4].

Theorem 2. *Let $\mathcal{A}$ be a synchronizing n-state automaton. If $\mathcal{A}$ is 1-cluster, then it has a reset word of length $1 + 2(n-1)(n-2)$.*

Let $\mathcal{A} = \langle Q, A, \delta \rangle$ be a n-state automaton. We say that a set of states K of $\mathcal{A}$ is *reducible* if, for some word w, $\delta(K, w)$ is a singleton. A set $K \subseteq Q$ is *stable* if for any $p, q \in K$, the pair (p, q) is stable. Any stable set K is reducible. Thus, even if $\mathcal{A}$ is not synchronizing, one may want to evaluate the minimal length of a word w such that $\mathrm{Card}(\delta(K, w)) = 1$.

In the sequel, we assume that W is an independent set of $\mathcal{A}$ with range R and denote by M the maximal cardinality of reducible subsets of R. Moreover, for any set of states K and any $w \in A^*$, we denote by Kw^{-1} the set $\delta(K, w^{-1})$. The following lemma holds.

Lemma 2. *Let K be a non-empty reducible subset of R. The following conditions are equivalent:*

1. $\mathrm{Card}(K) = M$,
2. *for all $w \in W$, $v \in A^{<n}$,* $\mathrm{Card}(K(vw)^{-1} \cap R) \leq \mathrm{Card}(K)$,

3. for all $w \in W$, $v \in A^{<n}$, $\mathrm{Card}(K(vw)^{-1} \cap R) = \mathrm{Card}(K)$,
4. for all $w \in W$, $v \in A^{*}$, $\mathrm{Card}(K(vw)^{-1} \cap R) = \mathrm{Card}(K)$.

Proof. Implication 1. ⇒ 2. is trivial, since $K(vw)^{-1} \cap R$ is reducible. Let us verify 2. ⇒ 3. First we recall that, by Lemma 1, for any $v \in A^*$, the set vW is independent. By Proposition 1, one has:

$$\sum_{w \in W} \mathrm{Card}(K(vw)^{-1} \cap R) = \mathrm{Card}(W)\,\mathrm{Card}(K).$$

In view of Condition 2, one obtains Condition 3.

Now let us prove 3. ⇒ 4. Consider the series $\mathcal{S}_1$, $\mathcal{S}_2$ defined respectively by

$$\mathcal{S}_1(v) = \mathrm{Card}((Kw^{-1})v^{-1} \cap R), \quad \mathcal{S}_2(v) = \mathrm{Card}(K), \quad v \in A^* .$$

In view of (3), $\mathcal{S}_1$ has a uniform linear representation of dimension n. Moreover, $\mathcal{S}_2$ has a uniform linear representation of dimension 1. By Condition 3, $\mathcal{S}_1(v) = \mathcal{S}_2(v)$ for all $v \in A^*$ such that $|v| < n$. By Theorem 1, it follows that $\mathcal{S}_1 = \mathcal{S}_2$. Thus Condition 4 holds true.

Finally, let us prove implication 4. ⇒ 1. Let X be a reducible subset of R with $\mathrm{Card}(X) = M$. One has $\delta(X, v) = \{q\}$ and $\delta(q, w) \in K$ for some $q \in Q$, $w \in W$. Hence, $X \subseteq K(vw)^{-1} \cap R$ so that $\mathrm{Card}(K) = \mathrm{Card}(K(vw)^{-1} \cap R) \geq M$. □

Lemma 3. *There exist* $K \subseteq R$ *and* $v \in A^*$ *such that*

$$\mathrm{Card}(K) = M\,, \quad \mathrm{Card}(\delta(K, v)) = 1\,, \quad |v| \leq (M-1)(n + L_W - 1)\,.$$

Proof. Using Condition 2 of Lemma 2, one can prove the following claim by induction on m:

For $1 \leq m \leq M$, there exist $K \subseteq R$ and $v \in A^*$ such that $\mathrm{Card}(K) \geq m$, $\mathrm{Card}(\delta(K, v)) = 1$, $|v| \leq (m-1)(n + L_W - 1)$. □

Lemma 4. *Let* K *be a reducible subset of* R *of maximal cardinality. There is no stable pair in* $K \times (R \setminus K)$.

Proof. By contradiction, let $(p, q) \in K \times (R \setminus K)$ be a stable pair. Then, $\delta(K, v) = \{\delta(p, v)\}$ and $\delta(p, vu) = \delta(q, vu) = s$, $s \in Q$ for some $u, v \in A^*$. Thus $\delta(K \cup \{q\}, vu) = \{s\}$, contradicting the maximality of K. □

Proposition 2. *For any stable set* C *there exists a word* v *such that*

$$\mathrm{Card}(\delta(C, v)) = 1\,, \quad |v| \leq (M-1)(n + L_W - 1) + L_W\,.$$

Proof. By Lemma 3, there exist $K \subseteq R$ and $u \in A^*$ such that $\mathrm{Card}(K) = M$, $\mathrm{Card}(\delta(K, u)) = 1$, $|u| \leq (M-1)(n+L_W-1)$. Since W is an independent set with range R, there is $w \in W$ such that $\delta(C, w) \cap K \neq \emptyset$. Moreover, $\delta(C, w)$ is a stable subset of R. By Lemma 4, one derives $\delta(C, w) \subseteq K$, so that $\mathrm{Card}(\delta(C, wu)) = \mathrm{Card}(\delta(K, u)) = 1$. The statement is thus verified for $v = wu$. □

If $\mathcal{A}$ is synchronizing, then Q itself is a stable set. Thus, with some minor changes in the proof of Proposition 2, one obtains the following result which refines the bound of [8].

Proposition 3. *Any synchronizing n-state automaton with an independent set W has a reset word of length*

$$(\mathrm{Card}(W) - 1)(n + L_W - 1) + \ell_W \, .$$

Corollary 1. *If $\mathcal{A}$ is not synchronizing, then for any stable set C there exists a word v such that*

$$\mathrm{Card}(\delta(C, v)) = 1 \, , \quad |v| \leq \left(\frac{\mathrm{Card}(W)}{2} - 1\right)(n + L_W - 1) + L_W \, .$$

Proof. There are $K \subseteq R$, $v \in A^*$, $w \in W$, $q \in Q$ such that $\mathrm{Card}(K) = M$, $\delta(K, v) = \{q\}$, $\delta(q, w) \notin K$. In view of Lemma 2, K and $K(vw)^{-1} \cap R$ are disjoint subsets of R of cardinality M. One derives $M \leq \mathrm{Card}(W)/2$. The conclusion follows from Proposition 2. □

Corollary 2. *Let C be a stable set of a 1-cluster n-state automaton which is not synchronizing. There exists a word v such that* $\mathrm{Card}(\delta(C, v)) = 1$ *and* $|v| \leq (n - 1)^2$.

Proof. Any 1-cluster n-state automaton has an independent set W with $L_W = n - 1$. Taking into account that $\mathrm{Card}(W) \leq n$, the statement follows from Corollary 1. □

4 The Hybrid Černý-Road Coloring Problem

In the sequel, with the word graph, we will term a finite, directed multigraph with all vertices of the same outdegree. A graph is *aperiodic* if the greatest common divisor of the lengths of all cycles of the graph is 1. A graph is called an *AGW-graph* if it is aperiodic and strongly connected. A synchronizing automaton which is a coloring of a graph G will be called a *synchronizing coloring* of G. The *Road coloring problem* asks for the existence of a synchronizing coloring for every AGW-graph. This problem was formulated in the context of Symbolic Dynamics by Adler, Goodwyn and Weiss and it is explicitly stated in [1]. In 2007, Trahtman has positively solved this problem [21]. Recently Volkov has raised the following problem [22].

Hybrid Černý–Road coloring problem. *Let G be an AGW-graph. What is the minimum length of a reset word for a synchronizing coloring of G?*

4.1 Relabeling

In order to prove our main theorem, we need to recall some basic results concerning colorings of graphs. Let $\mathcal{A} = \langle Q, A, \delta \rangle$ be an automaton. A map

$\delta' : Q \times A \longrightarrow Q$ is a *relabeling* of $\mathcal{A}$ if, for each $q \in Q$, there exists a permutation π_q of A such that

$$\delta'(q,a) = \delta(q, \pi_q(a)),\ a \in A\,.$$

It is clear that δ' is a relabeling of $\mathcal{A}$ if and only if the automata $\mathcal{A}$ and $\mathcal{A}' = \langle Q, A, \delta' \rangle$ are associated with the same graph.

Let $\mathcal{A} = \langle Q, A, \delta \rangle$ be an automaton, α be a congruence on Q and δ' be a relabeling of $\mathcal{A}$. According to [11], δ' *respects* α if for each congruence class C there exists a permutation π_C of A such that

$$\delta'(q,a) = \delta(q, \pi_C(a))\,, \quad q \in C\,,\ a \in A\,.$$

In such a case, for all $v \in A^*$ there is a word $u \in A^*$ such that $|u| = |v|$ and $\delta'(q,u) = \delta(q,v)$ for all $q \in C$.

As α is a congruence, we may consider the quotient automaton $\mathcal{A}/\alpha$. Any relabeling $\widehat{\delta}$ of $\mathcal{A}/\alpha$ induces a relabeling δ' of $\mathcal{A}$ which respects α. This means that

1. α is a congruence of $\mathcal{A}' = \langle Q, A, \delta' \rangle$ and $\mathcal{A}'/\alpha = \langle Q/\alpha, A, \widehat{\delta} \rangle$,
2. for all α-class C and all $v \in A^*$, there exists $u \in A^*$ such that $|v| = |u|$ and $\delta'(C,u) = \delta(C,v)$.

We end this section by recalling the following important result proven in [11].

Proposition 4. *Let ρ be the stability congruence of an automaton $\mathcal{A}$ associated with an AGW-graph G. Then the graph G' associated with the quotient automaton $\mathcal{A}/\rho$ is an AGW-graph. Moreover, if G' has a synchronizing coloring, then G has a synchronizing coloring as well.*

4.2 Hamiltonian Paths

In this section we give a partial answer to the Hybrid Černý–Road coloring problem. Precisely we prove that an AGW-graph of n vertices with a Hamiltonian path admits a synchronizing coloring with a reset word of length not larger that $2(n-2)(n-1)+1$. In order to prove this result, we need to establish some properties concerning automata with a monochromatic Hamiltonian path.

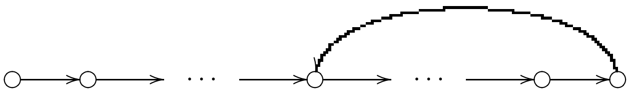
Let a be a letter. The graph of a-transitions of an automaton $\mathcal{A}$ consists of disjoint cycles and trees with root on the cycles. The *level* of a vertex in such a graph is its height in the tree to which it belongs. The following proposition was implicitly proved in [21, Theorem 3].

Proposition 5. *If in the graph of a-transitions of a transitive automaton $\mathcal{A}$ all the vertices of maximal positive level belong to the same tree, then $\mathcal{A}$ has a stable pair.*

As an application of the previous proposition, we obtain the following.

Proposition 6. *If an AGW-graph G with at least 2 vertices has a Hamiltonian path, then there is a coloring of G with a stable pair and a monochromatic Hamiltonian path.*

Proof. Let G be an AGW-graph with $n \geq 2$ vertices. Let us show that one can find in G a Hamiltonian path $(q_0, q_1, \ldots, q_{n-1})$ and an edge (q_{n-1}, q) with $q \neq q_0$ (see fig.).



Indeed, if G has no Hamiltonian cycle, it is sufficient to take a Hamiltonian path $(q_0, q_1, \ldots, q_{n-1})$ and any edge outgoing from q_{n-1}: such an edge exists because G has positive constant outdegree.

On the contrary, suppose that G has a Hamiltonian cycle $(q_0, q_1, \ldots, q_{n-1}, q_0)$. Since G is aperiodic, there is an edge (p, q) of G which does not belong to the cycle. We may assume, with no loss of generality, $p = q_{n-1}$, so that $q \neq q_0$. Thus, $(q_0, q_1, \ldots, q_{n-1})$ is a Hamiltonian path and (q_{n-1}, q) is an edge of G.

Choose a coloring $\mathcal{A}$ of G where the edges of the path $(q_0, q_1, \ldots, q_{n-1}, q)$ are labeled by the same letter a. In such a way, there is a monochromatic Hamiltonian path. Moreover, the graph of a-transitions has a unique tree, so that, by Proposition 5, $\mathcal{A}$ has a stable pair. □

Lemma 5. *If an automaton $\mathcal{A}$ has a monochromatic Hamiltonian path, then any quotient automaton of $\mathcal{A}$ has the same property.*

Proof. With no loss of generality, we may reduce ourselves to the case that $\mathcal{A}$ is a 1-letter automaton. Now, a 1-letter automaton has a Hamiltonian path if and only if it has a state q from which all states are accessible. The conclusion follows from the fact that the latter property is inherited by the quotient automaton. □

We are ready to prove our main result.

Theorem 3. *Let G be an AGW-graph with $n > 1$ vertices. If G has a Hamiltonian path, then there is a synchronizing coloring of G with a reset word w of length*

$$|w| \leq 2(n-2)(n-1) + 1\,. \tag{4}$$

Proof. The proof is by induction on the number n of the vertices of G.

Let $n = 2$. Since G is aperiodic, G has an edge (q, q) which immediatly implies the statement. Suppose $n \geq 3$. By Proposition 6, among the colorings of G, there is an automaton $\mathcal{A} = \langle Q, A, \delta \rangle$ with a stable pair and a monochromatic Hamiltonian path. In particular, $\mathcal{A}$ is a transitive 1-cluster automaton. If $\mathcal{A}$ is synchronizing, then the statement follows from Theorem 2. Thus, we assume that $\mathcal{A}$ is not synchronizing.

Let ρ be the stability congruence of $\mathcal{A}$, k be its index and G_ρ be the graph of $\mathcal{A}/\rho$ respectively. Since $\mathcal{A}$ is not synchronizing, one has $k > 1$. By Proposition 4

G_ρ is an AGW-graph with k vertices and $k < n$. Moreover, by Lemma 5, G_ρ has a Hamiltonian path. By the induction hypothesis, we may assume that there is a relabeling $\widehat{\delta}$ of $\mathcal{A}/\rho$ such that the automaton $\widehat{\mathcal{A}} = \langle Q/\rho, A, \widehat{\delta}\rangle$ has a reset word u such that

$$|u| \leq 2(k-2)(k-1) + 1.$$

As viewed in Section 4.1, $\widehat{\delta}$ induces a relabeling δ' of $\mathcal{A}$ which respects ρ. Moreover, since u is a reset word of $\widehat{\mathcal{A}}$, $C = \delta'(Q, u)$ is a stable set of $\mathcal{A}$.

First, we consider the case $n \geq 2k$. By Corollary 2, there is a word v such that $|v| \leq (n-1)^2$ and $\mathrm{Card}(\delta(C, v)) = 1$. Since δ' respects ρ, there is a word v' such that $|v'| = |v|$ and $\delta'(C, v') = \delta(C, v)$. Set $w = uv'$. Then $\delta'(Q, w) = \delta'(Q, uv') = \delta'(C, v') = \delta(C, v)$ is reduced to a singleton. Hence, w is a reset word of $\mathcal{A}' = \langle Q, A, \delta'\rangle$ and

$$|w| \leq 2(k-2)(k-1) + (n-1)^2 + 1.$$

As $n \geq 2k$, one easily obtains (4).

Now, we consider the case $n < 2k$. In such a case, there is a ρ-class K of cardinality 1. Moreover, by the transitivity of $\widehat{\mathcal{A}}$, there is a word $v \in A^*$ such that $\delta'(C, v) = K$ and $|v| \leq k - 1$. Hence, $w = uv$ is a reset word of $\mathcal{A}'$ of length

$$|w| \leq 2(k-2)(k-1) + k.$$

As $n > k$, one easily obtains (4). This concludes the proof. □

References

1. Adler, R.L., Goodwyn, L.W., Weiss, B.: Equivalence of topological Markov shifts. Israel J. Math. 27, 49–63 (1977)
2. Ananichev, D.S., Volkov, M.V.: Synchronizing generalized monotonic automata. Theoret. Comput. Sci. 330(1), 3–13 (2005)
3. Béal, M.-P.: A note on Černý's Conjecture and rational series, technical report, Institut Gaspard Monge, Université de Marne-la-Vallée (2003)
4. Béal, M.-P., Perrin, D.: A quadratic upper bound on the size of a synchronizing word in one-cluster automata. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 81–90. Springer, Heidelberg (2009)
5. Béal, M.-P., Berlinkov, M.V., Perrin, D.: A quadratic upper bound on the size of a synchronizing word in one-cluster automata. International Journal of Foundations of Computer Science (to appear 2010)
6. Berstel, J., Reutenauer, C.: Rational series and their languages. Springer, Heidelberg (1988)
7. Carpi, A.: On synchronizing unambiguous automata. Theoret. Comput. Sci. 60, 285–296 (1988)
8. Carpi, A., D'Alessandro, F.: Strongly transitive automata and the Černý conjecture. Acta Informatica 46, 591–607 (2009)
9. Carpi, A., D'Alessandro, F.: The synchronization problem for locally strongly transitive automata. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 211–222. Springer, Heidelberg (2009)

10. Černý, J.: Poznámka k. homogénnym experimenton s konečnými automatmi. Mat. fyz. cas SAV 14, 208–215 (1964)
11. Culik II, K., Karhumäki, J., Kari, J.: A note on synchronized automata and road coloring problem. Internat. J. Found. Comput. Sci. 13, 459–471 (2002)
12. Dubuc, L.: Sur les automates circulaires et la conjecture de Cerny. RAIRO Inform. Théor. Appl. 32, 21–34 (1998)
13. Eilenberg, S.: Automata, Languages and Machines, vol. A. Academic Press, London (1974)
14. Frankl, P.: An extremal problem for two families of sets. Eur. J. Comb. 3, 125–127 (1982)
15. Kari, J.: Synchronizing finite automata on Eulerian digraphs. Theoret. Comput. Sci. 295, 223–232 (2003)
16. Pin, J.E.: Le problème de la synchronization et la conjecture de Cerny, Thèse de 3ème cycle, Université de Paris 6 (1978)
17. Pin, J.E.: Sur un cas particulier de la conjecture de Cerny. In: Ausiello, G., Böhm, C. (eds.) ICALP 1978. LNCS, vol. 62, pp. 345–352. Springer, Heidelberg (1978)
18. Rystov, I.: Almost optimal bound of recurrent word length for regular automata. Cybern. Syst. Anal. 31(5), 669–674 (1995)
19. Steinberg, B.: The averaging trick and the Černý conjecture, Preprint arXiv: 0910.0410v2 (2010)
20. Trahtman, A.N.: The Cerny conjecture for aperiodic automata. Discrete Math. and Theor. Comput. Sci. 9(2), 3–10 (2007)
21. Trahtman, A.N.: The road coloring problem. Israel J. Math. 172, 51–60 (2009)
22. Volkov, M.V.: Communication. In: Around the Černý conjecture, International Workshop, University of Wroclaw, Poland (June 2008)
23. Volkov, M.V.: Synchronizing automata and the Cerny conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)

Computing Blocker Sets for the Regular Post Embedding Problem[⋆]

Pierre Chambart and Philippe Schnoebelen

LSV, ENS Cachan, CNRS
61, av. Pdt. Wilson, F-94230 Cachan, France
{chambart,phs}@lsv.ens-cachan.fr

Abstract. Blocker and coblocker sets are regular languages involved in the algorithmic solution of the Regular Post Embedding Problem. We investigate the computability of these languages and related decision problems.

1 Introduction

Post's Embedding Problem (shortly PEP, named by analogy with Post's Correspondence Problem) is the question whether two morphisms on words $u,v:\Sigma^*\to\Gamma^*$ agree non-trivially on some input, i.e., whether $u(x)$ is a (scattered) subword of $v(x)$ for some $x\in\Sigma^+$. The *subword* ordering, also called *embedding*, is denoted "$\sqsubseteq$": $x\sqsubseteq y \overset{\text{def}}{\Leftrightarrow} x$ can be obtained from y by erasing some letters, possibly all of them, possibly none.

PEP is trivial if there are no restrictions on the form of solutions. But when one looks for solutions x as above belonging to a regular language $R\subseteq\Sigma^*$, the problem (hereafter called the *Regular* Post Embedding Problem, or PEP^{reg}) becomes very interesting: decidable but surprisingly hard [1].

The Regular Post Embedding Problem was introduced in [1,2] where it is shown that PEP^{reg} is expressive enough to encode problems on lossy channel systems. In fact, encodings in both directions exist, hence PEP^{reg} is exactly at level $\mathcal{F}_{\omega^\omega}$ in the Fast Growing Hierarchy. Thus, although it is decidable, PEP^{reg} is not primitive-recursive, and not even multiply-recursive (see [3] and the references therein). Finally, PEP^{reg} is an abstract problem that is inter-reducible with a growing list of decidable problems having the same $\mathcal{F}_{\omega^\omega}$ complexity: metric temporal logic [4], alternating one-clock timed automata [5,6], leftist grammars [7,8], products of modal logics [9], etc.

Blockers and coblockers. The original decision algorithm for PEP^{reg} relies on so-called "*blocker*" and "*coblocker*" sets [1]. Write Sol_L for the set $\{x\in L \mid u(x)\sqsubseteq v(x)\}$ of solutions in some *constraint* language $L\subseteq\Sigma^*$ and define:

$$X_L \overset{\text{def}}{=} \{\alpha\in\Gamma^* \mid \forall x\in L, \alpha.u(x)\not\sqsubseteq v(x)\}, \qquad \text{(left L-blockers)}$$

$$X'_L \overset{\text{def}}{=} \{\alpha\in\Gamma^* \mid \forall x\in L, u(x).\alpha\not\sqsubseteq v(x)\}, \qquad \text{(right L-blockers)}$$

$$Y_L \overset{\text{def}}{=} \{\alpha\in\Gamma^* \mid \forall x\in L, u(x)\not\sqsubseteq \alpha.v(x)\}, \qquad \text{(left L-coblockers)}$$

$$Y'_L \overset{\text{def}}{=} \{\alpha\in\Gamma^* \mid \forall x\in L, u(x)\not\sqsubseteq v(x).\alpha\}. \qquad \text{(right L-coblockers)}$$

⋆ Work supported by the Agence Nationale de la Recherche, grant ANR-06-SETIN-001.

A key observation is that, in order to decide whether Sol_L is empty or not, it is simpler to reason about blocker and coblocker sets (see [1, Section 3] for more details on the decision algorithm). Rather than considering what are the solutions, the blocker and coblocker sets provide information on what latitude is allowed/required by the solutions, in particular by the most permissive ones. As a special case, they can tell us whether a given $\mathsf{PEP}^{\text{reg}}$ instance is solvable since

$$Sol_L = \varnothing \text{ iff } \varepsilon \in X_L \text{ iff } \varepsilon \in X'_L \text{ iff } \varepsilon \in Y_L \text{ iff } \varepsilon \in Y'_L. \tag{1}$$

Working with blocker sets rather than solutions sets has two main advantages:

– First, blocker and coblocker sets behave smoothly as a function of the constraint set L. This allows compositional reasoning w.r.t. L. The "Stability Inequations" (see long version of this paper) is the main example, but there are more. For instance, assume L is the product (concatenation) of two languages: $L = L_1.L_2$. Clearly Sol_L contains $Sol_{L_1}.Sol_{L_2}$. However the containment is strict in general, and it is not possible to express Sol_L as a function of Sol_{L_1} and Sol_{L_2}. By contrast, the following holds:

$$X_{L_1.L_2} = \Gamma^* \text{ iff } \left(X'_{L_1} \cup Y_{L_2}\right) \cap \left(Y'_{L_1} \cup X_{L_2}\right) = \Gamma^*. \tag{2}$$

– Second, blocker and coblocker sets are always regular languages, unlike the Sol_L sets [10]. This makes them easier to handle algorithmically, representing them via FSA's or regular expressions. In particular, compositional reasoning as exemplified in Equation (2) can easily be turned into simple and effective algorithms.

Our contribution. In this paper we consider the computability of the blocker and coblocker sets X_R and Y_R for R a regular constraint language. This is a natural question in view of the decision algorithm for $\mathsf{PEP}^{\text{reg}}$, where lower approximations of these sets are enumerated. More importantly, and as we explain in Section 7, it is another step in our attempts at enlarging the class of known decidable problems that combine Post-embedding and regular constraints.

We prove that blocker sets are not computable[1] while, quite unexpectedly, coblocker sets are computable. Concerning blocker sets, and since they cannot be computed, we consider decision problems that are weaker than computability, e.g., whether a blocker set is empty, infinite, whether is it contained in ("safety"), or contains ("cosafety"), a given set. A summary of our results will be found in Fig. 3 (section 3). In addition, we answer a question raised by [10] and prove that the regularity of Post-embedding languages is undecidable.

Comparison with existing work. This work continues our exploration of the Regular Post Embedding Problem. The problem was introduced and proved decidable in [1]. The links between lossy channels and $\mathsf{PEP}^{\text{reg}}$ are clarified in [2] where it is also shown that looking for infinite solutions within an ω-regular constraint set can be reduced to

[1] Here, and in the rest of the paper, we say informally that regular sets like X_L are "*computable*" when we really mean that an index for them can be computed uniformly from an index for L.

looking for finite solutions. In [10] it is shown how to count solutions, and how to check whether a regular property entails Post embedding. That blocker sets are not computable was claimed in [1, Remark 3.8] without any details or proofs (nor comments on the difference between blocker and coblocker sets).

Outline of the paper. Section 2 recalls the necessary definitions and notations, and proves a few useful lemmas on subwords. Section 3 formally introduces the problems we address. Then Section 4 shows how to compute coblocker sets, while Section 5 considers what can be computed on blocker sets. The undecidability results in that section are proved by a reduction from lossy counter machines described in Section 6. Proofs omitted in the main text can be found in the full version of this extended abstract.

2 Notations and Definitions

Words and their morphisms. We write $x,y,w,t,\sigma,\rho,\alpha,\beta,\ldots$ for words, i.e., finite sequences of letters such as $a,b,i,j,\ldots$ from alphabets $\Sigma,\Gamma,\ldots$. With $x.y$, or xy, we denote the concatenation of x and y. With ε we denote the empty word. The *length* of x is written $|x|$.

A *morphism* from Σ^* to Γ^* is a map $u : \Sigma^* \to \Gamma^*$ that respects the monoidal structure, i.e., with $u(\varepsilon) = \varepsilon$ and $u(x.y) = u(x).u(y)$. A morphism u is completely defined by its image $u(a)$, $u(b)$, $\ldots$, on $\Sigma = \{a,b,\ldots\}$. Most of the time, we shall write $u_a, u_b, \ldots$, and u_x, instead of $u(a), u(b), \ldots$, and $u(x)$.

The mirror image of a word x is denoted $\widetilde{x}$, e.g., $\widetilde{abc} = bca$. The mirror image of a language L is $\widetilde{L} \overset{\text{def}}{=} \{\widetilde{x} \mid x \in L\}$. The mirror image of a morphism u, denoted $\widetilde{u}$, is defined by $\widetilde{u}(a) \overset{\text{def}}{=} \widetilde{u(a)}$, so that $\widetilde{u}(x) = \widetilde{u(\widetilde{x})}$.

Subword ordering. Given two words x and y, we write $x \sqsubseteq y$ when x is a (scattered) *subword* of y, i.e., when x can be obtained by erasing some letters (possibly none) from y. For example, $abba \sqsubseteq \underline{ab}racada\underline{b}r\underline{a}$. The subword relation is a partial ordering, compatible with the monoidal structure: $\varepsilon \sqsubseteq x$, and $xy \sqsubseteq x'y'$ when $x \sqsubseteq x'$ and $y \sqsubseteq y'$. Higman's Lemma further states that, over a finite alphabet, the subword relation is a well-quasi-ordering, i.e., it is well-founded and all antichains (sets of incomparable words) are finite.

Section 6 relies on the following lemma (see long version of this paper for a proof):

Lemma 2.1 (Elimination Lemma)
If $xw \sqsubseteq y$ and $x' \sqsubseteq wy'$ then $xx' \sqsubseteq yy'$.
If $x \sqsubseteq yw$ and $wx' \sqsubseteq y'$ then $xx' \sqsubseteq yy'$.

Upward-closed and downward-closed languages. A language $L \subseteq \Gamma^*$ is *upward-closed* if $x \in L$ and $x \sqsubseteq y$ imply $y \in L$. It is *downward-closed* if $x \in L$ and $y \sqsubseteq x$ imply $y \in L$ (equivalently, if its complement is upward-closed). Higman's Lemma entails that upward-closed languages and downward-closed languages are regular [11]. In fact, upward-closed languages can be denoted by very simple regular expressions since they obviously reside at level 1/2 of the Straubing-Thérien Hierarchy [12]. Downward-closed languages too can be denoted by simple regular expressions [13,14]. In Section 4 we use "$*$-products", defined as concatenations of *atoms* that are either of the form $a+\varepsilon$ for some $a\in\Gamma$, or of the form A^* for some sub-alphabet $A\subseteq\Gamma$. For example, with $\Gamma=\{a,b,c\}$, the set of subwords of *abac* is $(a+\varepsilon).(b+\varepsilon).(a+\varepsilon).(c+\varepsilon)$ and the set of words that do not have *ab* as a subword is $\{b,c\}^*.\{a,c\}^*$. Any downward-closed language is, in a unique way, a finite union of maximal $*$-products.

3 Blockers and Coblockers

In the rest of the paper, we consider a generic PEP instance given by some $u,v:\Sigma^*\to\Gamma^*$. Recall that, for a regular constraint set $R\subseteq\Sigma^*$, the (left) blocker and coblocker sets X_R and Y_R are defined by:

$$X_R \overset{\text{def}}{=} \{\alpha\in\Gamma^* \mid \forall x\in R, \alpha.u_x \not\sqsubseteq v_x\}, \qquad Y_R \overset{\text{def}}{=} \{\alpha\in\Gamma^* \mid \forall x\in R, u_x \not\sqsubseteq \alpha.v_x\}.$$

Observe that X_R is upward-closed and Y_R is downward-closed. Hence both are regular.

Remark 3.1. In the rest of the paper, starting with Def. 3.2 below, we restrict our attention to the *left* sets X_L and Y_L. This is no loss of generality in view of the symmetry between the left-handed and the right-handed notions: α is a right L-blocker (or coblocker) if, and only if, $\widetilde{\alpha}$ is a left $\widetilde{L}$-blocker (resp., coblocker) in the mirror instance $\widetilde{u},\widetilde{v}$. □

For blocker and coblocker sets, we consider questions that range in generality from just checking one α for membership, to computing the whole set.

Definition 3.2 (Decision problems for blocker and coblocker sets). *We consider questions where one is given two morphisms $u,v:\Sigma^*\to\Gamma^*$ and a regular language $R\subseteq\Sigma^*$ as inputs, with possibly some additional input in the form of a word $\alpha\in\Gamma^*$, or a regular "safe" set $S\subseteq\Gamma^*$.*

- Blockers_Membership*: does $\alpha\in X_R$?*
- Blockers_Emptiness*: does $X_R=\varnothing$?*
- Blockers_Universality*: does $X_R=\Gamma^*$?*
- Blockers_Safety*: does $X_R\subseteq S$?*
- Blockers_Cosafety*: does $S\subseteq X_R$?*
- Blockers_Finiteness*: is X_R finite?*
- Blockers_Cofiniteness*: is X_R cofinite?, i.e., is $\Gamma^*\setminus X_R$ finite?*

The same decision problems CoBlockers_Membership, CoBlockers_Safety, ..., *are defined for coblocker sets.*

Finally, Blockers_Computation *and* CoBlockers_Computation *ask one to* compute *a representation of X_R (resp., Y_R) under the form of a regular expression or a FSA. (These are not decision problems).*

Remark 3.3. The restriction to *regular* safe sets S is a natural assumption that is both expressive and tractable. However, in our setting where blocker and coblocker sets are upward-closed (resp., downward-closed), the expressive power is even larger. Indeed, for any L, $X_R\subseteq L$ iff $X_R\subseteq S$ where S is the upward-closure of L. Thus, and since the upward-closure of L is always regular, our positive results automatically apply to any class of safe sets for which the upward and downward closures can be effectively computed (e.g., context-free languages [15]). □

	Blockers	**Coblockers**
Membership	decidable (Coro. 4.2)	decidable (Coro. 4.2)
Safety	undecidable (Theo. 5.3)	decidable (Theo. 4.3)
Cosafety	decidable (Coro. 4.2)	decidable (Coro. 4.2)
Emptiness	undecidable (Theo. 5.3)	decidable (Theo. 4.3)
Universality	decidable (Coro. 4.2)	trivial
Finiteness	undecidable (Theo. 5.3)	decidable (Theo. 4.3)
Cofiniteness	undecidable (Theo. 5.2)	trivial
Computation	no	yes (Theo. 4.3)

Fig. 1. Computability for blocker and coblocker sets. See Remark 3.5 about complexity.

Remark 3.4 (Relations among problems). Safety is a general problem that subsumes Emptiness and Membership. Cosafety subsumes Universality and (non-)Membership. Blockers_Universality reduces to Blockers_Membership since $X_R = \Gamma^*$ iff $\varepsilon \in X_R$. CoBlockers_Universality is trivial since $Y_R = \Gamma^*$ iff $R = \varnothing$. Finiteness and Cofiniteness are natural counting questions. Finiteness coincides with Emptiness for blocker sets (assuming Γ is not empty) and more generally for all upward-closed sets (Cofiniteness and Universality coincide for downward-closed sets in general, and coblocker sets in particular).

There are no other obvious reductions between the above decision problems (e.g., Finiteness and Cofiniteness are in general unrelated).

Regarding computability of the blocker and coblocker sets, observe that since these sets are regular, the decidability of Safety and Cosafety would entail their computability (see also Section 4). Conversely, all the decision problems listed above can easily be answered from an FSA description of the sets. Hence our decision problems can be seen as different special cases of the general Blockers_Computation and CoBlockers_-Computation problems. □

Remark 3.5 (On the complexity of blocker and coblocker problems). All the non-trivial problems listed in Def. 3.2 are more general than PEP$^{\text{reg}}$. This was made precise in Remark 3.4 except for CoBlockers_Finiteness, but it is easy to provide a reduction from CoBlockers_Emptiness to CoBlockers_Finiteness: add one extra symbol to Γ, ensuring that Y_R is finite iff it is empty. Hence all the above problems are at least as hard as PEP$^{\text{reg}}$ and none of them is multiply-recursive. □

4 Computing Coblocker Sets

We start with the computability results. They can be obtained via reductions to PEP$^{\text{reg}}$:

Lemma 4.1. Blockers_Cosafety *and* CoBlockers_Cosafety *many-one reduce to (the complement of)* PEP$^{\text{reg}}$.

Proof. Blockers_Cosafety: with u, v, R and S we associate a PEP$^{\text{reg}}$ instance $u',v' : \Sigma'^* \to \Gamma^*$ and a regular constraint $R' \subseteq \Sigma'^*$. Assume w.l.o.g. that Σ and Γ are disjoint alphabets and let $\Sigma' \stackrel{\text{def}}{=} \Sigma \cup \Gamma$. u' and v' are extensions of u and v with $u'(\gamma) = \gamma$ and $v'(\gamma) = \varepsilon$ for all $\gamma \in \Gamma$. Finally let $R' \stackrel{\text{def}}{=} S.R$, this is indeed a regular subset of Σ'^*.

Now, u', v', R' is a positive PEP$^{\text{reg}}$ instance iff $u'_x \sqsubseteq v'_x$ for some $x \in R'$, iff $u'_{\alpha y} \sqsubseteq v'_{\alpha y}$ for some $\alpha \in S$ and some $y \in R$, iff $u'_\alpha.u'_y \sqsubseteq v'_\alpha.v'_y$, iff $\alpha.u_y \sqsubseteq v_y$ for some $\alpha \in S$ and y, i.e., iff some $\alpha \in S$ is not in X_R, i.e., $S \not\subseteq X_R$.

CoBlockers_Cosafety: the same idea works provided we let $u'(\gamma) = \varepsilon$ and $v'(\gamma) = \gamma$. □

Since PEP$^{\text{reg}}$ is decidable, and thanks to Remark 3.4, Lemma 4.1 entails:

Corollary 4.2. *For blocker and coblocker sets,* Cosafety*,* Universality *and* Membership *are decidable.*

We are now ready to proceed to the main computability result:

Theorem 4.3. *The coblocker sets Y_R and Y'_R are computable.*

Our proof simply leverages the decidability of CoBlockers_Cosafety (Coro. 4.2) with the VJGL Lemma (here specialized to words with embeddings).

Lemma 4.4 (VJGL Lemma, see Theo. 2 of [16]). *Let $(U_i)_i$ be an enumeration of upward-closed languages on some finite alphabet. One can compute a finite representation for the U_i's if, and only if, one can decide whether $U_i \cap P = \varnothing$ for $*$-products P (when i and P are inputs).*

Here, computing "a finite representation" means computing the finite basis, i.e, the set of minimal words, but this can easily be transformed into a regular expression or an FSA representation. The VJGL-Lemma is based on a generic algorithm that, in the case of words with embedding, computes such finite bases using an oracle for non-intersection with $*$-products.

Another wording of the VJGL-Lemma is given by the following corollary.

Corollary 4.5. *1. If $(U_i)_i$ are upward-closed languages with a decidable safety problem, then they are computable.*
2. Equivalently, if $(V_i)_i$ are downward-closed languages with a decidable cosafety problem, then they are computable.

Proof. $U_i \cap P = \varnothing$ is equivalent to $U_i \subseteq (\Sigma^* \setminus P)$, a safety question. □

We now prove Theorem 4.3: The coblocker sets Y_R are downward-closed and have a decidable cosafety problem (Coro. 4.2). Hence they are computable by Coro. 4.5.2. Then Theorem 4.3 accounts for all the positive results on coblocker sets in Fig. 3.

5 Blocker Sets Are Not Computable

It is not possible to effectively compute the blocker sets X_R from given u, v, R, even though X_R is known to be regular. This is shown with Lemma 5.1, our main negative result (proved in Section 6):

Lemma 5.1. Blockers_Cofiniteness *is Σ^0_1-hard and* Blockers_Emptiness *is Π^0_1-hard.*

With Lemma 5.1, we are in a position to prove all the undecidability results in Fig. 3:

Theorem 5.2. Blockers_Cofiniteness *is* Σ^0_1*-complete.*

Proof (Sketch). Membership in Σ^0_1 can be seen by writing the cofiniteness of X_R under the form $\exists n \in \mathbb{N}, \Gamma^{\geq n} \subseteq X_R$ and relying on the decidability of Blockers_Cosafety (Coro. 4.2). □

Theorem 5.3. Blockers_Safety, Blockers_Emptiness *and* Blockers_Finiteness *are* Π^0_1*-complete.*

Proof. The Π^0_1-hardness of Blockers_Emptiness (Lemma 5.1) also applies to Blockers_-Finiteness (since the two problems coincide) and Blockers_Safety (a more general problem), see Remark 3.4.

For upper bounds, we observe that Blockers_Safety (hence also Blockers_Emptiness) is in Π^0_1 since it can be written under the form $\forall \alpha \in \Gamma^*, (\alpha \in S \vee \alpha \notin X_R)$ (recall that $\alpha \notin X_R$ is decidable). □

6 Lossy Counter Machines

Lossy counter machines or, for short, *LCM*'s, were introduced by R. Mayr [17]. They are a variant of Minsky counter machines (with zero-test, increments and decrements) where counters are *lossy*, i.e., they may decrease non-deterministically. We only give a streamlined presentation of LCM's here and refer to [17,18] for more details.

Let $M = (Q, C, \Delta, q_{\text{init}})$ be a Minsky counter machine with finite set of control states $Q \ni q_{\text{init}}$, finite set of counters C, and finite set of transitions rules Δ. Four counters are sufficient for our purposes so we fix $C = \{c_1, c_2, c_3, c_4\}$. A configuration of M is some $\tau = (q, n_1, n_2, n_3, n_4) \in Conf(M) \overset{\text{def}}{=} Q \times \mathbb{N}^4$, with *size*, denoted $|\tau|$, being $n_1 + n_2 + n_3 + n_4$. We (partially) order $Conf(M)$ with

$$(q, n_1, n_2, n_3, n_4) \leq (q', n'_1, n'_2, n'_3, n'_4) \overset{\text{def}}{\Leftrightarrow} q = q' \wedge n_1 \leq n'_1 \wedge \cdots \wedge n_4 \leq n'_4.$$

An initial state $q_{\text{init}} \in Q$ is fixed, and the initial configuration is $\tau_{\text{init}} \overset{\text{def}}{=} (q_{\text{init}}, 0, 0, 0, 0)$. Observe that the only way to have $\tau \leq \tau_{\text{init}}$ is with $\tau = \tau_{\text{init}}$.

A transition rule δ is a directed edge between states of M, labeled by an operation $op \in OP \overset{\text{def}}{=} C \times \{\texttt{++}, \texttt{--}, \texttt{=0?}\}$, and denoted (q, op, q'). The rules in Δ give rise to two different transition relations between configurations. First, steps $\tau \xrightarrow{\delta} \tau'$ are defined in the expected way. Formally, with $\delta = (q_1, op, q_2)$, there is a step $(q, n_1, n_2, n_3, n_4) \xrightarrow{\delta} (q', n'_1, n'_2, n'_3, n'_4)$ if, and only if, the following three conditions are satisfied:

1. $q_1 = q$ and $q_2 = q'$;
2. *op* is some c_k++ or c_k-- or c_k=0?, and $n'_i = n_i$ for all $i \neq k$;
3. if *op* is c_k++ then $n'_k = n_k + 1$; if *op* is c_k-- then $n'_k = n_k - 1$; if *op* is c_k=0? then $0 = n_k = n'_k$.

These so-called *perfect steps* describe the operational semantics of M when its counters are not assumed to be lossy. Then a second operational semantics, with transitions denoted $\tau \xrightarrow{\delta}_{\text{lossy}} \tau'$, is derived[2] in the following way:

$$\tau \xrightarrow{\delta}_{\text{lossy}} \tau' \overset{\text{def}}{\Leftrightarrow} \tau \xrightarrow{\delta} \tau'' \text{ for some } \tau'' \geq \tau'. \tag{3}$$

These *lossy steps* describe the behavior of M when its counters are assumed to be lossy. In the usual way, the δ superscript on transitions is omitted when irrelevant. *Lossy runs*, denoted $\tau_0 \xrightarrow{*}_{\text{lossy}} \tau_n$, are sequences of chained lossy steps $\tau_0 \rightarrow_{\text{lossy}} \tau_1 \rightarrow_{\text{lossy}} \cdots \tau_n$. We write $\mathit{Reach}_{\text{lossy}}(M)$ for the set of configurations that can be reached via lossy runs of M, starting from τ_{init}.

We rely on known undecidability results on LCM's and use the following two problems:

LCM_Infinite**:** the question whether $\mathit{Reach}_{\text{lossy}}(M)$ is infinite, for a given LCM M;
LCM_Unbounded_Counter**:** the question whether $\mathit{Reach}_{\text{lossy}}(M)$ contains configurations with arbitrarily large values for the first counter c_1.

These two problems are a variant of one another, and they are easily seen to be inter-reducible. The following theorem is from [17,18]:

Theorem 6.1. LCM_Infinite *and* LCM_Unbounded_Counter *are* Π^0_1*-complete.*

6.1 From Lossy Counters to Post-Embedding

With a LCM $M = (Q, C, \Delta, q_{\text{init}})$ we associate a PEP instance $u, v : \Sigma^* \rightarrow \Gamma^*$ that will be used in three different reductions (with different constraint languages $R_1, R_2, R_3 \subseteq \Sigma^*$). Here $\Gamma \overset{\text{def}}{=} Q \cup C$ is used to encode the configurations of M: a configuration $\tau = (q, n_1, n_2, n_3, n_4)$ is encoded by the word $c_1^{n_1} c_2^{n_2} c_3^{n_3} c_4^{n_4} q$, denoted $\lceil \tau \rceil$. Observe that $\lceil \tau \rceil \sqsubseteq \lceil \tau' \rceil$ iff $\tau \leq \tau'$.

We further let $\Sigma \overset{\text{def}}{=} \Gamma \cup \Delta \cup \mathit{OP} \cup \overline{Q} \cup \overline{C}$ where $\overline{Q} = \{\overline{q} \mid q \in Q\}$ and $\overline{C} = \{\overline{c}_1, \overline{c}_2, \overline{c}_3, \overline{c}_4\}$ are copies of Q and C, with new symbols obtained by overlining the original symbols from $Q \cup C$. We define two morphisms $u, v : \Sigma^* \rightarrow \Gamma^*$ with

$$u((q, op, q')) \overset{\text{def}}{=} q, \qquad v((q, op, q')) \overset{\text{def}}{=} q', \qquad u(\overline{c}_i) \overset{\text{def}}{=} c_i, \qquad v(\overline{c}_i) \overset{\text{def}}{=} c_i,$$
$$u(c_i{+}{+}) \overset{\text{def}}{=} \varepsilon, \qquad v(c_i{+}{+}) \overset{\text{def}}{=} c_i, \qquad u(c_i{-}{-}) \overset{\text{def}}{=} c_i, \qquad v(c_i{-}{-}) \overset{\text{def}}{=} \varepsilon.$$

How u and v evaluate on the rest of Σ will be defined later when it becomes relevant.

With every transition rule $\delta = (q, op, q')$ in Δ, we associate a language $R_\delta \subseteq \Sigma^*$ given via the following regular expressions:

$$R_\delta \overset{\text{def}}{=} \begin{cases} \overline{c}_1{}^* \cdots \overline{c}_{k-1}{}^* \cdot op \cdot \overline{c}_k{}^* \cdots \overline{c}_4{}^* \cdot \delta & \text{if } op \text{ is } c_k{+}{+} \text{ or } c_k{-}{-}, \\ \overline{c}_1{}^* \cdots \overline{c}_{k-1}{}^* \cdot \overline{c}_{k+1}{}^* \cdots \overline{c}_4{}^* \cdot \delta & \text{if } op \text{ is } c_k\texttt{=0?}. \end{cases}$$

[2] Lossy steps could also be defined *directly* without deriving them from perfect steps, but the indirect definition is very convenient as it permits reasoning simultaneously on both kinds of steps for the same counter machine.

These definitions ensure that, when $x \in R_\delta$, u_x and v_x are the encodings of related configurations. We let the reader check that the following more precise statement holds:

Lemma 6.2
1. If $x \in R_\delta$, then $u_x = \lceil\tau\rceil$ and $v_x = \lceil\tau'\rceil$ for some configurations τ, τ' such that $\tau \xrightarrow{\delta} \tau'$.
2. Reciprocally, if $\tau \xrightarrow{\delta} \tau'$, then $\lceil\tau\rceil = u_x$ and $\lceil\tau'\rceil = v_x$ for some (unique) $x \in R_\delta$.

We further define $R_\Delta \overset{\text{def}}{=} \bigcup_{\delta\in\Delta} R_\delta$ and $R_M \overset{\text{def}}{=} (R_\Delta)^*$: these languages are regular.

Lemma 6.3. *Let $\alpha \in \Gamma^*$. If $u_x.\alpha \sqsubseteq \lceil\tau_{init}\rceil.v_x$ for some $x \in R_M$, then $\alpha \sqsubseteq \lceil\tau\rceil$ for some $\tau \in Reach_{lossy}(M)$.*

Proof. We assume $\alpha \neq \varepsilon$ and $x \neq \varepsilon$, otherwise $\alpha \sqsubseteq \lceil\tau_{\text{init}}\rceil$ trivially. Thus $x \in R_M$ must be of the form $x = x_1 \dots x_n$ with $n > 0$ and $x_i \in R_\Delta$ for all $i = 1, \dots, n$. By Lemma 6.2, u_x is some $\lceil\tau_0\rceil.\lceil\tau_1\rceil \dots \lceil\tau_{n-1}\rceil$ and v_x is some $\lceil\tau'_1\rceil.\lceil\tau'_2\rceil \dots \lceil\tau'_n\rceil$ such that, for all $i = 1, \dots, n$, $\tau_{i-1} \to \tau'_i$ is a perfect step of M.

We now use the assumption that $u_x.\alpha \sqsubseteq \lceil\tau_{\text{init}}\rceil.v_x$. Since $\alpha \neq \varepsilon$, u_x embeds into a strict prefix, denoted w, of $\lceil\tau_{\text{init}}\rceil.v_x$. Note that u_x contains $n > 0$ symbols from Q and ends with one of them, while w has at most n (it is shorter than $\lceil\tau_{\text{init}}\rceil.v_x$ that has $n+1$ symbols from Q and ends with one of them). Hence w necessarily has n symbols from Q and $u_x.\alpha \sqsubseteq \lceil\tau_{\text{init}}\rceil.v_x$ can be decomposed as $\lceil\tau_i\rceil \sqsubseteq \lceil\tau'_i\rceil$ (i.e., $\tau_i \leq \tau'_i$) for all $i = 1, \dots, n-1$, with also $\lceil\tau_0\rceil \sqsubseteq \lceil\tau_{\text{init}}\rceil$ (hence $\tau_0 = \tau_{\text{init}}$) and $\alpha \sqsubseteq \lceil\tau'_n\rceil$. Combining with $\tau_{i-1} \to \tau'_i$ we deduce $\tau_{i-1} \to_{\text{lossy}} \tau_i$ for $i = 1, \dots, n-1$. Finally $\tau_{\text{init}} = \tau_0 \to_{\text{lossy}} \tau_1 \cdots \to_{\text{lossy}} \tau_{n-1} \to \tau'_n$ is a lossy run of M, so that $\tau'_n \in Reach_{\text{lossy}}(M)$. □

There is a converse to Lemma 6.3:

Lemma 6.4. *If $\tau \in Reach_{lossy}(M)$, there exists some $x \in R_M$ such that $u_x.\lceil\tau\rceil \sqsubseteq \lceil\tau_{init}\rceil.v_x$.*

Proof. Since $\tau \in Reach_{\text{lossy}}(M)$ there exists a lossy run $\tau_{\text{init}} = \tau_0 \to_{\text{lossy}} \tau_1 \to_{\text{lossy}} \cdots \tau_n = \tau$. We show, by induction on $i = 0, 1, \dots, n$, that $u_{x_i}.\lceil\tau_i\rceil \sqsubseteq \lceil\tau_{\text{init}}\rceil.v_{x_i}$ for some $x_i \in R_M$.

The base case, $i = 0$, is dealt with $x_0 = \varepsilon$ since $\tau_0 = \tau_{\text{init}}$.

For the case $i > 0$, we know by ind. hyp. that there is some $x_{i-1} \in R_M$ with

$$u_{x_{i-1}}.\lceil\tau_{i-1}\rceil \sqsubseteq \lceil\tau_{\text{init}}\rceil.v_{x_{i-1}}. \tag{4}$$

The lossy step $\tau_{i-1} \to_{\text{lossy}} \tau_i$ implies the existence of a perfect step $\tau_{i-1} \to \tau'$ with $\tau' \geq \tau_i$ (Equation (3)). Thus $\lceil\tau_{i-1}\rceil = u_y$ and $\lceil\tau'\rceil = v_y$ for some $y \in R_\Delta$ (Lemma 6.2).

From $\tau_i \leq \tau'$, we deduce

$$u_y.\lceil\tau_i\rceil \sqsubseteq \lceil\tau_{i-1}\rceil.v_y. \tag{5}$$

We now put together Equations (4) and (5). The Elimination Lemma yields

$$u_{x_{i-1}}.u_y.\lceil\tau_i\rceil \sqsubseteq \lceil\tau_{\text{init}}\rceil.v_{x_{i-1}}.v_y, \tag{6}$$

so that setting $x_i \overset{\text{def}}{=} x_{i-1}.y$ concludes our proof. We observe that $x_i \in R_M$ since $x_{i-1} \in R_M$ and $y \in R_\Delta$. □

6.2 Reducing LCM_Infinite and LCM_Unbounded_Counter to Blockers Problems

For the next step in the reduction, we extend u and v on $Q \cup C$ $(= \Gamma)$ with

$$u(\gamma) \stackrel{\text{def}}{=} \pi_1(\gamma) = \begin{cases} c_1 & \text{if } \gamma = c_1, \\ \varepsilon & \text{if } \gamma \in \Gamma \smallsetminus \{c_1\}, \end{cases} \qquad v(\gamma) \stackrel{\text{def}}{=} \gamma \text{ for all } \gamma \in \Gamma.$$

When $\alpha \in \Gamma^*$, we shall write $\pi_1(\alpha)$ rather than u_α to emphasize the fact that u only retains the c_1 symbols of α and erases the rest. Below, we rely on a few obvious properties of this erasing morphism, such as $\pi_1(\alpha) \sqsubseteq \alpha$, or $\pi_1(\alpha\beta) = \pi_1(\beta\alpha)$, and in particular the following:

Fact 6.5. *For all $\beta \in \Gamma^*$ and $x, y \in \Sigma^*$, $x.c_1.\pi_1(\beta) \sqsubseteq y.\beta$ implies $x.c_1 \sqsubseteq y$.*

Finally, we let $R_1 \stackrel{\text{def}}{=} q_{\text{init}}.R_M$ and $R_2 \stackrel{\text{def}}{=} R_1.\Gamma^*$. This provides two different reductions, with properties captured by Lemmas 6.6 and 6.8.

Lemma 6.6. *Let $\alpha \in \Gamma^*$. The following are equivalent:*
(1) $\alpha \notin X'_{R_1}$,
(2) there exists $x \in R_1$ such that $u_x.\alpha \sqsubseteq v_x$,
(3) there exists $\tau \in Reach_{lossy}(M)$ such that $\alpha \sqsubseteq \lceil\tau\rceil$.

Proof (Sketch). (1) $\Leftrightarrow$ (2) by definition of X'_{R_1}. Then, given the definitions of R_1, u and v, Lemma 6.3 shows "(2) $\Rightarrow$ (3)" (note that $u(q_{\text{init}}) = \varepsilon$ and $v(q_{\text{init}}) = q_{\text{init}} = \lceil\tau_{\text{init}}\rceil$). Finally, Lemma 6.4 shows "(3) $\Rightarrow$ (2)". □

In particular, X'_{R_1} is cofinite iff M does not satisfy LCM_Infinite.

Corollary 6.7. Blockers_Cofiniteness *is Σ^0_1-hard.*

Lemma 6.8. *Let $\alpha \in \Gamma^*$. The following are equivalent:*
(1) $\alpha \notin X'_{R_2}$,
(2) there exists $y \in R_2$ such that $u_y.\alpha \sqsubseteq v_y$,
(3) there exists $\tau \in Reach_{lossy}(M)$ such that $\pi_1(\alpha) \sqsubseteq \pi_1(\lceil\tau\rceil)$.

Proof. (1) $\Leftrightarrow$ (2) by definition of X'_{R_2}.
(3) $\Rightarrow$ (2): Assume $\pi_1(\alpha) \sqsubseteq \pi_1(\lceil\tau\rceil)$ for some $\tau \in Reach_{\text{lossy}}(M)$. Then, $\pi_1(\alpha) \sqsubseteq \lceil\tau\rceil$ so that, by Lemma 6.6, there exists some $x \in R_1$ with $u_x.\pi_1(\alpha) \sqsubseteq v_x$. Appending α to the right yields $u_x.\pi_1(\alpha).\alpha = u_x.u_\alpha.\alpha \sqsubseteq v_x.\alpha = v_x.v_\alpha$. Letting $y \stackrel{\text{def}}{=} x.\alpha$ $(\in R_2)$ proves (2).
(2) $\Rightarrow$ (3): Assume $u_y.\alpha \sqsubseteq v_y$ for some $y \in R_2$ of the form $x.\beta$ with $x \in R_1$ and $\beta \in \Gamma^*$. We assume $\pi_1(\alpha) \neq \varepsilon$ since otherwise $\pi_1(\alpha) \sqsubseteq \pi_1(\lceil\tau_{\text{init}}\rceil)$ holds trivially. From $u_y.\alpha \sqsubseteq v_y$, we deduce

$$u_x.\pi_1(\alpha).\pi_1(\beta) = u_x.\pi_1(\beta).\pi_1(\alpha) = u_y.\pi_1(\alpha) \sqsubseteq u_y.\alpha \sqsubseteq v_y = v_x.v_\beta = v_x.\beta.$$

From $u_x.\pi_1(\alpha).\pi_1(\beta) \sqsubseteq v_x.\beta$, one deduces $u_x.\pi_1(\alpha) \sqsubseteq v_x$ (using Fact 6.5 and the assumption that $\pi_1(\alpha) \neq \varepsilon$). Thus there exists a $\tau \in Reach_{\text{lossy}}(M)$ with $\pi_1(\alpha) \sqsubseteq \lceil\tau\rceil$ (Lemma 6.3), hence $\pi_1(\alpha) \sqsubseteq \pi_1(\lceil\tau\rceil)$. □

In other words, $\alpha \notin X'_{R_2}$ iff there is a reachable configuration where the c_1 counter is larger than, or equal to, the number of c_1 symbols in α. Thus $X'_{R_2} = \varnothing$ iff M satisfies LCM_Unbounded_Counter.

Corollary 6.9. Blockers_Emptiness *is* Π^0_1*-hard.*

As an aside, the reduction from LCM's can be used to prove Theo. 6.11 below. The regularity problem for Post-embedding languages is a natural question since Sol_R is not always regular, and since comparisons with a regular S are possible:

Theorem 6.10 ([10]). *The questions, for $S \subseteq \Sigma^*$ a regular language, whether $S \subseteq Sol_R$, and whether $Sol_R \subseteq S$, are decidable.*

Theorem 6.11. *The question whether, for $u, v : \Sigma^* \to \Gamma^*$ and a regular $R \subseteq \Sigma^*$, Sol_R is a regular language, is Σ^0_1-complete.*

The proof for Σ^0_1-hardness simply adapts our previous reduction, providing u, v and R such that Sol_R is regular iff $Reach_{\text{lossy}}(M)$ is finite, then relying on Theo. 6.1.

7 Concluding Remarks

The decidability of PEP$^{\text{reg}}$ is the decidability of existential questions of the form

$$\exists x \in R : u(x) \sqsubseteq v(x) \tag{Q1}$$

for regular R's. This result is fragile and does not extend easily. When one looks for solutions satisfying more expressive constraints, e.g., deterministic context-free, or also Presburger-definable, the problem becomes undecidable [1]. In another direction, combining two embeddings quickly raises undecidable questions, e.g., the following questions are undecidable [10, Theo. 4.1]:

$$\exists x \in \Sigma^+ : (u_1(x) \sqsubseteq v_1(x) \wedge u_2(x) \sqsubseteq v_2(x)), \tag{Q2}$$

$$\exists x \in \Sigma^+ : (u_1(x) \sqsubseteq v_1(x) \wedge u_2(x) \not\sqsubseteq v_2(x)). \tag{Q3}$$

Remark that, by Theorem 6.10, the following universal question is decidable [10]:

$$\forall x \in R : u(x) \sqsubseteq v(x). \tag{Q4}$$

This suggests considering questions like

$$\forall x \in R\, \exists x' \in R' : u(xx') \sqsubseteq v(xx'), \tag{Q5}$$

$$\exists x \in R\, \forall x' \in R' : u(xx') \sqsubseteq v(xx'). \tag{Q6}$$

The undecidability of (Q5) is clear since already Blockers_Emptiness is undecidable. The (un?)decidability of (Q6) is still open. We believe blockers and coblockers may play a useful role here. Indeed, by analogy with blockers, we may define

$$A_R \stackrel{\text{def}}{=} \{\alpha \mid \forall x \in R, \alpha.u(x) \sqsubseteq v(x)\}, \qquad B_R \stackrel{\text{def}}{=} \{\beta \mid \forall x \in R, u(x) \sqsubseteq \beta.v(x)\}.$$

Note that membership in A_R (or in B_R), being an instance of (Q5), is decidable. Furthermore, B_R is upward-closed and A_R is finite (unless R is empty). Now, the following observation:

$$\big(\exists x \in R\, \forall x' \in R' : u(xx') \sqsubseteq v(xx')\big) \quad \text{iff} \quad \big((A_{R'} \smallsetminus Y_R) \cup (B_{R'} \smallsetminus X_R) \neq \varnothing\big)$$

provides a direct link between (Q6) and blocker-like languages. We leave this as a suggestion for future investigations.

References

1. Chambart, P., Schnoebelen, P.: Post embedding problem is not primitive recursive, with applications to channel systems. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 265–276. Springer, Heidelberg (2007)
2. Chambart, P., Schnoebelen, P.: The ω-Regular Post Embedding Problem. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 97–111. Springer, Heidelberg (2008)
3. Chambart, P., Schnoebelen, P.: The ordinal recursive complexity of lossy channel systems. In: Proc. LICS 2008, pp. 205–216. IEEE Comp. Soc. Press, Los Alamitos (2008)
4. Ouaknine, J., Worrell, J.: On the decidability and complexity of Metric Temporal Logic over finite words. Logical Methods in Comp. Science 3(1), 1–27 (2007)
5. Abdulla, P.A., Deneux, J., Ouaknine, J., Quaas, K., Worrell, J.: Universality analysis for one-clock timed automata. Fundamenta Informaticae 89(4), 419–450 (2008)
6. Lasota, S., Walukiewicz, I.: Alternating timed automata. ACM Trans. Computational Logic 9(2) (2008)
7. Jurdziński, T.: Leftist grammars are nonprimitive recursive. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 51–62. Springer, Heidelberg (2008)
8. Chambart, P., Schnoebelen, P.: Toward a compositional theory of leftist grammars and transformations. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 237–251. Springer, Heidelberg (2010)
9. Gabelaia, D., et al.: Non-primitive recursive decidability of products of modal logics with expanding domains. Annals of Pure and Applied Logic 142(1-3), 245–268 (2006)
10. Chambart, P., Schnoebelen, P.: Pumping and counting on the Regular Post Embedding Problem. In: Proc. ICALP 2010. LNCS. Springer, Heidelberg (2010)
11. Haines, L.H.: On free monoids partially ordered by embedding. J. Combinatorial Theory 76, 94–98 (1969)
12. Pin, J.-E., Weil, P.: Polynomial closure and unambiguous product. Theory of Computing Systems 30(4), 383–422 (1997)
13. Finkel, A., Goubault-Larrecq, J.: Forward analysis for WSTS, part I: Completions. In: Proc. STACS 2009. Leibniz International Proceedings in Informatics, pp. 433–444 (2009)
14. Abdulla, P.A., Collomb-Annichini, A., Bouajjani, A., Jonsson, B.: Using forward reachability analysis for verification of lossy channel systems. Formal Methods in System Design 25(1), 39–65 (2004)
15. van Leeuwen, J.: Effective constructions in well-partially-ordered free monoids. Discrete Mathematics 21(3), 237–252 (1978)
16. Goubault-Larrecq, J.: On a generalization of a result by Valk and Jantzen. Research Report LSV-09-09, Laboratoire Spécification et Vérification, ENS Cachan, France (May 2009)
17. Mayr, R.: Undecidable problems in unreliable computations. Theoretical Computer Science 297(1-3), 337–354 (2003)
18. Schnoebelen, P.: Lossy counter machines: A survey. In: Proc. RP 2010. LNCS. Springer, Heidelberg (2010)

Rankers over Infinite Words[⋆]

(Extended Abstract)

Luc Dartois[1], Manfred Kufleitner[2], and Alexander Lauser[2]

[1] ENS Cachan, France
ldartois@dptinfo.ens-cachan.fr
[2] FMI, Universität Stuttgart, Germany
{kufleitner,lauser}@fmi.uni-stuttgart.de

Abstract. We consider the fragments FO^2, $\Sigma_2 \cap \mathrm{FO}^2$, $\Pi_2 \cap \mathrm{FO}^2$, and Δ_2 of first-order logic $\mathrm{FO}[<]$ over finite and infinite words. For all four fragments, we give characterizations in terms of rankers. In particular, we generalize the notion of a ranker to infinite words in two possible ways. Both extensions are natural in the sense that over finite words they coincide with classical rankers, and over infinite words they both have the full expressive power of FO^2. Moreover, the first extension of rankers admits a characterization of $\Sigma_2 \cap \mathrm{FO}^2$ while the other leads to a characterization of $\Pi_2 \cap \mathrm{FO}^2$. Both versions of rankers yield characterizations of the fragment $\Delta_2 = \Sigma_2 \cap \Pi_2$. As a byproduct, we also obtain characterizations based on unambiguous temporal logic and unambiguous interval temporal logic.

1 Introduction

We consider fragments of two-variable first-order logic FO^2. Formulas are interpreted over words which may be finite or infinite. Over finite words only, a large number of different characterizations of FO^2 is known, see e.g. [8] or [2] for an overview. Some of the characterizations have been generalized to infinite words in [3]. We continue this line of work. For this paper the main difference between finite word models and infinite word models is the following: Over finite words, FO^2 and the fragment $\Delta_2 = \Sigma_2 \cap \Pi_2$ have the same expressive power [9], whereas Δ_2 is a strict subclass of FO^2 over infinite words. Moreover, in the case of infinite words, FO^2 is incomparable to Σ_2 and Π_2. By definition, Σ_2 is the class of formulas in prenex normal form with two blocks of quantifiers starting with a block of existential quantifiers, and Π_2 is the class of negations of Σ_2-formulas. Here and throughout the paper, we identify a logical fragment with the class of languages definable in the fragment.

An important concept in this paper are rankers which have been introduced by Immerman and Weis [10] in order to give a combinatorial characterization of quantifier alternation within FO^2 over finite words. Casually speaking, a *ranker*

[⋆] The last two authors acknowledge the support by the German Research Foundation (DFG) for the project DI 435/5-1.

is a sequence of instructions of the form *"go to the next a-position"* and *"go to the previous a-position"* for some letters a. For every word, a ranker is either undefined or it determines a unique position. We generalize rankers to infinite words in two possible ways. The main difference to finite words is that we have to define the semantics of *"go to the last a-position"* if there are infinitely many occurrences of the letter a. The first solution is to say that the position is undefined. The second approach is to stay at an infinite position. For example, if a word has infinitely many a-positions but only two b-positions, then in the first semantics *"go to the last a-position and from there, go to the previous b-position"* would be undefined while in the second semantics it would determine the last b-position. By delaying the interpretation of instructions until some letter with finite occurrence is met, the second semantics is reminiscent of the *lazy evaluation* principle, and we therefore call it *lazy rankers*. If we want to emphasize that we use the first semantics, then we often use the term *eager ranker*. The language $L(r)$ generated by a ranker r consists of all words on which r is defined. A *ranker language* is a Boolean combination of languages of the form $L(r)$.

In both ways, rankers admit natural combinatorial characterizations of the first-order fragments FO^2 (Theorems 1 and 5) and Δ_2 (Theorem 3) over finite and infinite words. Moreover, the eager semantics yields a characterization of $\Sigma_2 \cap \mathrm{FO}^2$ (Theorem 2) while lazy rankers lead to a characterization of $\Pi_2 \cap \mathrm{FO}^2$ (Theorem 4). We note that the decidability results for the first-order fragments lead to decidability results for the respective ranker fragments [3].

It turns out that unambiguous temporal logic [4] and unambiguous interval temporal logic [5] allow natural intermediate characterizations on the way from first-order logic to rankers. In particular, this yields temporal logic counterparts of the first-order fragments. Moreover, we show that it is possible to convert formulas in unambiguous interval temporal logic into equivalent formulas in unambiguous temporal logic, without introducing new negations (Propositions 1 and 2). This also leads to a new characterization of FO^2 over finite words in terms of restricted ranker languages (Corollary 1).

Due to lack of space, most proofs are omitted. For complete proofs, we refer to the full version of this paper [1].

2 Preliminaries

In the following Γ denotes a finite alphabet. For $A \subseteq \Gamma$, we denote by A^* the set of finite words over A. The set of infinite words is A^ω and $A^\infty = A^* \cup A^\omega$ is the set of finite and infinite words. The empty word is ε and we have $\{\varepsilon\} = \emptyset^\infty$. For a word α and a position x of the word, $\alpha(x)$ is the x-th letter of α. By $|\alpha| \in \mathbb{N} \cup \{\infty\}$ we denote the *length* of α. Therefore $\alpha = \alpha(1) \cdots \alpha(|\alpha|)$ if α is finite and $\alpha = \alpha(1)\alpha(2)\cdots$ if α is infinite. We call $\mathrm{alph}(\alpha)$ the *alphabet* of α, i.e., the set of letters occurring in α. For $a \in \Gamma$, a position labeled by a is called an *a-position*. By $\mathrm{im}(\alpha)$ we mean the *imaginary* alphabet of α, i.e., the set of letters occurring infinitely often in α. For $A \subseteq \Gamma$, the set of words with imaginary alphabet A is denoted by A^{im}. In particular, $\Gamma^* = \emptyset^{\mathrm{im}}$. A *monomial* (of *degree* k) is a language of the form $A_1^* a_1 \cdots A_k^* a_k A_{k+1}^\infty$ for letters $a_i \in \Gamma$ and sets $A_i \subseteq \Gamma$. It is *unambiguous* if each word of the monomial has a unique factorization $u_1 a_1 \cdots u_k a_k \beta$ with $u_i \in A_i^*$ and $\beta \in A_{k+1}^\infty$. A *polynomial* is a finite union of monomials. It is called *unambiguous* if it is a finite union of unambiguous monomials.

We denote by FO = FO[<] the first-order logic over words interpreted as labeled linear orders (without ∞). As atomic formulas, FO comprises $\top$ (for *true*), the unary predicate $\lambda(x) = a$ for $a \in \Gamma$, and the binary predicate $x < y$ for variables x and y. The idea is that variables range over the linearly ordered positions of a word, and $\lambda(x) = a$ means that x is an a-position. Apart from the Boolean connectives, we allow composition of formulas using existential quantification $\exists x \colon \varphi$ and universal quantification $\forall x \colon \varphi$ for $\varphi \in$ FO. The semantics is as usual. Every formula in FO can be converted into a semantically equivalent formula in prenex normal form by renaming variables and moving quantifiers to the front. This observation gives rise to the fragment Σ_2 (resp. Π_2) consisting of all FO-formulas in prenex normal form with only two blocks of quantifiers, starting with a block of existential quantifiers (resp. universal quantifiers). Note that the negation of a formula in Σ_2 is equivalent to a formula in Π_2 and vice versa. The fragments Σ_2 and Π_2 are both closed under conjunction and disjunction. Furthermore, FO^2 is the fragment of FO containing all formulas which use at most two different names for the variables. This is a natural restriction, since FO with three variables already has the full expressive power of FO. A *sentence* in FO is a formula without free variables. The *language defined by* φ, denoted by $L(\varphi)$, is the set of words $\alpha \in \Gamma^\infty$ for which φ is true. We frequently identify logical fragments with the classes of languages they define (as in the definition of the fragment $\Delta_2 = \Sigma_2 \cap \Pi_2$ for example).

Example 1. Consider the formulas

$$\varphi \;=\; \exists x \forall y \colon y \leq x \lor \lambda(y) \neq a \qquad \text{and} \qquad \psi \;=\; \forall x \exists y \colon y > x \land \lambda(y) = a.$$

The formula $\varphi \in \Sigma_2 \cap \mathrm{FO}^2$ states that after some position there is no a-position, i.e., $L(\varphi)$ contains all words with finitely many a-positions. Its negation $\psi \in \Pi_2 \cap \mathrm{FO}^2$ says that for all positions there is a greater a-position, i.e., $L(\psi)$ is set of all words α with $a \in \mathrm{im}(\alpha)$. Surprisingly, $L(\varphi)$ is not definable in Π_2, while $L(\psi)$ is not definable in Σ_2, cf. [3]. $\Diamond$

3 Rankers and Unambiguous Temporal Logics

For finite words, rankers have been introduced by Immerman and Weis [10]. They can be seen as a generalization of *turtle programs* used by Schwentick, Thérien, and Vollmer [7] for characterizing FO^2-definable languages over finite words. The main difference between rankers and turtle programs is that rankers either uniquely determine a position in a word or they are undefined, whereas turtle programs mainly distinguish between being defined and being undefined.

$$\begin{tikzcd}[row sep=tiny] a_1 & a_2 & a_3 & & \mathrm{im}(\alpha) \\ 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & \cdots \arrow[r] & \infty \arrow[loop right] \end{tikzcd}$$

Fig. 1. Signature of $\alpha = a_1\, a_2\, a_3 \cdots$ over lazy rankers

Extending rankers with Boolean connectives yields unambiguous temporal logic (unambiguous TL). It is called *unambiguous* since each position considered by some formula in this logic is unique. Unambiguous TL has been introduced for Mazurkiewicz traces [4] which are a generalization of finite words.

All of our characterizations of first-order fragments rely on so-called unambiguous polynomials. A natural intermediate step from polynomials to temporal logic is interval temporal logic. Unambiguous interval temporal logic (unambiguous ITL) has been introduced by Lodaya, Pandya, and Shah [5] for finite words. They showed that over finite words it has the same expressive power as FO^2.

In this section, we generalize all three concepts (rankers, unambiguous TL, and unambiguous ITL) to infinite words. In fact, for each concept we shall give two natural generalizations. Surprisingly, it turns out that one of the two extensions can be used for the characterization of the first-order fragment $\Sigma_2 \cap \mathrm{FO}^2$ over Γ^∞ while the other yields a characterization of $\Pi_2 \cap \mathrm{FO}^2$. Moreover, both semantics can be used to describe FO^2 and Δ_2. In fact, for Δ_2 we use some fragment of rankers which conceals the difference between the two versions.

3.1 Rankers

A *ranker* is a finite word over the alphabet $\{\mathsf{X}_a, \mathsf{Y}_a \mid a \in \Gamma\}$. It can be interpreted as a sequence of instructions of the form X_a and Y_a. Here, X_a (for ne$\mathsf{X}$t-a) means "go to the next a-position" and Y_a (for Yesterday-a) means "go to the previous a-position". Below, we shall introduce a second variant of rankers called lazy rankers. If we want to emphasize the usage of this first version of rankers we refer to *eager rankers*. For a word α and a position $x \in \mathbb{N} \cup \{\infty\}$ we define

$$\begin{aligned} \mathsf{X}_a(\alpha, x) &= \min \{y \in \mathbb{N} \mid \alpha(y) = a \text{ and } y > x\}, \\ \mathsf{Y}_a(\alpha, x) &= \max \{y \in \mathbb{N} \mid \alpha(y) = a \text{ and } y < x\}. \end{aligned}$$

As usual, we set $y < \infty$ for all $y \in \mathbb{N}$. The minimum and the maximum of $\emptyset$ as well as the maximum of an infinite set are undefined. In particular, $\mathsf{X}_a(\alpha, \infty)$ is always undefined and $\mathsf{Y}_a(\alpha, \infty)$ is defined if and only if $a \in \mathrm{alph}(\alpha) \setminus \mathrm{im}(\alpha)$. We extend this definition to rankers by setting $\mathsf{X}_a\, r(\alpha, x) = r(\alpha, \mathsf{X}_a(\alpha, x))$ and $\mathsf{Y}_a\, r(\alpha, x) = r(\alpha, \mathsf{Y}_a(\alpha, x))$, i.e., rankers are processed from left to right. If $r(\alpha, x)$ is defined for some non-empty ranker r, then $r(\alpha, x) \neq \infty$.

Next, we define another variant of rankers as finite words over the alphabet $\{\mathsf{X}^\ell_a, \mathsf{Y}^\ell_a \mid a \in \Gamma\}$. The superscript ℓ is derived from *lazy*, and such rankers are called *lazy rankers*, accordingly. The difference to eager rankers is that lazy rankers can point to an infinite position ∞. The idea is that the position ∞ is not reachable from any finite position and that it represents the behavior at infinity. We imagine that ∞ is labeled by all letters in $\mathrm{im}(\alpha)$ for words α. Therefore, it

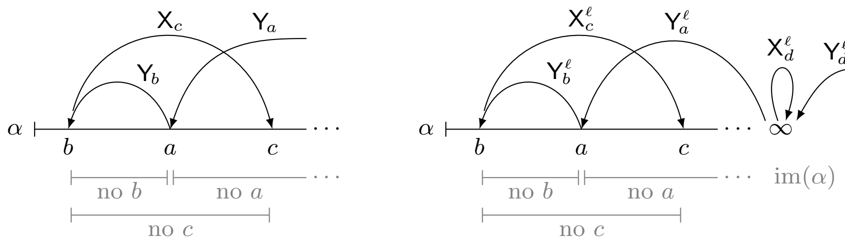


Fig. 2. An eager and a lazy ranker

is often adequate to set $\infty < \infty$, since the infinite position simulates a set of finite positions, see Fig. 1. For a word α and a finite position $x \in \mathbb{N}$ we define $\mathsf{X}^\ell_a(\alpha, x) = \mathsf{X}_a(\alpha, x)$ and $\mathsf{Y}^\ell_a(\alpha, x) = \mathsf{Y}_a(\alpha, x)$. For the infinite position we set

$$\mathsf{X}^\ell_a(\alpha,\infty) = \begin{cases} \infty & \text{if } a \in \mathrm{im}(\alpha) \\ \textit{undefined} & \text{else} \end{cases}$$

$$\mathsf{Y}^\ell_a(\alpha,\infty) = \begin{cases} \infty & \text{if } a \in \mathrm{im}(\alpha) \\ \mathsf{Y}_a(\alpha,\infty) & \text{else} \end{cases}$$

i.e., $\mathsf{Y}^\ell_a(\alpha,\infty)$ is undefined if $a \notin \mathrm{alph}(\alpha)$, and $\mathsf{Y}^\ell_a(\alpha,\infty) = \mathsf{Y}_a(\alpha,\infty)$ is a finite position if $a \in \mathrm{alph}(\alpha) \setminus \mathrm{im}(\alpha)$. As before, we extend this definition to rankers by setting $\mathsf{X}^\ell_a\, r(\alpha, x) = r(\alpha, \mathsf{X}^\ell_a(\alpha, x))$ and $\mathsf{Y}^\ell_a\, r(\alpha, x) = r(\alpha, \mathsf{Y}^\ell_a(\alpha, x))$. We denote by $\mathrm{alph}_\Gamma(r)$ the set of letters $a \in \Gamma$ such that r contains a modality using the letter a. It can happen that $r(\alpha,\infty) = \infty$ for some non-empty lazy ranker r. This is the case if and only if r is of the form $\mathsf{Y}^\ell_a\, s$ and $\mathrm{alph}_\Gamma(r) \subseteq \mathrm{im}(\alpha)$.

If the reference to the word α is clear from the context, then for eager and lazy rankers r we shorten the notation and write $r(x)$ instead of $r(\alpha, x)$.

An eager ranker r is an X-*ranker* if $r = \mathsf{X}_a\, s$ for some ranker s and $a \in \Gamma$, and it is a Y-*ranker* if r is of the form $\mathsf{Y}_a\, s$. Lazy X^ℓ-rankers and Y^ℓ-rankers are defined similarly. We proceed to define $r(\alpha)$, the position of α reached by the ranker r by starting "outside" the word α. The intuition is as follows. If r is an X-ranker or an X^ℓ-ranker, we imagine that we start at an outside position in front of α; if r is a Y-ranker or a Y^ℓ-ranker, then we start at a position behind α. Therefore, we define

$$\begin{aligned} r(\alpha) &= r(\alpha, 0) && \text{if } r \text{ is an } \mathsf{X}\text{-ranker or an } \mathsf{X}^\ell\text{-ranker,} \\ r(\alpha) &= r(\alpha, \infty) && \text{if } r \text{ is a } \mathsf{Y}\text{-ranker or a } \mathsf{Y}^\ell\text{-ranker.} \end{aligned}$$

On the left hand side of Fig. 2, a possible situation for the eager ranker $\mathsf{Y}_a\, \mathsf{Y}_b\, \mathsf{X}_c$ being defined on some word α is depicted. The right hand side of the same figure illustrates a similar situation for the lazy ranker $\mathsf{Y}^\ell_d\, \mathsf{X}^\ell_d\, \mathsf{Y}^\ell_a\, \mathsf{Y}^\ell_b\, \mathsf{X}^\ell_c$ with $d \in \mathrm{im}(\alpha)$ and $a \in \mathrm{alph}(\alpha) \setminus \mathrm{im}(\alpha)$. Note that the eager version of the same ranker is not defined on α since $d \in \mathrm{im}(\alpha)$.

For an eager or lazy ranker r the language $L(r)$ generated by r is the set of all words in Γ^∞ on which r is defined. A *(positive) ranker language* is a finite (positive) Boolean combination of languages of the form $L(r)$ for eager rankers r. A *(positive) lazy ranker language* is a finite (positive) Boolean combination of languages of the form $L(r)$ for lazy rankers r. Finally, a *(positive)* X*-ranker language* is a (positive) ranker language using only X-rankers. At the end of the next section, we extend rankers by some atomic modalities.

3.2 Unambiguous Temporal Logic

Our generalization of rankers allows us to define unambiguous temporal logic (unambiguous TL) over infinite words. As for rankers, we have an eager and a lazy variant. The syntax is given by:

$$\top \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \mathsf{X}_a\,\varphi \mid \mathsf{Y}_a\,\varphi \mid \mathsf{G}_{\bar{a}} \mid \mathsf{H}_{\bar{a}} \mid \mathsf{X}^{\ell}_a\,\varphi \mid \mathsf{Y}^{\ell}_a\,\varphi \mid \mathsf{G}^{\ell}_{\bar{a}} \mid \mathsf{H}^{\ell}_{\bar{a}}$$

for $a \in \Gamma$ and formulas φ, ψ in unambiguous TL. The atomic formulas are $\top$ (which is *true*), and the eager modalities $\mathsf{G}_{\bar{a}}$ (for Globally-no-a) and $\mathsf{H}_{\bar{a}}$ (for Historically-no-a), as well as the lazy modalities $\mathsf{G}^{\ell}_{\bar{a}}$ (for lazy-Globally-no-a) and $\mathsf{H}^{\ell}_{\bar{a}}$ (for lazy-Historically-no-a). We now define, when a word α with a position $x \in \mathbb{N} \cup \{\infty\}$ satisfies a formula φ in unambiguous TL, denoted by $\alpha, x \models \varphi$. The atomic formula $\top$ is true for all positions, and the semantics of the Boolean connectives is as usual. For $\mathsf{Z} \in \{\mathsf{X}_a, \mathsf{Y}_a, \mathsf{X}^{\ell}_a, \mathsf{Y}^{\ell}_a \mid a \in \Gamma\}$ we define

$$\alpha, x \models \mathsf{Z}\,\varphi \quad \text{iff} \quad \mathsf{Z}(x) \text{ is defined and } \alpha, \mathsf{Z}(x) \models \varphi.$$

The semantics of the atomic modalities is given by

$$\mathsf{G}_{\bar{a}} \;=\; \neg\,\mathsf{X}_a\,\top, \quad \mathsf{H}_{\bar{a}} \;=\; \neg\,\mathsf{Y}_a\,\top, \quad \mathsf{G}^{\ell}_{\bar{a}} \;=\; \neg\,\mathsf{X}^{\ell}_a\,\top, \quad \mathsf{H}^{\ell}_{\bar{a}} \;=\; \neg\,\mathsf{Y}^{\ell}_a\,\top.$$

In order to define when a word α models a formula φ, we have to distinguish whether φ starts with a future or with a past modality:

$$\begin{array}{llllll}
\alpha \models \mathsf{X}_a\,\varphi & \text{iff} & \alpha, 0 \models \mathsf{X}_a\,\varphi, & \alpha \models \mathsf{Y}_a\,\varphi & \text{iff} & \alpha, \infty \models \mathsf{Y}_a\,\varphi, \\
\alpha \models \mathsf{G}_{\bar{a}} & \text{iff} & \alpha, 0 \models \mathsf{G}_{\bar{a}}, & \alpha \models \mathsf{H}_{\bar{a}} & \text{iff} & \alpha, \infty \models \mathsf{H}_{\bar{a}}, \\
\alpha \models \mathsf{X}^{\ell}_a\,\varphi & \text{iff} & \alpha, 0 \models \mathsf{X}^{\ell}_a\,\varphi, & \alpha \models \mathsf{Y}^{\ell}_a\,\varphi & \text{iff} & \alpha, \infty \models \mathsf{Y}^{\ell}_a\,\varphi, \\
\alpha \models \mathsf{G}^{\ell}_{\bar{a}} & \text{iff} & \alpha, 0 \models \mathsf{G}^{\ell}_{\bar{a}}, & \alpha \models \mathsf{H}^{\ell}_{\bar{a}} & \text{iff} & \alpha, \infty \models \mathsf{H}^{\ell}_{\bar{a}}\,.
\end{array}$$

The modalities on the left are called *future modalities*, while the modalities on the right are called *past modalities.* The atomic modalities $\mathsf{G}_{\bar{a}}$ and $\mathsf{G}^{\ell}_{\bar{a}}$ differ only for the infinite position, but the semantics of $\mathsf{H}_{\bar{a}}$ and $\mathsf{H}^{\ell}_{\bar{a}}$ differs a lot: $\alpha \models \mathsf{H}_{\bar{a}}$ if and only if $a \in \mathrm{im}(\alpha)$ or $a \notin \mathrm{alph}(\alpha)$ whereas $\alpha \models \mathsf{H}^{\ell}_{\bar{a}}$ if and only if $a \notin \mathrm{alph}(\alpha)$. Every formula φ defines a language $L(\varphi) = \{\alpha \in \Gamma^{\infty} \mid \alpha \models \varphi\}$.

Finally, for $\mathcal{C} \subseteq \{\mathsf{X}_a, \mathsf{Y}_a, \mathsf{G}_{\bar{a}}, \mathsf{H}_{\bar{a}}, \mathsf{X}^{\ell}_a, \mathsf{Y}^{\ell}_a, \mathsf{G}^{\ell}_{\bar{a}}, \mathsf{H}^{\ell}_{\bar{a}}\}$ we define the following fragments of TL:

- TL[$\mathcal{C}$] consists of all formulas using only $\top$, Boolean connectives, and temporal modalities in $\mathcal{C}$,
- TL^{+}[$\mathcal{C}$] consists of all formulas using only $\top$, positive Boolean connectives (i.e., no negation), and temporal modalities in $\mathcal{C}$,

– $\mathrm{TL}_{\mathsf{X}}[\mathcal{C}]$ consists of all formulas using only $\top$, Boolean connectives, and temporal modalities in $\mathcal{C}$ such that all outmost modalities are future modalities,
– $\mathrm{TL}^{+}_{\mathsf{X}}[\mathcal{C}]$ consists of all formulas in $\mathrm{TL}^{+}[\mathcal{C}] \cap \mathrm{TL}_{\mathsf{X}}[\mathcal{C}]$.

Example 2. Consider the language $L \subseteq \Gamma^{\infty}$ consisting of all non-empty words with a as the first letter. This language is defined by each of following formulas:

$$\begin{aligned}
\varphi_1 &= \mathsf{X}_a \top \wedge \bigwedge_{b \in \Gamma} \neg \mathsf{X}_a \mathsf{Y}_b \top && \in \mathrm{TL}_{\mathsf{X}}[\mathsf{X}_a, \mathsf{Y}_a],\\
\varphi_2 &= \bigwedge_{b \in \Gamma} \mathsf{X}_a \mathsf{H}_{\bar{b}} && \in \mathrm{TL}^{+}_{\mathsf{X}}[\mathsf{X}_a, \mathsf{H}_{\bar{a}}],\\
\varphi_3 &= \mathsf{X}_a \top \wedge \bigwedge_{b \in \Gamma \setminus \{a\}} \big(\mathsf{G}_{\bar{b}} \vee \mathsf{X}_b \mathsf{Y}_a \top\big) && \in \mathrm{TL}^{+}_{\mathsf{X}}[\mathsf{X}_a, \mathsf{Y}_a, \mathsf{G}_{\bar{a}}]. \qquad \Diamond
\end{aligned}$$

Inspired by the atomic logical modalities, we extend the notion of a ranker by allowing the atomic modalities $\mathsf{G}_{\bar{a}}$ and $\mathsf{H}_{\bar{a}}$ as well as $\mathsf{G}^{\ell}_{\bar{a}}$ and $\mathsf{H}^{\ell}_{\bar{a}}$. We call r a *ranker with atomic modality* $\mathsf{G}_{\bar{a}}$ ($\mathsf{H}_{\bar{a}}$, $\mathsf{G}^{\ell}_{\bar{a}}$, $\mathsf{H}^{\ell}_{\bar{a}}$, resp.) if $r = s\,\mathsf{G}_{\bar{a}}$ ($r = s\,\mathsf{H}_{\bar{a}}$, $r = s\,\mathsf{G}^{\ell}_{\bar{a}}$, $r = s\,\mathsf{H}^{\ell}_{\bar{a}}$, resp.) for some ranker s. In this setting, $r = \mathsf{G}_{\bar{a}}$ is an X-ranker, and $r = \mathsf{H}_{\bar{a}}$ is a Y-ranker. Similarly, $r = \mathsf{G}^{\ell}_{\bar{a}}$ is an X^{ℓ}-ranker, and $r = \mathsf{H}^{\ell}_{\bar{a}}$ is a Y^{ℓ}-ranker. Note that any ranker with some atomic modality is also a formula in unambiguous TL. We can therefore define the domain of an extended ranker r with some atomic modality by

$$r(\alpha, x) \text{ is defined} \qquad \text{iff} \qquad \alpha, x \models r.$$

If $r \in s\,\{\mathsf{G}_{\bar{a}}, \mathsf{H}_{\bar{a}}, \mathsf{G}^{\ell}_{\bar{a}}, \mathsf{H}^{\ell}_{\bar{a}} \mid a \in \Gamma\}$ is an extended ranker with $r(\alpha, x)$ being defined, then we set $r(\alpha, x) = s(\alpha, x)$, i.e., $r(\alpha, x)$ is the position reached after the execution of s. The reinterpretation of rankers as formulas also makes sense for a ranker $r \in \{\mathsf{X}_a, \mathsf{Y}_a, \mathsf{X}^{\ell}_a, \mathsf{Y}^{\ell}_a\}^{*}$ without atomic modality by identifying r with $r\top$ in unambiguous TL. This is justified since r is defined on α if and only if $\alpha \models r\top$.

Let $\mathcal{C} \subseteq \{\mathsf{G}_{\bar{a}}, \mathsf{H}_{\bar{a}}, \mathsf{G}^{\ell}_{\bar{a}}, \mathsf{H}^{\ell}_{\bar{a}}\}$. A language is a *ranker language with atomic modalities* $\mathcal{C}$ if it is a Boolean combination of languages $L(r)$ such that r is either a ranker without atomic modalities or a ranker with some atomic modality in $\mathcal{C}$. Similarly, the notions of lazy / positive / X-ranker languages are adapted to the use of atomic modalities.

3.3 Unambiguous Interval Temporal Logic

We extend unambiguous interval temporal logic (unambiguous ITL) to infinite words in such a way that it coincides with FO^2. Again, we have two extensions with this property, one being eager and one being lazy. The syntax of unambiguous ITL is given by Boolean combinations and:

$$\top \mid \varphi\, \mathsf{F}_a\, \psi \mid \varphi\, \mathsf{L}_a\, \psi \mid \mathsf{G}_{\bar{a}} \mid \mathsf{H}_{\bar{a}} \mid \varphi\, \mathsf{F}^{\ell}_a\, \psi \mid \varphi\, \mathsf{L}^{\ell}_a\, \psi \mid \mathsf{G}^{\ell}_{\bar{a}} \mid \mathsf{H}^{\ell}_{\bar{a}}$$

with $a \in \Gamma$ and formulas φ, ψ in unambiguous ITL. The name F_a derives from "First-a" and L_a from "Last-a". As in unambiguous temporal logic, the atomic formulas are $\top$, the eager modalities $\mathsf{G}_{\bar{a}}$ and $\mathsf{H}_{\bar{a}}$, and the lazy modalities $\mathsf{G}^{\ell}_{\bar{a}}$ and $\mathsf{H}^{\ell}_{\bar{a}}$. We now define, when a word α together with an interval $(x; y) = \{z \in \mathbb{N} \cup \{\infty\} \mid x < z < y\}$ satisfies a formula φ in unambiguous ITL, denoted by $\alpha, (x; y) \models \varphi$. Remember that we have set $\infty < \infty$. In particular $(\infty; \infty) = \{\infty\}$. The atomic formula $\top$ is true for all intervals, and the semantics of the Boolean connectives is as usual. The semantics of the binary modalities is as follows:

$$\begin{aligned}
\alpha, (x; y) \models \varphi \, \mathsf{F}_a \, \psi \quad & \text{iff} \quad \mathsf{X}_a(x) \text{ is defined, } \mathsf{X}_a(x) < y, \\
& \qquad \alpha, \big(x; \mathsf{X}_a(x)\big) \models \varphi \ \text{ and } \ \alpha, \big(\mathsf{X}_a(x); y\big) \models \psi, \\
\alpha, (x; y) \models \varphi \, \mathsf{L}_a \, \psi \quad & \text{iff} \quad \mathsf{Y}_a(y) \text{ is defined, } \mathsf{Y}_a(y) > x, \\
& \qquad \alpha, \big(x; \mathsf{Y}_a(y)\big) \models \varphi \ \text{ and } \ \alpha, \big(\mathsf{Y}_a(y); y\big) \models \psi.
\end{aligned}$$

The semantics of F^{ℓ}_a and L^{ℓ}_a is defined analogously using X^{ℓ}_a and Y^{ℓ}_a, respectively. The semantics of the atomic modalities is given by

$$\begin{aligned}
\mathsf{G}_{\bar{a}} &= \neg(\top \, \mathsf{F}_a \, \top), & \mathsf{H}_{\bar{a}} &= \neg(\top \, \mathsf{L}_a \, \top), \\
\mathsf{G}^{\ell}_{\bar{a}} &= \neg(\top \, \mathsf{F}^{\ell}_a \, \top), & \mathsf{H}^{\ell}_{\bar{a}} &= \neg(\top \, \mathsf{L}^{\ell}_a \, \top) \vee \bigvee_{b \in \Gamma} ((\top \, \mathsf{L}^{\ell}_b \, \top) \, \mathsf{F}^{\ell}_b \, \top).
\end{aligned}$$

In the definition of $\mathsf{H}^{\ell}_{\bar{a}}$, the disjunction on the right-hand side ensures that $\alpha, (\infty; \infty) \models \mathsf{H}^{\ell}_{\bar{a}}$ for every infinite word $\alpha \in \Gamma^{\omega}$ and every $a \in \Gamma$. It will turn out that the inability of specifying the letters not in $\operatorname{im}(\alpha)$ is crucial in the characterization of the fragment $\Pi_2 \cap \mathrm{FO}^2$. Observe that only for the interval $(\infty; \infty)$, there can be a b before the "first" b. Also note that for every finite interval, the formula $\mathsf{G}_{\bar{a}}$ is true if and only if $\mathsf{H}_{\bar{a}}$ is true and that $\mathsf{G}^{\ell}_{\bar{a}}$ is equivalent to $\neg(\top \, \mathsf{L}^{\ell}_a \, \top)$. Whether a word α models a formula φ in unambiguous ITL (i.e., $\alpha \models \varphi$) is defined by

$$\alpha \models \varphi \quad \text{iff} \quad \alpha, (0; \infty) \models \varphi.$$

The language defined by φ is $L(\varphi) = \{\alpha \in \Gamma^{\infty} \mid \alpha \models \varphi\}$.

Fig. 3 depicts the situation for the formula $(\varphi_1 \, \mathsf{F}_b \, \psi_1) \, \mathsf{L}_a \, (\varphi_2 \, \mathsf{F}_c \, \psi_2)$ being defined on α. The main difference to rankers and unambiguous TL is that there is no crossing over in unambiguous ITL, e.g., in the situation depicted on the left side of Fig. 2, the formula $(\top \, \mathsf{L}_b \, (\top \, \mathsf{F}_c \, \top)) \, \mathsf{L}_a \, \top$ is false even though the ranker $\mathsf{Y}_a \mathsf{Y}_b \mathsf{X}_c$ is defined.

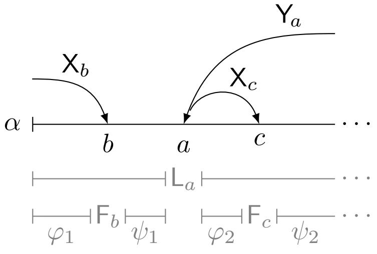


Fig. 3.

In unambiguous ITL, the modalities F_a, $\mathsf{G}_{\bar{a}}$, F^{ℓ}_a, $\mathsf{G}^{\ell}_{\bar{a}}$ are *future modalities* and L_a, $\mathsf{H}_{\bar{a}}$, L^{ℓ}_a, $\mathsf{H}^{\ell}_{\bar{a}}$ are *past modalities*. A formula φ is a *future formula* if in the parse tree of φ, every past modality occurs on the left branch of some future modality, i.e., if it is never necessary to interpret a past modality over an unbounded interval.

For $\mathcal{C} \subseteq \{\mathsf{F}_a, \mathsf{L}_a, \mathsf{G}_{\bar{a}}, \mathsf{H}_{\bar{a}}, \mathsf{F}^{\ell}_a, \mathsf{L}^{\ell}_a, \mathsf{G}^{\ell}_{\bar{a}}, \mathsf{H}^{\ell}_{\bar{a}}\}$ we define the following fragments of ITL:

– $\mathrm{ITL}[\mathcal{C}]$ consists of all formulas using only $\top$, Boolean connectives, and temporal modalities in $\mathcal{C}$,
– $\mathrm{ITL}^+[\mathcal{C}]$ consists of all formulas using only $\top$, positive Boolean connectives (i.e., no negation), and temporal modalities in $\mathcal{C}$,
– $\mathrm{ITL}_{\mathsf{F}}[\mathcal{C}]$ consists of all future formulas using only $\top$, Boolean connectives, and temporal modalities in $\mathcal{C}$,
– $\mathrm{ITL}^+_{\mathsf{F}}[\mathcal{C}]$ consists of all formulas in $\mathrm{ITL}^+[\mathcal{C}] \cap \mathrm{ITL}_{\mathsf{F}}[\mathcal{C}]$.

The proofs of the following two propositions give a procedure for converting unambiguous ITL formulas into unambiguous TL formulas without introducing new negations. A similar relativization technique as in our proof has been used by Lodaya, Pandya, and Shah [5] for the conversion of ITL over finite words into so-called *deterministic partially ordered two-way automata* (without the focus on not introducing negations).

Proposition 1. *We have the following inclusions:*

$$\begin{aligned}
\mathrm{ITL}[\mathsf{F}_a, \mathsf{L}_a] &\subseteq \mathrm{TL}[\mathsf{X}_a, \mathsf{Y}_a],\\
\mathrm{ITL}^+[\mathsf{F}_a, \mathsf{L}_a, \mathsf{G}_{\bar{a}}, \mathsf{H}_{\bar{a}}] &\subseteq \mathrm{TL}^+[\mathsf{X}_a, \mathsf{Y}_a, \mathsf{G}_{\bar{a}}, \mathsf{H}_{\bar{a}}],\\
\mathrm{ITL}^+[\mathsf{F}_a, \mathsf{L}_a, \mathsf{G}_{\bar{a}}] &\subseteq \mathrm{TL}^+[\mathsf{X}_a, \mathsf{Y}_a, \mathsf{G}_{\bar{a}}],\\
\mathrm{ITL}^+_{\mathsf{F}}[\mathsf{F}_a, \mathsf{L}_a, \mathsf{G}_{\bar{a}}, \mathsf{H}_{\bar{a}}] &\subseteq \mathrm{TL}^+_{\mathsf{X}}[\mathsf{X}_a, \mathsf{Y}_a, \mathsf{G}_{\bar{a}}],\\
\mathrm{ITL}_{\mathsf{F}}[\mathsf{F}_a, \mathsf{L}_a] &\subseteq \mathrm{TL}_{\mathsf{X}}[\mathsf{X}_a, \mathsf{Y}_a].
\end{aligned}$$

Proposition 2. *We have the following inclusions:*

$$\begin{aligned}
\mathrm{ITL}[\mathsf{F}^{\ell}_a, \mathsf{L}^{\ell}_a] &\subseteq \mathrm{TL}[\mathsf{X}^{\ell}_a, \mathsf{Y}^{\ell}_a],\\
\mathrm{ITL}^+[\mathsf{F}^{\ell}_a, \mathsf{L}^{\ell}_a, \mathsf{G}^{\ell}_{\bar{a}}, \mathsf{H}^{\ell}_{\bar{a}}] &\subseteq \mathrm{TL}^+[\mathsf{X}^{\ell}_a, \mathsf{Y}^{\ell}_a, \mathsf{G}^{\ell}_{\bar{a}}, \mathsf{H}^{\ell}_{\bar{a}}],\\
\mathrm{ITL}^+[\mathsf{F}^{\ell}_a, \mathsf{L}^{\ell}_a, \mathsf{H}^{\ell}_{\bar{a}}] &\subseteq \mathrm{TL}^+[\mathsf{X}^{\ell}_a, \mathsf{Y}^{\ell}_a, \mathsf{H}^{\ell}_{\bar{a}}].
\end{aligned}$$

4 Main Results

We start this section with various ITL, TL, and ranker characterizations using the eager variants. We postpone characterizations in terms of the lazy fragments to Theorem 4 and Theorem 5.

Theorem 1. *For $L \subseteq \Gamma^{\infty}$ the following assertions are equivalent:*

1. *L is definable in* FO^2.
2. *L is definable in* $\mathrm{ITL}^+[\mathsf{F}_a, \mathsf{L}_a, \mathsf{G}_{\bar{a}}, \mathsf{H}_{\bar{a}}]$.
3. *L is definable in* $\mathrm{ITL}[\mathsf{F}_a, \mathsf{L}_a]$.
4. *L is definable in* $\mathrm{TL}[\mathsf{X}_a, \mathsf{Y}_a]$.
5. *L is definable in* $\mathrm{TL}^+[\mathsf{X}_a, \mathsf{Y}_a, \mathsf{G}_{\bar{a}}, \mathsf{H}_{\bar{a}}]$.
6. *L is a positive ranker language with atomic modalities* $\mathsf{G}_{\bar{a}}$ *and* $\mathsf{H}_{\bar{a}}$.
7. *L is a ranker language.*

Every FO^2-definable language is a Boolean combination of unambiguous monomials and languages of the form A^{im}, see [3]. The language A^{im} is definable by the formula

$$\bigwedge_{a \in A} (\top\ \mathsf{F}_a\ \top) \wedge \mathsf{H}_{\bar{a}} \quad \in \mathrm{ITL}^{+}[\mathsf{F}_a, \mathsf{H}_{\bar{a}}].$$

Hence, the following lemma provides the missing part in order to show that every language in FO^2 is definable in unambiguous ITL.

Lemma 1. *Every unambiguous monomial $L = A_1^* a_1 \cdots A_k^* a_k A_{k+1}^\infty$ is definable in* $\mathrm{ITL}^{+}[\mathsf{F}_a, \mathsf{L}_a, \mathsf{G}_{\bar{a}}]$.

Proof. We perform an induction on k. For $k = 0$ we have $L\big(\bigwedge_{a \notin A_1} \mathsf{G}_{\bar{a}}\big) = A_1^\infty$. Let $k \geq 1$. Since L is unambiguous, we have $\{a_1, \ldots, a_k\} \not\subseteq A_1 \cap A_{k+1}$; otherwise $(a_1 \cdots a_k)^2$ admits two different factorizations showing that L is not unambiguous. First, consider the case $a_i \notin A_1$ and let i be minimal with this property. Each word $\alpha \in L$ has a unique factorization $\alpha = u a_i \beta$ such that $a_i \notin \mathrm{alph}(u)$. Depending on whether the first a_i of α coincides with the marker a_i or not, we have

$$\begin{array}{lll} u \in A_1^* a_1 \cdots A_i^*, & & \beta \in A_{i+1}^* a_{i+1} \cdots A_k^* a_k A_{k+1}^\infty \quad \text{or} \\ u \in A_1^* a_1 \cdots A_j^*, & a_i \in A_j, & \beta \in A_j^* a_j \cdots A_k^* a_k A_{k+1}^\infty \end{array}$$

with $2 \leq j \leq i$. In both cases, since L is unambiguous, each expression containing u or β is unambiguous. Moreover, each of these expressions is strictly shorter than L. By induction, for each $2 \leq j \leq k$, there exist formulas $\varphi, \psi \in \mathrm{ITL}^{+}[\mathsf{F}_a, \mathsf{L}_a, \mathsf{G}_{\bar{a}}]$ such that $L(\varphi) = A_1^* a_1 \cdots A_j^\infty$ and $L(\psi) = A_j^* a_j \cdots A_k^* a_k A_{k+1}^\infty$. By the above reasoning, we see that L is the union of (at most i) languages of the form

$$\big(L(\varphi) \cap (\Gamma \setminus \{a_i\})^*\big)\, a_i\, L(\psi)$$

and each of them is defined by $\varphi\ \mathsf{F}_{a_i}\ \psi$.

For $a_i \notin A_{k+1}$ with i maximal, we consider the unique factorization $\alpha = u a_i \beta$ with $a_i \notin \mathrm{alph}(\beta)$ and, again, we end up with one of the two cases from above, with the difference that $1 \leq i < j \leq k$ in the second case. Inductively L is defined by a disjunction of formulas $\varphi\ \mathsf{L}_{a_i}\ \psi$. ☐

Theorem 2. *Let $L \subseteq \Gamma^\infty$. The following assertions are equivalent:*

1. *L is definable in Σ_2 and* FO^2.
2. *L is definable in* $\mathrm{ITL}^{+}[\mathsf{F}_a, \mathsf{L}_a, \mathsf{G}_{\bar{a}}]$.
3. *L is definable in* $\mathrm{TL}^{+}[\mathsf{X}_a, \mathsf{Y}_a, \mathsf{G}_{\bar{a}}]$.
4. *L is a positive ranker language with atomic modality $\mathsf{G}_{\bar{a}}$.*

Theorem 5 shows that the same characterizations for FO^2 hold using the lazy variants. Note that we cannot use lazy counterparts in the characterizations for $\Sigma_2 \cap \mathrm{FO}^2$, since for example $\mathsf{Y}_a^\ell\, \mathsf{X}_a^\ell$ is defined if and only if there are infinitely many a's, but this property is not Σ_2-definable.

Over finite words, the fragments FO^2 and Δ_2 coincide [9]. In particular, $\mathrm{FO}^2 \cap \Sigma_2 = \mathrm{FO}^2$ over finite words. Since finiteness of a word is definable in $\mathrm{FO}^2 \cap \Sigma_2$, we obtain the following corollary of Theorem 2.

Corollary 1. *A language $L \subseteq \Gamma^*$ of finite words is definable in* FO^2 *if and only if L is a positive ranker language with atomic modality $\mathsf{G}_{\bar{a}}$.*

Over infinite words, the fragment Δ_2 is a strict subclass of FO^2. The next theorem says that Δ_2 is basically FO^2 with the lack of past formulas and Y-rankers. Since eager future formulas and X-rankers coincide with their lazy counterparts, all of the characterizations in the next theorem could be replaced by their lazy pendants.

Theorem 3. *Let $L \subseteq \Gamma^\infty$. The following assertions are equivalent:*

1. *L is definable in Δ_2.*
2. *L is definable in $\mathrm{ITL}_{\mathsf{F}}^{+}[\mathsf{F}_a, \mathsf{L}_a, \mathsf{G}_{\bar{a}}]$.*
3. *L is definable in $\mathrm{ITL}_{\mathsf{F}}[\mathsf{F}_a, \mathsf{L}_a]$.*
4. *L is definable in $\mathrm{TL}_{\mathsf{X}}[\mathsf{X}_a, \mathsf{Y}_a]$.*
5. *L is definable in $\mathrm{TL}_{\mathsf{X}}^{+}[\mathsf{X}_a, \mathsf{Y}_a, \mathsf{G}_{\bar{a}}]$.*
6. *L is a positive X-ranker language with atomic modality $\mathsf{G}_{\bar{a}}$.*
7. *L is an X-ranker language.*

In the next theorem we give characterizations of the fragment $\Pi_2 \cap \mathrm{FO}^2$ in terms of the lazy variants of ITL, TL, and rankers. We cannot use the eager variants, since Y_a says that there are only finitely many a's, but this property is not Π_2-definable. Also note that $\alpha, (\infty;\infty) \models \hat{\mathsf{H}}_{\bar{a}}$ for $\hat{\mathsf{H}}_{\bar{a}} = \neg(\top\ \mathsf{L}_a^{\ell}\ \top)$ if and only if $a \notin \mathrm{im}(\alpha)$, i.e., if and only if a occurs at most finitely often. As before, this property is not Π_2-definable. This is the reason why we did not define $\mathsf{H}_{\bar{a}}^{\ell}$ simply as $\hat{\mathsf{H}}_{\bar{a}}$.

Theorem 4. *Let $L \subseteq \Gamma^\infty$. The following assertions are equivalent:*

1. *L is definable in Π_2 and FO^2.*
2. *L is definable in $\mathrm{ITL}^{+}[\mathsf{F}_a^{\ell}, \mathsf{L}_a^{\ell}, \mathsf{H}_{\bar{a}}^{\ell}]$.*
3. *L is definable in $\mathrm{TL}^{+}[\mathsf{X}_a^{\ell}, \mathsf{Y}_a^{\ell}, \mathsf{H}_{\bar{a}}^{\ell}]$.*
4. *L is a positive lazy ranker language with atomic modality $\mathsf{H}_{\bar{a}}^{\ell}$.*

For completeness, we give a counterpart of Theorem 1 using the lazy versions of ITL, TL, and rankers.

Theorem 5. *For $L \subseteq \Gamma^\infty$ the following assertions are equivalent:*

1. *L is definable in FO^2.*
2. *L is definable in $\mathrm{ITL}^{+}[\mathsf{F}_a^{\ell}, \mathsf{L}_a^{\ell}, \mathsf{G}_{\bar{a}}^{\ell}, \mathsf{H}_{\bar{a}}^{\ell}]$.*
3. *L is definable in $\mathrm{ITL}[\mathsf{F}_a^{\ell}, \mathsf{L}_a^{\ell}]$.*
4. *L is definable in $\mathrm{TL}[\mathsf{X}_a^{\ell}, \mathsf{Y}_a^{\ell}]$.*
5. *L is definable in $\mathrm{TL}^{+}[\mathsf{X}_a^{\ell}, \mathsf{Y}_a^{\ell}, \mathsf{G}_{\bar{a}}^{\ell}, \mathsf{H}_{\bar{a}}^{\ell}]$.*
6. *L is a positive ranker language with atomic modalities $\mathsf{G}_{\bar{a}}^{\ell}$ and $\mathsf{H}_{\bar{a}}^{\ell}$.*
7. *L is a lazy ranker language.*

5 Open Problems

Rankers over finite words have been introduced for characterizing quantifier alternation within FO^2. We conjecture that similar results for infinite words can be obtained using our generalizations of rankers.

Over infinite words, the class of X-ranker languages corresponds to the fragment Δ_2. Over finite words however, X-ranker languages form a strict subclass of Δ_2 (which for finite words coincides with FO^2). An algebraic counterpart of X-ranker languages over finite words is still missing. The main problem is that over finite words X-rankers do not define a variety of languages.

A well-known theorem by Schützenberger [6] implies that over finite words, arbitrary finite unions of unambiguous monomials and finite *disjoint* unions of unambiguous monomials describe the same class of languages. In the case of infinite words, it is open whether one can require that unambiguous polynomials are disjoint unions of unambiguous monomials without changing the class of languages.

Acknowledgments. We thank Volker Diekert for a suggestion which led to Theorem 2. We also thank the anonymous referees for several useful suggestions which helped to improve the presentation of this paper.

References

1. Dartois, L., Kufleitner, M., Lauser, A.: Rankers over Infinite Words. Technical report no. 2010/01. Universität Stuttgart, Informatik (2010)
2. Diekert, V., Gastin, P., Kufleitner, M.: A survey on small fragments of first-order logic over finite words. International Journal of Foundations of Computer Science 19(3), 513–548 (2008); Special issue DLT 2007
3. Diekert, V., Kufleitner, M.: Fragments of first-order logic over infinite words. In: Dagstuhl Seminar Proceedings of STACS 2009, vol. 09001, pp. 325–336 (2009)
4. Kufleitner, M.: Polynomials, fragments of temporal logic and the variety DA over traces. Theoretical Computer Science 376, 89–100 (2007); Special issue DLT 2006
5. Lodaya, K., Pandya, P.K., Shah, S.S.: Marking the chops: an unambiguous temporal logic. IFIP TCS 273, 461–476 (2008)
6. Schützenberger, M.P.: Sur le produit de concaténation non ambigu. Semigroup Forum 13, 47–75 (1976)
7. Schwentick, T., Thérien, D., Vollmer, H.: Partially-ordered two-way automata: A new characterization of DA. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 239–250. Springer, Heidelberg (2002)
8. Tesson, P., Thérien, D.: Diamonds are forever: The variety DA. In: Semigroups, Algorithms, Automata and Languages, pp. 475–500 (2002)
9. Thérien, D., Wilke, T.: Over words, two variables are as powerful as one quantifier alternation. In: Proceedings of STOC 1998, pp. 234–240 (1998)
10. Weis, P., Immerman, N.: Structure theorem and strict alternation hierarchy for FO^2 on words. Logical Methods in Computer Science 5(3:3), 1–23 (2009)

Kleene and Büchi Theorems for Weighted Automata and Multi-valued Logics over Arbitrary Bounded Lattices

Manfred Droste[1] and Heiko Vogler[2]

[1] Institute of Computer Science, Leipzig University, 04109 Leipzig, Germany
[2] Department of Computer Science, TU Dresden, 01062 Dresden, Germany
droste@informatik.uni-leipzig.de, Heiko.Vogler@tu-dresden.de

Abstract. We show that $\mathcal{L}$-weighted automata, $\mathcal{L}$-rational series, and $\mathcal{L}$-valued monadic second-order logic have the same expressive power, for any bounded lattice $\mathcal{L}$ and for finite and infinite words. This extends classical results of Kleene and Büchi to arbitrary bounded lattices, without any distributivity assumption that is fundamental in the theory of weighted automata over semirings. In fact, we obtain these results for large classes of strong bimonoids which properly contain all bounded lattices.

Introduction

Most of Mathematics and Computer Science are usually based on classical two-valued logic. However, already Łukasiewicz [33] and Post [40] investigated logics with different degrees of certainty, and Birkhoff and von Neumann [3] introduced quantum logic with values in orthomodular lattices as the logic of quantum mechanics. In general distributivity fails in quantum logics; therefore, orthomodular lattices are not required to satisfy the distributivity law. Recently, quantum automata and quantum logic with values in orthomodular lattices were investigated in [32,41,42,49,50]. On a different strand, the concept of multi-valued logics and automata over distributive De Morgan algebras led to the development of new practical tools for multi-valued model checking, cf. [6,12,27,31]. The importance of non-distributive De Morgan algebras for multi-valued model checking was stressed in [34].

The topic of this paper are general lattice-valued versions of the fundamental results of Kleene [28] and Büchi [7,8] characterizing the behaviors of classical finite automata by rational languages and by logic, respectively. We make no distributivity assumptions about the lattices. In contrast, weighted automata as introduced by Schützenberger [47] are based on semirings where by definition multiplication distributes over addition; this led to a theory of semiring-based weighted automata, cf. [22,46,30,1,29,17]. A semiring-based weighted logic with the same expressive power as weighted automata was presented in [14,15].

Here, we will consider automata and logics with weights taken in arbitrary bounded lattices. A lattice is bounded, if it contains a smallest and a greatest element, which model the classical truth values 0 and 1. We note that this is a much more general class of lattices than the distributive lattices, cf. [2,26] for background. Given a bounded lattice $\mathcal{L}$ and an alphabet Σ, the behavior of an $\mathcal{L}$-weighted finite automaton M is a function from Σ^* to $\mathcal{L}$, assigning to each word $w \in \Sigma^*$ its value which is computed in $\mathcal{L}$, just as for semiring-weighted automata, by calculating the values of successful runs. We also obtain the class of $\mathcal{L}$-rational series in analogy to rational languages and rational power series. We call a function $s : \Sigma^* \to \mathcal{L}$ a recognizable step function if s has finite image and for each $a \in \mathcal{L}$ the language $s^{-1}(a)$ is recognizable. We define the class $\mathrm{MSO}(\mathcal{L}, \Sigma)$ of monadic second-order formulas over $\mathcal{L}$ and Σ simply by enriching classical MSO-logic by arbitrary elements from $\mathcal{L}$ as constants, and by adding conjunction and universal quantification (both first- and second-order) to our syntax. The existential fragment of this logic is denoted by $\mathrm{EMSO}(\mathcal{L}, \Sigma)$. As usual, the semantics of such $\mathrm{MSO}(\mathcal{L}, \Sigma)$-formulas is defined inductively, using the supremum operation of the lattice for disjunction and existential quantifications, and the infimum operation for conjunction and universal quantifications. In order to define the semantics of negations of formulas, we assume that the lattice is equipped with a unary complement function interchanging 0 and 1. We note that any bounded lattice can be equipped with such a function by choosing the complement of elements of $\mathcal{L}$ not equal to 0 or 1 arbitrarily; hence, this is no essential restriction on the bounded lattice $\mathcal{L}$. Muller and Büchi recognizability and $\mathrm{EMSO}(\mathcal{L}, \Sigma)$-definability of infinitary power series $s : \Sigma^\omega \to \mathcal{L}$ are defined as, e.g., in [45]; ω-rational infinitary power series can be defined as in [23]. Our first aim is the following result.

Theorem A. Let $\mathcal{L}$ be any bounded lattice, Σ any alphabet, and $s : \Sigma^* \to \mathcal{L}$ (resp., $s : \Sigma^\omega \to \mathcal{L}$) a series. Then the following are equivalent:

(1) s is the behavior of some $\mathcal{L}$-weighted finite automaton (resp., Muller automaton).
(2) s is $\mathcal{L}$-rational (resp., ω-rational).
(3) s is a recognizable step function (resp., Muller recognizable step function).

If $\mathcal{L}$ has a complement function, then (1) is also equivalent to the following:

(4) s is definable by some sentence from $\mathrm{MSO}(\mathcal{L}, \Sigma)$.
(5) s is definable by some sentence from $\mathrm{EMSO}(\mathcal{L}, \Sigma)$.

Here, for $\mathcal{L} = \{0, 1\}$, we obtain the fundamental results of Kleene and Büchi. Note that corresponding results in the standard theory of semiring-weighted automata (cf. [1,22,30,46,14,15]) heavily use the distributivity in semirings of multiplication over addition. Our result shows that for lattices, the distributivity assumption is, somewhat surprisingly, not needed. Also, the class of $\mathrm{MSO}(\mathcal{L}, \Sigma)$-definable series does not depend on the complement function of $\mathcal{L}$.

Due to lack of space we only sketch the proof for the finitary case. However, we prove the equivalences of Theorem A even for more general structures than bounded lattices: they hold for any bi-locally finite strong bimonoid A. A strong bimonoid $(A, +, \cdot, 0, 1)$ can be viewed as "semiring without distributivity", and it is bi-locally finite if each finitely generated submonoid of the monoids $(A, +, 0)$ or $(A, \cdot, 1)$ is finite. For instance, the unit interval $[0, 1] \subset \mathbb{R}$ with Łukasiewicz t-norm and t-conorm forms a non-distributive bi-locally finite strong bimonoid. Such non-standard operations for the semantics of the logical connectives often occur in multi-valued logics, e.g., in Gödel-logics, Łukasiewicz-logics, Post-logics, cf. [25]. We show that our bimonoid results also apply to an interesting new class of weighted automata recently investigated in a series of papers [9,10,11] modeling, e.g., peak power consumption of energy.

In Theorem A and its general version for arbitrary bi-locally finite strong bimonoids, the equivalence between (1) and (3) for finite words is due to [19]. The implications (3) $\Rightarrow$ (2) and (1) $\Rightarrow$ (5) are easy to see. The main contribution of Theorem A is the proof of the implications (2) $\Rightarrow$ (1) and (4) $\Rightarrow$ (1) (cf. Theorems 2 and 3, resp.). For this we use the equivalence (1) $\Leftrightarrow$ (3) and we follow the lines of [14,15,22], but with explicit automata constructions since the theory of semiring-based weighted automata cannot be used. In particular, the general model of weighted automata is employed in the proofs concerning the Cauchy-product and the Kleene-iteration of series and the universal quantification of formulas. We show that the implication (2)$\Rightarrow$(1) fails in general without the assumption of bi-local finiteness, even for commutative right-distributive strong bimonoids.

We also characterize the recognizability of series for arbitrary strong bimonoids. As shown in [14], for this we have to restrict our MSO-logic. We define a class srMSO(A, Σ) of syntactically restricted MSO(A, Σ)-formulas and show:

Theorem B. *Let A be any strong bimonoid with complement function, and let $s : \Sigma^* \to A$ be a series. Then the following are equivalent:*

(1) *s is recognizable.*
(2) *s is definable by some sentence from srMSO(A, Σ).*

Similar equivalence results for weighted automata and suitably restricted weighted MSO-logics have been obtained recently, with semirings as weight structures, for words [14,15], trees [20,21], infinite words [18], infinite trees [44], pictures [24], traces [37], distributed processes [5], texts [35], nested words [36], and timed words [43]. Semirings may be viewed as quantitative weight structures which allow us to count. In contrast, lattices as employed here may be viewed as a logical counterpart. For lattices, we do not need to restrict the weighted logic as in the above papers but we can employ the full logic. For bounded distributive lattices this equivalence was obtained in [16]. All our proofs are constructive, yielding therefore decision procedures for MSO$(\mathcal{L}, \Sigma)$-sentences. We remark that our results also hold, correspondingly, for automata and $\mathcal{L}$-valued MSO-logic formulas over ranked trees, instead of words. Whether they hold for the other classes of structures mentioned above remains open at present.

We have also obtained an equivalence between aperiodic, star-free, weighted first-order, and weighted LTL-definable series, extending the classical equivalence results for languages [48,13] to arbitrary bi-idempotent commutative strong bimonoids, but also this is not included here because of lack of space.

1 Lattices and Bimonoids

Let $(\mathcal{L}, \leq)$ be a partially ordered set. If two elements $a, b \in \mathcal{L}$ have a least upper bound in $\mathcal{L}$, this element is called the *supremum of a and b*, denoted by $a \vee b$. Dually, the *infimum of a and b* is defined to be the greatest lower bound of a and b (provided it exists) and is denoted by $a \wedge b$. If any two elements of $\mathcal{L}$ have both a supremum and an infimum, then $(\mathcal{L}, \leq)$ is called a *lattice*. This lattice is often denoted as $(\mathcal{L}, \vee, \wedge)$ (compare [2,26]).

A lattice $\mathcal{L}$ is called *bounded* if it contains a smallest and a greatest element, denoted by 0 and 1, respectively. The lattice $\mathcal{L}$ is *distributive* if $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ for all $a, b, c \in \mathcal{L}$. This is equivalent to $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ for all $a, b, c \in \mathcal{L}$.

As is well known from lattice theory (cf. [2,26]), there is an abundance of lattices which are not distributive. In fact, a lattice is non-distributive iff it contains one of the two lattices M_3 and N_5 (cf. Figure 1). Note that B_2 is the four-element Boolean algebra $\{0, 1\} \times \{0, 1\}$ which has two incomparable complementary truth values a and b. In comparison, in M_3 we have three pairwise incomparable truth values any two of which have supremum 1 and infimum 0. Moreover, N_5 could be considered as a refinement of B_2 where the truth value b of B_2 was refined into two values b and c, each having the same relationship to a as the original element before.

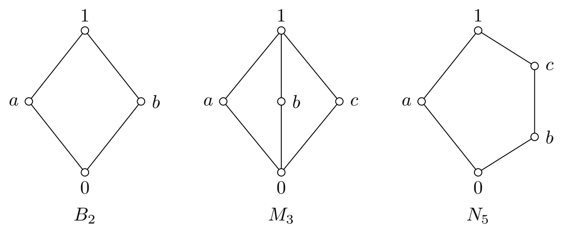


Fig. 1. Three lattices

A bounded lattice $\mathcal{L}$ together with a *complement function* $^{-} : \mathcal{L} \to \mathcal{L}$ is called an *Ockham algebra*, if it satisfies the de Morgan identities $\overline{a \vee b} = \overline{a} \wedge \overline{b}$ and $\overline{a \wedge b} = \overline{a} \vee \overline{b}$ for all $a, b \in \mathcal{L}$, and $\overline{0} = 1$ and $\overline{1} = 0$; and an Ockham algebra is called a *de Morgan algebra*, if $\overline{\overline{a}} = a$. Distributive de Morgan algebras have been intensively investigated for multi-valued model checking including the development of new practical tools (cf. [6,12,27,31]). In this context, let us consider the non-distributive lattice M_3. We define $^{-}$: $M_3 \to M_3$ by letting $\overline{0} = 1$, $\overline{1} = 0$, $\overline{a} = c$, $\overline{c} = a$, and $\overline{b} = b$. Then $(M_3, \vee, \wedge, ^{-})$ is a de Morgan algebra. We just note that N_5 becomes a (non-distributive) Ockham algebra if we put $\overline{0} = 1$, $\overline{1} = 0$, $\overline{b} = \overline{c} = a$, and either $\overline{a} = b$ or $\overline{a} = c$.

In fact, we will prove our main results for even more general structures than lattices. A *bimonoid* is a structure $(A, +, \cdot, 0, 1)$ consisting of a set A, two binary operations $+$ and $\cdot$ on A and two constants $0, 1 \in A$ such that $(A, +, 0)$ and $(A, \cdot, 1)$ are monoids. As usual, we identify the structure $(A, +, \cdot, 0, 1)$ with its carrier set A. We call A a *strong bimonoid* if the operation $+$ is commutative and 0 acts as multiplicative zero, i.e., $a \cdot 0 = 0 = 0 \cdot a$ for every $a \in A$. The bimonoid A is *commutative* if the multiplication operation is commutative. A strong bimonoid in which multiplication distributes (from both sides) over addition is called a *semiring*. For a range of examples of strong bimonoids which are not semirings we refer the reader to [19].

For our logics, in order to deal with negation of formulas, we will need strong bimonoids with *complement function* $(A, +, \cdot, ^{-}, 0, 1)$. They consist of a strong bimonoid $(A, +, \cdot, 0, 1)$ and a function$^{-}\colon A \to A$ such that $\overline{0} = 1$ and $\overline{1} = 0$. Note that trivially any strong bimonoid A can be equipped with such a complement function $^{-}$ by choosing $\overline{a}$ arbitrarily, for any $a \in A \setminus \{0, 1\}$.

We call the strong bimonoid A *additively locally finite* (*multiplicatively locally finite*, respectively) if for every finite $B \subseteq A$, the smallest submonoid of $(A, +, 0)$ (of $(A, \cdot, 1)$, respectively) containing B is finite. Moreover, A is called *bi-locally finite* if it is additively locally finite and multiplicatively locally finite. We call A *additively idempotent*, if $a + a = a$ for each $a \in A$, and *bi-idempotent*, if $a + a = a = a \cdot a$ for each $a \in A$. Clearly, every bi-idempotent, commutative strong bimonoid is bi-locally finite.

Example 1

1. Every bounded lattice $(\mathcal{L}, \vee, \wedge, 0, 1)$ is a bi-idempotent, commutative strong bimonoid.

2. Let $0 < \varepsilon < 1$. Let $A = ([\varepsilon, 1] \cup \{0\}, +, \cdot, 0, 1)$ with the interval $[\varepsilon, 1] \subseteq \mathbb{R}$ and the usual addition and multiplication on $\mathbb{R}$, which however are truncated to 1 and 0 if, respectively, larger values than 1 or smaller values than ε occur. Then A is bi-locally finite and commutative but obviously not bi-idempotent and not a semiring.

3. Let $(A, +)$ be a commutative semigroup and $(A, \cdot)$ be a semigroup. Combining these two structures we obtain a strong bimonoid structure on A by adding constants 0 and 1. Formally, let $0, 1 \notin A$ and put $A' = A \cup \{0, 1\}$. Then we define binary operations $\oplus$ and $\odot$ on A' by letting $\oplus\,|_{A \times A} = +$, $\odot\,|_{A \times A} = \cdot$, and $0 \oplus x = x \oplus 0 = x$, $1 \oplus x = x \oplus 1 = x$ if $x \neq 0$, $0 \odot x = x \odot 0 = 0$, and $1 \odot x = x \odot 1 = x$ for all $x \in A'$. Then $(A', \oplus, \odot, 0, 1)$ is a strong bimonoid. For example, if $(A, \cdot) = (A, +) = (A, \vee)$ is a join-semilattice, then the resulting bimonoid A' is a bi-idempotent, commutative semiring. Note that here the two bimonoid operations of A' coincide on $A \cup \{1\}$. As another example, let $(A, +) = (A, \vee)$ be a join-semilattice and let $\cdot$ on A be given by $a \cdot b = b$ for all $a, b \in A$. Then the resulting bimonoid A' is bi-idempotent and bi-locally finite, but not commutative and not a semiring if $|A| \geq 2$. We will come back to these examples below.

2 Weighted Finite Automata over Finite Words

In all of this paper, let Σ be an alphabet and $(A, +, \cdot, 0, 1)$ be a strong bimonoid. We recall the definition of weighted automata and their basic properties. A *weighted finite automaton* over A and Σ (for short: WFA, or A-weighted automaton) is a quadruple $M = (Q, \mathrm{in}, \mathrm{wt}, \mathrm{out})$ where Q is a finite set of states, $\mathrm{wt} : Q \times \Sigma \times Q \to A$ is the transition weight function, and $\mathrm{in}, \mathrm{out} : Q \to A$ are weight functions for entering and leaving a state, respectively. The value $\mathrm{wt}(p, \sigma, q) \in A$ indicates the weight of the transition $p \xrightarrow{\sigma} q$.

If $w = \sigma_1 \dots \sigma_n \in \Sigma^*$ where $n \geq 0$ and $\sigma_i \in \Sigma$, a *path P over w* is a sequence $P : q_0 \xrightarrow{\sigma_1} q_1 \dots \xrightarrow{\sigma_n} q_n$ where $q_0, q_1, \dots, q_n \in Q$. The *weight of P* is the product $\mathrm{wt}(P) = \mathrm{in}(q_0) \cdot \mathrm{wt}(q_0, \sigma_1, q_1) \cdot \ldots \cdot \mathrm{wt}(q_{n-1}, \sigma_n, q_n) \cdot \mathrm{out}(q_n)$. The *behavior* of M is the series $\|M\| : \Sigma^* \to A$ such that for every $w \in \Sigma^*$, $(\|M\|, w) = \sum_{\substack{P \text{ path} \\ \text{over } w}} \mathrm{wt}(P)$. Note that, if A is a semiring, then this definition of the behavior of M is precisely the one used for semiring-weighted automata, compare [22].

We also note that a large (powerful and elegant) part of the theory of semiring-based weighted automata employs matrices of transition weights and the fact that they form a monoid under usual matrix multiplication. As is easy to see, the multiplication of $n \times n$-matrices ($n \geq 2$) over a strong bimonoid A is associative iff A is a semiring. Therefore these semiring-based methods of weighted automata (as well as other particular proofs) are not applicable in our general bimonoid setting, and we convert to direct automata-theoretic arguments.

We let $A\langle\langle \Sigma^* \rangle\rangle$ comprise all formal power series, for short: series, from Σ^* to A. A series $s \in A\langle\langle \Sigma^* \rangle\rangle$ is *recognizable* if there is a WFA M such that $s = \|M\|$. If $s \in A\langle\langle \Sigma^* \rangle\rangle$ is a series and $w \in \Sigma^*$, as usual we write (s, w) for $s(w)$.

Now let $s, s' \in A\langle\langle \Sigma^* \rangle\rangle$ be series and $a \in A$. We define the scalar product $a \cdot s$ and the sum $s + s'$ by letting $(a \cdot s, w) = a \cdot (s, w)$ and $(s + s', w) = (s, w) + (s', w)$, for each $w \in \Sigma^*$. For $L \subseteq \Sigma^*$, we define the *characteristic series* $\mathbb{1}_L \in A\langle\langle \Sigma^* \rangle\rangle$ by $(\mathbb{1}_L, w) = 1$ if $w \in L$, and $(\mathbb{1}_L, w) = 0$ otherwise.

We recall that $s \in A\langle\langle \Sigma^* \rangle\rangle$ is a *recognizable step function* if there are $n \geq 0$, recognizable languages $L_1, \dots, L_n \subseteq \Sigma^*$, and $a_1, \dots, a_n \in A$ such that $s = \sum_{i=1}^{n} a_i \cdot \mathbb{1}_{L_i}$. Since the class of recognizable languages is closed under Boolean operations, we can assume that the family $(L_i \mid 1 \leq i \leq n)$ forms a partitioning of Σ^*. By Lemma 8 of [19] every recognizable step function is recognizable. The following result will be crucial for us.

Theorem 1 ([19], Theorem 11). *Let A be any bi-locally finite strong bimonoid and $s \in A\langle\langle \Sigma^* \rangle\rangle$. If s is recognizable, then s is a recognizable step function.*

Now we point out a relationship between weighted automata over strong bimonoids and two weighted automata models investigated recently in a series of papers [9,10,11]. Chatterjee, Doyen, and Henzinger consider weighted automata where $A = \mathbb{Q}$, the set of rational numbers, together with a value function $\mathsf{Val} : \mathbb{Q}^+ \to \mathbb{R}$. The weight of a path is given by the value of Val applied to the sequence of weights of the transitions composing the path. The value $(\|M\|, w)$ of M for a word $w \in \Sigma^*$ is defined as the supremum of the weights of all successful paths over w. In particular, they consider the value functions $\mathsf{Val} = \mathsf{Max}$ and $\mathsf{Val} = \mathsf{Last}$ where $\mathsf{Max}(v) = \max\{v_i \mid 1 \leq i \leq n\}$ and $\mathsf{Last}(v) = v_n$ for every finite sequence $v = v_1 \dots v_n \in \mathbb{Q}^+$. For instance, peak power consumption can be modelled as the maximum of a sequence of weights representing power usage [9,11].

These two automata models can be viewed as weighted automata over strong bimonoids in the following way. In both cases, we let the addition operation on $A = \mathbb{Q}$ be the usual supremum operation for pairs of rational numbers. For the value function Max, we let the multiplication operation on A also be the supremum operation, and for the value function Last, we use the multiplication given by $a \cdot b = b$ for $a, b \in \mathbb{Q}$. Now let A' be the bimonoid constructed in Example 1(3).

Observe that A' is bi-locally finite and bi-idempotent and is a semiring in the Max-case. Then the behavior of Max-, resp., Last-automata coincides with the behavior of the weighted automaton M over the corresponding bimonoid A'. This shows that all our results are applicable to Max- and Last-automata. For instance, Theorems 2 and 3 show that the behaviors of Max- and Last-systems can be specified equivalently also by rational expressions or by weighted MSO-formulas.

3 Recognizability, Rationality, and MSO-Definability

In this section we prove the equivalence between recognizable, rational, and (E)MSO-definable series over strong bimonoids.

Let $s, s' \in A\langle\langle \Sigma^* \rangle\rangle$ be series, $a \in A$, and $m \geq 0$. We define the Cauchy product $s \cdot s'$ and the m-th power s^m $(m \geq 1)$ by letting, for each $w \in \Sigma^*$, $(s \cdot s', w) = \sum_{w=uv} (s, u) \cdot (s', v)$ and $(s^m, w) = \sum_{w=u_1 \ldots u_m} (s, u_1) \cdot \ldots \cdot (s, u_m)$. Also, let $s^0 = \mathbb{1}_{\{\varepsilon\}}$. We say that s is *proper* if $(s, \varepsilon) = 0$. In this case we define the *star* $s^* \in A\langle\langle \Sigma^* \rangle\rangle$ by letting $(s^*, w) = \sum_{m=0}^{|w|} (s^m, w)$. Given $a \in A$ and $w \in \Sigma^*$, the series $a \cdot \mathbb{1}_{\{w\}}$ is called a *monomial*. A series $s \in A\langle\langle \Sigma^* \rangle\rangle$ is called *rational* if it can be constructed from finitely many monomials using the operations sum, Cauchy product, and star where the latter is applied only to proper series. First we will show the following result:

Theorem 2. *Let A be any bi-locally finite strong bimonoid and $s \in A\langle\langle \Sigma^* \rangle\rangle$. Then s is recognizable iff s is rational.*

Proof. (sketch) Let s be recognizable. By Theorem 1, s is a recognizable step function. Since $a \cdot \mathbb{1}_L = (a \cdot \mathbb{1}_{\{\varepsilon\}}) \cdot \mathbb{1}_L$, it suffices to prove that $\mathbb{1}_L$ is rational for every recognizable language L. We construct L from finitely many singletons using the operations unambiguous union, unambiguous product, and unambiguous star. Then we can obtain $\mathbb{1}_L$ in the same way as L by the corresponding rational operations.

Conversely, we show that every rational series is a recognizable step function. Trivially, the class of recognizable step functions contains all monomials and is closed under sum. For two recognizable steps functions $s, s' \in A\langle\langle \Sigma^* \rangle\rangle$ we construct (according to Lemma 8 of [19]) deterministic automata M, M' with $s = \|M\|$ and $s' = \|M'\|$; moreover, M and M' have the property that the weight of each of their transitions is either 1 or 0. Then, a usual sequential product automaton of M and M' recognizes $s \cdot s'$. Next let s be proper. We can choose M with $s = \|M\|$ such that M has exactly one initial state and this state is not reachable by a transition with non-zero weight; then a looping of M with itself recognizes s^*. So, $s \cdot s'$ and s^* are recognizable, and hence again recognizable step functions by Theorem 1. ◻

Next we show that for arbitrary commutative strong bimonoids, rationality of series does not imply recognizability.

Example 2. Let A be the free commutative bimonoid freely generated by $\{a, b\}$. Note that the elements of A can be obtained from the generators by alternating two constructions, which are: (1) taking products of powers of different elements, and (2) forming sums of multiples of different elements. Now consider $\Sigma = \{\sigma\}$ and the series $s = (a \cdot \mathbb{1}_{\{\sigma\}})^* + (b \cdot \mathbb{1}_{\{\sigma\}})^* \in A\langle\langle \Sigma^* \rangle\rangle$. Then s is rational (and recognizable by a WFA with two states). We have $(s, \sigma^n) = a^n + b^n$ for each $n \in \mathbb{N}$, and $(s \cdot s, \sigma^k) = \sum_{\substack{m,n \geq 0 \\ m+n=k}} (a^m + b^m) \cdot (a^n + b^n)$ for each $k \in \mathbb{N}$. We claim that $s \cdot s$ is not recognizable.

Let $M = (Q, \mathrm{in}, \mathrm{wt}, \mathrm{out})$ be any WFA. Choose $m, n \geq 1$ such that $a^n + b^n \notin I$ and $(a^m + b^m)(a^n + b^n) \notin I$, where $I = \mathrm{im}(\mathrm{in}) \cup \mathrm{im}(\mathrm{wt}) \cup \mathrm{im}(\mathrm{out})$ and $\mathrm{im}(f)$ denotes the image of a function f. The values of $\|M\|$ arise by performing construction (1) on the elements of I, followed by construction (2). We cannot obtain $(a^m + b^m)(a^n + b^n)$ by construction (1) on the elements of I. Hence, if $k = m + n$, we have $(s \cdot s, \sigma^k) \notin \mathrm{im}(\|M\|)$, showing $s \cdot s \neq \|M\|$, and our claim follows. A similar argument works if we replace A by the free commutative right-distributive bimonoid generated freely by $\{a, b\}$. In this case, we have $(s \cdot s, \sigma^k) = \sum_{\substack{m,n \geq 0 \\ m+n=k}} (a^m \cdot (a^n + b^n) + b^m \cdot (a^n + b^n))$ for $k \in \mathbb{N}$.

Next let $(A, +, \cdot, ^-, 0, 1)$ be a strong bimonoid with complement function.

Definition 1. The syntax of formulas of the *weighted MSO-logic over A and Σ* is given by the grammar

$$\varphi ::= a \mid R_\sigma(x) \mid x \leq y \mid x \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \exists X.\varphi \mid \forall x.\varphi \mid \forall X.\varphi$$

where $a \in A$, $\sigma \in \Sigma$, x, y are first-order variables, and X is a second-order variable. We let $\mathrm{Free}(\varphi)$ be the set of all free variables of φ. We denote by $\mathrm{MSO}(A, \Sigma)$ the collection of all such weighted MSO-formulas.

In the sequel, for $w \in \Sigma^*$ and $n = |w|$, the length of w, we also write $w = w(1) \ldots w(n)$ with $w(i) \in \Sigma$, and $\mathrm{dom}(w) = \{1, \ldots, n\}$. Now let $\mathcal{V}$ be a finite set of first-order and second-order variables and $w \in \Sigma^*$. A $(\mathcal{V}, w)$-*assignment* ρ maps first-order variables in $\mathcal{V}$ to elements of $\mathrm{dom}(w)$ and second-order variables in $\mathcal{V}$ to subsets of $\mathrm{dom}(w)$. Then we encode such a pair (w, ρ) as a word u with $|u| = |w|$ over the extended alphabet $\Sigma_\mathcal{V} = \Sigma \times \{0,1\}^\mathcal{V}$ as usual, and we call a word $u \in \Sigma_\mathcal{V}^*$ *valid* if u corresponds to such a pair. Clearly, the language $N_\mathcal{V} = \{u \in \Sigma_\mathcal{V}^* \mid u \text{ is valid}\}$ is recognizable.

Let $(w, \rho) \in N_\mathcal{V}$, x be a first-order variable, and $i \in \mathrm{dom}(w)$. Then $\rho[x \to i]$ is the $(\mathcal{V} \cup \{x\}, w)$-assignment which maps x to i and acts as ρ on each of the other variables. If u corresponds to (w, ρ), we also write $u[x \to i]$ for $(w, \rho[x \to i])$. Similarly, $\rho[X \to I]$ and $u[X \to I]$ are defined for $I \subseteq \mathrm{dom}(w)$.

Definition 2 (compare [14]). Let $\varphi \in \mathrm{MSO}(A, \Sigma)$ and $\mathcal{V}$ be a finite set of variables containing $\mathrm{Free}(\varphi)$. The $\mathcal{V}$-*semantics* of φ is a formal power series $[\![\varphi]\!]_\mathcal{V} \in A\langle\langle \Sigma_\mathcal{V}^* \rangle\rangle$. Let $u = (w, \rho) \in \Sigma_\mathcal{V}^*$. If $u \notin N_\mathcal{V}$, then we put $([\![\varphi]\!]_\mathcal{V}, u) = 0$. Otherwise, we define $([\![\varphi]\!]_\mathcal{V}, u) \in A$ inductively as follows:

$$([\![a]\!]_\mathcal{V}, u) = a \qquad ([\![R_\sigma(x)]\!]_\mathcal{V}, u) = \begin{cases} 1 & \text{if } w(\rho(x)) = \sigma \\ 0 & \text{otherwise} \end{cases}$$

$$([\![x \leq y]\!]_\mathcal{V}, u) = \begin{cases} 1 & \text{if } \rho(x) \leq \rho(y) \\ 0 & \text{otherwise} \end{cases} \qquad ([\![x \in X]\!]_\mathcal{V}, u) = \begin{cases} 1 & \text{if } \rho(x) \in \rho(X) \\ 0 & \text{otherwise} \end{cases}$$

$$(\llbracket \neg\varphi \rrbracket_{\mathcal{V}}, u) = \overline{(\llbracket \varphi \rrbracket_{\mathcal{V}}, u)}$$

$$(\llbracket \varphi \vee \psi \rrbracket_{\mathcal{V}}, u) = (\llbracket \varphi \rrbracket_{\mathcal{V}}, u) + (\llbracket \psi \rrbracket_{\mathcal{V}}, u) \qquad (\llbracket \varphi \wedge \psi \rrbracket_{\mathcal{V}}, u) = (\llbracket \varphi \rrbracket_{\mathcal{V}}, u) \cdot (\llbracket \psi \rrbracket_{\mathcal{V}}, u)$$

$$(\llbracket \exists x.\varphi \rrbracket_{\mathcal{V}}, u) = \sum_{i \in \mathrm{dom}(u)} (\llbracket \varphi \rrbracket_{\mathcal{V} \cup \{x\}}, u[x \to i]) \qquad (\llbracket \forall x.\varphi \rrbracket_{\mathcal{V}}, u) = \prod_{i \in \mathrm{dom}(u)} (\llbracket \varphi \rrbracket_{\mathcal{V} \cup \{x\}}, u[x \to i])$$

$$(\llbracket \exists X.\varphi \rrbracket_{\mathcal{V}}, u) = \sum_{I \subseteq \mathrm{dom}(u)} (\llbracket \varphi \rrbracket_{\mathcal{V} \cup \{X\}}, u[X \to I])$$

$$(\llbracket \forall X.\varphi \rrbracket_{\mathcal{V}}, u) = \prod_{I \subseteq \mathrm{dom}(u)} (\llbracket \varphi \rrbracket_{\mathcal{V} \cup \{X\}}, u[X \to I])$$

where in the product over $\mathrm{dom}(u)$ we follow the natural order, and we fix some order on the power set of $\{1, \dots, |u|\}$ so that the last product is defined. We let $\llbracket \varphi \rrbracket = \llbracket \varphi \rrbracket_{\mathrm{Free}(\varphi)}$.

We let $\mathrm{EMSO}(A, \Sigma)$ contain all *existential* $\mathrm{MSO}(A, \Sigma)$*-formulas* φ, i.e., φ is of the form $\exists X_1 \dots \exists X_n.\psi$ with $\psi \in \mathrm{MSO}(A, \Sigma)$ containing no set quantifications. Let $Z \subseteq \mathrm{MSO}(A, \Sigma)$ and $s \in A\langle\langle \Sigma^* \rangle\rangle$. We say that s is Z*-definable* if there is a sentence (i.e., a formula without free variables) $\varphi \in Z$ such that $s = \llbracket \varphi \rrbracket$.

Theorem 3. *Let A be any bi-locally finite strong bimonoid with complement function, and let $s \in A\langle\langle \Sigma^* \rangle\rangle$. Then the following are equivalent:*

(1) s is recognizable.
(2) s is $\mathrm{EMSO}(A, \Sigma)$*-definable.*

If A is commutative, then conditions (1) and (2) are also equivalent to:

(3) s is $\mathrm{MSO}(A, \Sigma)$*-definable.*

Following [4], we call a formula $\varphi \in \mathrm{MSO}(A, \Sigma)$ *Boolean* if it does not contain constants a (with $a \in A \setminus \{0, 1\}$) and does not use disjunction or existential first or second order quantification. Clearly, the Boolean formulas capture the full power of classical (unweighted) MSO logic. In particular, for every recognizable language $L \subseteq \Sigma^*$ there is a Boolean sentence φ such that $\mathbb{1}_L = \llbracket \varphi \rrbracket$.

A formula $\varphi \in \mathrm{MSO}(A, \Sigma)$ is *almost unambiguous* if it is constructed from constants a ($a \in A$) and Boolean formulas, using disjunction, conjunction, and negation. One can show that the almost unambiguous formulas define precisely the recognizable step functions.

Definition 3 (compare [15]). A formula $\varphi \in \mathrm{MSO}(A, \Sigma)$ is called *syntactically restricted*, if it satisfies the following conditions:

1. Whenever φ contains a conjunction $\psi \wedge \psi'$ as subformula but not in the scope of a universal first order quantifier, then ψ or ψ' is Boolean.
2. Whenever φ contains $\forall X.\psi$ as a subformula, then ψ is a Boolean.
3. Whenever φ contains $\forall x.\psi$ or $\neg\psi$ as a subformula, then ψ is almost unambiguous.

We let $\mathrm{srMSO}(A, \Sigma)$ denote the set of all syntactically restricted formulas of $\mathrm{MSO}(A, \Sigma)$.

We note that condition (1) of Definition 3 is slightly more restrictive than condition (1) of Definition 4.6 of [15]. This provides all technical definitions for our second main result of this section, see Theorem B in the introduction.

For our proofs we will need preservation properties of recognizability under morphisms. Let Σ, Γ be alphabets and $h : \Sigma^* \to \Gamma^*$ be a length-preserving morphism. For every series $s \in A\langle\langle \Sigma^* \rangle\rangle$ we define $h(s) \in A\langle\langle \Gamma^* \rangle\rangle$ by $(h(s), v) = \sum_{w \in h^{-1}(v)} (s, w)$ for every $v \in \Gamma^*$. In the theory of semiring-weighted automata there are very short algebraic proofs that length-preserving morphisms preserve recognizability (cf., e.g., Theorem II.4.3 of [46]) which, however, ultimately rest on the distributivity of the semiring. In our setting we cannot use these methods; therefore we give a direct automata-theoretic proof.

Lemma 1. Let Σ, Γ be alphabets and $h : \Sigma^* \to \Gamma^*$ be a length-preserving monoid morphism. Then $h : A\langle\langle \Sigma^* \rangle\rangle \to A\langle\langle \Gamma^* \rangle\rangle$ preserves recognizability.

Proof. Let $M = (Q, \mathrm{in}, \mathrm{wt}, \mathrm{out})$ be a WFA over A and Σ. Choose $\sigma_0 \in \Sigma$. We construct the WFA $M' = (Q', \mathrm{in}', \mathrm{wt}', \mathrm{out}')$ over A and Γ by letting $Q' = Q \times \Sigma$; $\mathrm{in}'(q, \sigma) = \mathrm{in}(q)$ for every $(q, \sigma) \in Q'$; for every $\gamma \in \Gamma$ and $(p, \sigma), (q, \sigma') \in Q'$ we put $\mathrm{wt}'((p, \sigma), \gamma, (q, \sigma')) = \mathrm{wt}(p, \sigma, q)$ if $h(\sigma) = \gamma$, and $\mathrm{wt}'((p, \sigma), \gamma, (q, \sigma')) = 0$ otherwise, and for every $(q, \sigma) \in Q'$ we let $\mathrm{out}'(q, \sigma) = \mathrm{out}(q)$ if $\sigma = \sigma_0$, and 0 otherwise. One can show that $h(\|M\|) = \|M'\|$. $\square$

Using Lemma 1 we can show that for $\varphi \in \mathrm{MSO}(A, \Sigma)$, the semantics $[\![\varphi]\!]_{\mathcal{V}}$ ($\mathcal{V}$ a finite set of variables containing $\mathrm{Free}(\varphi)$) are *consistent* , i.e., $([\![\varphi]\!]_{\mathcal{V}}, (w, \rho)) = ([\![\varphi]\!], (w, \rho_{|\,\mathrm{Free}(\varphi)}))$ for each $(w, \rho) \in N_{\mathcal{V}}$ (cf. [15], Prop. 3.3). In particular, $[\![\varphi]\!]$ is recognizable iff $[\![\varphi]\!]_{\mathcal{V}}$ is recognizable, and $[\![\varphi]\!]$ is a recognizable step function iff $[\![\varphi]\!]_{\mathcal{V}}$ is a recognizable step function.

Proof of Theorem 3 and Theorem B, (1) $\Rightarrow$ (2): Let $s = \|M\|$ for some WFA M. The proof of Theorem 5.7 of [15] explicitly describes an $\mathrm{EMSO}(A, \Sigma)$-sentence ξ such that $\|M\| = [\![\xi]\!]$. For Theorem B, we construct M with initial and final weights in $\{0, 1\}$; then ξ is syntactically restricted.

(2) $\Rightarrow$ (1)*,* (3) $\Rightarrow$ (1) *if A is commutative*: (sketch). By induction over the structure of a formula φ we show that $[\![\varphi]\!]$ is recognizable. In case of Theorem B, for atomic formulas and for disjunction and conjunction of formulas, we use automata constructions; for the case of Theorem 3 and for negation, this is easy using Theorem 1 and recognizable step functions. For existential quantifications we apply Lemma 1. For universal first-order respectively second-order quantifications we apply the arguments of Lemmas 5.4 of [15] resp. Proposition 6.3 of [14]. $\square$

References

1. Berstel, J., Reutenauer, C.: Rational Series and Their Languages. EATCS Monograph on Theoretical Computer Science, vol. 12. Springer, Heidelberg (1988)
2. Birkhoff, G.: Lattice Theory. Revised ed. Amer. Math. Soc. Colloq. Publ., vol. XXV. Amer. Math. Soc., Providence (1961)

3. Birkhoff, G., von Neumann, J.: The logic of quantum mechanics. Annals of Math. 37, 823–843 (1936)
4. Bollig, B., Gastin, P.: Weighted versus probabilistic logics. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 18–38. Springer, Heidelberg (2009)
5. Bollig, B., Meinecke, I.: Weighted distributed systems and their logics. In: Artemov, S., Nerode, A. (eds.) LFCS 2007. LNCS, vol. 4514, pp. 54–68. Springer, Heidelberg (2007)
6. Bruns, G., Godefroid, P.: Model checking with multi-valued logics. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 281–293. Springer, Heidelberg (2004)
7. Büchi, J.R.: Weak second-order arithmetic and finite automata. Zeitschr. für math. Logik und Grundl. der Mathem. 6, 66–92 (1960)
8. Büchi, J.R.: On a decision method in restricted second-order arithmetic. In: Proc. 1960 Int. Congr. for Logic, Methodology and Philosophy of Science, pp. 1–11. Stanford Univ. Press, Stanford (1962)
9. Chatterjee, K., Doyen, L., Henzinger, T.: Quantitative languages. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 385–400. Springer, Heidelberg (2008)
10. Chatterjee, K., Doyen, L., Henzinger, T.: Alternating weighted automata. In: Gębala, M. (ed.) FCT 2009. LNCS, vol. 5699, pp. 3–13. Springer, Heidelberg (2009)
11. Chatterjee, K., Doyen, L., Henzinger, T.: Expressiveness and closure properties for quantitative languages. In: Proc. of 24th LICS 2009, pp. 199–208. IEEE Computer Society Press, Los Alamitos (2009)
12. Chechik, M., Devereux, B., Gurfinkel, A.: Model-checking infinite state-space systems with fine-grained abstractions using spin. In: Dwyer, M.B. (ed.) SPIN 2001. LNCS, vol. 2057, pp. 16–36. Springer, Heidelberg (2001)
13. Diekert, V., Gastin, P.: First-order definable languages. In: Flum, J., Grädel, E., Wilke, T. (eds.) Logic and Automata: History and Perspectives, pp. 261–306. Amsterdam University Press, Amsterdam (2007)
14. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theor. Comput. Sci. 380(1-2), 69–86 (2007)
15. Droste, M., Gastin, P.: Weighted automata and weighted logics. In: [17], ch. 5
16. Droste, M., Kuich, W., Rahonis, G.: Multi-valued MSO logics over words and trees. Fundamenta Informaticae 84, 305–327 (2008)
17. Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg (2009)
18. Droste, M., Rahonis, G.: Weighted automata and weighted logics on infinite words. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 49–58. Springer, Heidelberg (2006)
19. Droste, M., Stüber, T., Vogler, H.: Weighted finite automata over strong bimonoids. Information Sciences 180, 156–166 (2010)
20. Droste, M., Vogler, H.: Weighted tree automata and weighted logics. Theoret. Comput. Sci. 366, 228–247 (2006)
21. Droste, M., Vogler, H.: Weighted logics for unranked tree automata. Theory of Comput. Systems (2010) (in print)

22. Eilenberg, S.: Automata, Languages, and Machines – Volume A. Pure and Applied Mathematics, vol. 59. Academic Press, London (1974)
23. Ésik, Z., Kuich, W.: A semiring-semimodule generalization of ω-regular languages I. J. Automata, Languages and Combinatorics 10(2/3), 203–242 (2005)
24. Fichtner, I.: Weighted picture automata and weighted logics. Theory of Comput. Syst (to appear 2010)
25. Gottwald, S.: A Treatise on Many-Valued Logics. Studies in Logic and Computation, vol. 9. Research Studies Press, Baldock (2001)
26. Grätzer, G.: General Lattice Theory. Birkhäuser, Basel (2003)
27. Gurfinkel, A., Chechik, M.: Multi-valued model checking via classical model checking. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 266–280. Springer, Heidelberg (2003)
28. Kleene, S.E.: Representation of events in nerve nets and finite automata. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, pp. 3–42. Princeton University Press, Princeton (1956)
29. Kuich, W.: Semirings and formal power series. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, ch. 9, vol. 1, pp. 609–677. Springer, Heidelberg (1997)
30. Kuich, W., Salomaa, A.: Semirings, Automata, Languages. Monogr. Theoret. Comput. Sci. EATCS Ser., vol. 5. Springer, Heidelberg (1986)
31. Kupferman, O., Lustig, Y.: Lattice automata. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 199–213. Springer, Heidelberg (2007)
32. Li, Y.M.: Finite automata based on quantum logic and monadic second-order quantum logic. Science China Information Sciences 53, 101–114 (2010)
33. Łukasiewicz, J.: O logice trojwartosciowej. Ruch Filosoficzny 5, 169–171 (1920)
34. Mallya, A.: Deductive multi-valued model checking. In: Gabbrielli, M., Gupta, G. (eds.) ICLP 2005. LNCS, vol. 3668, pp. 297–310. Springer, Heidelberg (2005)
35. Mathissen, C.: Definable transductions and weighted logics for texts. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 324–336. Springer, Heidelberg (2007)
36. Mathissen, C.: Weighted logics for nested words and algebraic formal power series. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 221–232. Springer, Heidelberg (2008)
37. Meinecke, I.: Weighted logics for traces. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 235–246. Springer, Heidelberg (2006)
38. Perrin, D.: Finite automata. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, ch. 1, vol. B, pp. 3–57. Elsevier, Amsterdam (1990)
39. Perrin, D., Pin, J.E.: Infinite Words. Elsevier, Amsterdam (2004)
40. Post, E.L.: Introduction to a general theory of elementary propositions. Am. J. Math. 43, 163–185 (1921)
41. Qiu, D.: Automata theory based on quantum logic: some characterizations. Information and Computation 190, 179–195 (2004)
42. Qiu, D.: Automata theory based on quantum logic: Reversibilities and pushdown automata. Theoret. Comput. Sci. 386, 38–56 (2007)
43. Quaas, K.: Weighted timed MSO logics. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 419–430. Springer, Heidelberg (2009)
44. Rahonis, G.: Weighted Muller tree automata and weighted logics. J. Automata, Languages and Combinatorics 12, 455–483 (2007)

45. Rahonis, G.: Fuzzy languages. In: [17], ch. 12
46. Salomaa, A., Soittola, M.: Automata-Theoretic Aspects of Formal Power Series. Texts and Monographs in Computer Science. Springer, Heidelberg (1978)
47. Schützenberger, M.P.: On the definition of a family of automata. Inf. and Control 4, 245–270 (1961)
48. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, ch. 4, vol. B, pp. 133–191. Elsevier Publishing Company, Amsterdam (1990)
49. Ying, M.: Automata theory based on quantum logic (I) and (II). Int. J. of Theoret. Physics 39, 985–995, 2545–2557 (2000)
50. Ying, M.: A theory of computation based on quantum logic (I). Theoret. Comput. Sci. 344, 134–207 (2005)

On Müller Context-Free Grammars*

Zoltán Ésik and Szabolcs Iván

Department of Computer Science, University of Szeged, Hungary

Abstract. We define context-free grammars with Müller acceptance condition that generate languages of countable words. We establish several elementary properties of the class of Müller context-free languages including closure properties and others. We show that every Müller context-free grammar can be transformed into a normal form grammar in polynomial space without increasing the size of the grammar, and then we show that many decision problems can be solved in polynomial time for Müller context-free grammars in normal form. These problems include deciding whether the language generated by a normal form grammar contains only well-ordered, scattered, or dense words. In a further result we establish a limitedness property of Müller context-free grammars: If the language generated by a grammar contains only scattered words, then either there is an integer n such that each word of the language has Hausdorff rank at most n, or the language contains scattered words of arbitrarily large Hausdorff rank. We also show that it is decidable which of the two cases applies.

1 Introduction

In a general setting, a word over an alphabet Σ is an isomorphism class of a linear order labeled in Σ. In this paper, we consider languages of countable words including scattered and dense words, i.e., words whose underlying linear order is scattered or dense, cf. [16].

Whereas finite automata over ω-words and more generally countable and even uncountable words have been studied since the 1960's, cf. e.g., [7,8,1,18,19,2,5], context-free grammars generating infinite words received little attention.

Context-free grammars have been used to generate languages of ω-words in [9] and [4,15]. Context-free grammars generating languages of countable words equipped with Büchi acceptance condition were considered in [11]. At the end of [11], we have also defined context-free grammars with Müller acceptance condition and showed that they generate a strictly larger class of languages. In this paper, our aim is to study Müller context-free languages in a systematic way.

We establish several elementary properties of the class of Müller context-free languages including closure properties and others. We show that every Müller context-free grammar can be transformed into a normal form grammar in polynomial space with only a linear increase in the size of the grammar, and then we

* This research was supported by OTKA grant no. K75249 and the TÁMOP-4.2.2/08/1/2008-0008 program of the Hungarian National Development Agency.

show that many decision problems can be solved in polynomial time for Müller context-free grammars in normal form. These problems include deciding whether the language generated by a normal form grammar contains only well-ordered, scattered, or dense words. In a further result we establish a limitedness property of Müller context-free grammars: If the language generated by a grammar contains only scattered words, then either there is an integer n such that each word of the language has Hausdorff rank at most n, or the language contains scattered words of arbitrarily large Hausdorff rank. We also prove that it is decidable which of the two cases applies and show that if the rank of the words in the language is bounded by some integer, then the least such bound is computable. Again, we give a polynomial time algorithm for grammars in normal form.

Countable words were first investigated in [10], where they were called "arrangements". Any countable word can be represented as the frontier of an infinite tree. Accordingly, any Müller context-free language can be seen as the frontier language of a tree language recognized by a Müller tree automaton, cf. [17]. Many of our decidability results can thus alternatively be proved using certain closure properties and decidability results on Müller automata and some specific constructions, but these general arguments do not provide the simple characterizations developed in the paper and usually yield higher complexity. See also Section 4.

2 Notation

In this section we recall some concepts for linear orders and words and introduce the notion of Müller context-free grammars and languages.

2.1 Linear Orders and Words

We make use of the standard notions concerning linear orderings, see e.g. [16].

A linear order $(P, \leq)$ is a *well-order* if each nonempty subset of P has a least element, and is *dense* if it has at least two elements and for any $x < y$ in P there is some z with $x < z < y$.[1] A *quasi-dense* linear order is a linear order $(P, \leq)$ containing a dense linear sub-order, so that P has a subset P' such that $(P', \leq)$ is a dense order. A *scattered* linear order is a linear order which is not quasi-dense. The *order type* of a linear order is the isomorphism class of the linear order.

It is clear that every finite linear order is a well-order, every well-order is a scattered order, and every dense order is quasi-dense. It is well-known that up to isomorphism there are 4 countable dense linear orders: the rationals $\mathbb{Q}$ with the usual order and possibly endowed with either a least or a greatest element (or both).

An *ordinal* is the order type of a well-order. The finite ordinals n are the order types of the finite linear orders. As usual, we denote by ω the least infinite ordinal which is the order type of the finite ordinals and of the natural numbers

[1] In [16], a singleton linear order is also called dense.

$\mathbb{N} = \{1, 2, \dots\}$ equipped with the usual order. The order type of $\mathbb{Q}$ will be denoted η.

An *alphabet* is a finite nonempty set Σ. A *word* over an alphabet Σ is a labeled linear order, i.e., a triple $u = (\mathrm{dom}(u), \leq_u, \lambda_u)$, where $(\mathrm{dom}(u), \leq_u)$ is a linear order and λ_u is a labeling function $\mathrm{dom}(u) \to \Sigma$. The underlying linear order $\mathrm{dom}(\varepsilon)$ of the empty word ε is the empty linear order. We say that a word is *finite* (*infinite* or *countable*, respectively), if its underlying linear order is finite (infinite or countable, respectively). An *isomorphism* of words is an isomorphism of the underlying linear orders that preserves the labeling. *Embeddings* of words are defined in the same way. We usually identify isomorphic words. We will say that a word u is a *subword* of a word v if there is an embedding $u \hookrightarrow v$. When in addition the image of the underlying linear order of u is an interval of the underlying linear order of v we call u a *factor* of v.

The *order type of a word* is the order type of its underlying linear order. Thus, the order type of a finite word is a finite ordinal. A word whose order type is ω is called an *ω-word*. Suppose that $u = (P, \leq_u, \lambda_u)$ and $v = (Q, \leq_v, \lambda_v)$ are words over Σ. Then their *concatenation uv* is the word over Σ whose underlying linear order is the ordered sum $P + Q$ (cf. [16]) and whose labeling function agrees with λ_u on points in P, and with λ_v on points in Q. More generally, when I is a linear order and u_i is a word over Σ with underlying linear order $P_i = (\mathrm{dom}(u_i), \leq_i)$, for each $i \in I$, then the *generalized concatenation* $\prod_{i \in I} u_i$ is the word whose underlying linear order is the generalized sum $\sum_{i \in I} P_i$ (cf. [16]) and whose labeling function agrees with the labeling function of P_i on the elements of each P_i. In particular, when $u_0, u_1, \dots, u_n, \dots$ are words over Σ and I is the linear order ω or its reverse $-\omega$, then $\prod_{i \in I} u_i$ is the word $u_0 u_1 \dots u_n \dots$ or $\dots u_n \dots u_1 u_0$, respectively. When $u_i = u$ for each i, these words are denoted u^ω and $u^{-\omega}$, respectively.

Some examples of words over the alphabet $\Sigma = \{a, b\}$ are the finite word aab which is the (isomorphism class of the) 3-element labeled linear order $\{0 < 1 < 2\}$ whose points are labeled a, a and b, in this order. Examples of infinite words are a^ω and $a^{-\omega}$, whose order types are ω and $-\omega$, respectively, such that each point is labeled a. For another example, consider the linear order $\mathbb{Q}$ of the rationals and label each point a. The resulting word of order type η is denoted a^η. More generally, let Σ contain the (different) letters $a_1, \dots, a_n$. Then up to isomorphism there is a unique labeling of the rationals such that between any two points there are n points labeled $a_1, \dots, a_n$, respectively. The resulting word is denoted $(a_1, \dots, a_n)^\eta$, cf. [12].

We call a word over an alphabet Σ *well-ordered*, *scattered*, *dense*, or *quasi-dense* if its underlying linear order has the appropriate property. For example, the words a^ω, $a^\omega b^\omega a$, $(a^\omega)^\omega$ over the alphabet $\{a, b\}$ are well-ordered, the words $a^\omega a^{-\omega}$, $a^{-\omega} a^\omega$ are scattered but not well-ordered, the words a^η, $a^\eta b a^\eta$, $(a, b)^\eta$ are dense, and the words $(ab)^\eta$, $(a^\omega)^\eta$, $(a^\eta b)^\omega$ are quasi-dense but not dense.

As already mentioned, we will usually identify isomorphic words, so that a word is an isomorphism type (or isomorphism class) of a labeled linear order. When Σ is an alphabet, we let Σ^*, Σ^ω and Σ^∞ respectively denote the set of all finite words, ω-words, and *countable* words over Σ. Σ^+ is the set of all finite nonempty words, and $\Sigma^{+\infty} = \Sigma^\infty \Sigma \Sigma^\infty$ is the set of all countable nonempty words over Σ. The length of a finite word w will be denoted $|w|$.

A *language over* Σ is any subset L of Σ^∞. When $L \subseteq \Sigma^*$ or $L \subseteq \Sigma^\omega$, we sometimes call L a *language of finite words* or an *ω-language* (or a language of ω-words).

When A is a set, $P(A)$ stands for $\{X : X \subseteq A\}$, the *power set of* A.

2.2 Tree Domains and Müller Context-Free Grammars

We follow standard notions concerning trees and tree domains.

A *tree domain* is an arbitrary nonempty, prefix closed subset T of $\{1, \dots, k\}^*$ for some $k \geq 1$ (whose elements are usually referred to as *nodes of* T). The *lexicographic order* on $\{1, \dots, k\}^*$ is defined by $u \leq_{\text{lex}} v$ if and only if $u = v$, or $u = xiy$ and $v = xjz$ for some $x, y, z \in \{1, \dots, k\}^*$ and $1 \leq i < j \leq k$.

A *context-free grammar with Müller acceptance condition*, or MCFG for short, is a system $G = (V, \Sigma, P, S, \mathcal{F})$ where V is the finite, nonempty set of *nonterminals* (or variables), Σ is the *terminal* alphabet with $V \cap \Sigma = \emptyset$, $S \in V$ is the *start symbol*, P is the finite set of *productions* (or rules) of the form $A \to \alpha$ with $A \in V$ and $\alpha \in (V \cup \Sigma)^*$, and $\mathcal{F} \subseteq P(V)$ is the *Müller acceptance condition.*

A *derivation tree* of the above grammar G is a mapping $t : \text{dom}(t) \to \Sigma \cup V \cup \{\varepsilon\}$ where $T = \text{dom}(t)$ is a tree domain satisfying the following conditions:

- Each inner node $x \in T$ is labeled by some nonterminal A, i.e., $t(x) = A$ for some $A \in V$;
- For any node $x \in T$ and $i \in \mathbb{N}$, $x \cdot i \in T$ if and only if $1 \leq i \leq \deg(x)$;
- For any inner node $x \in T$ exactly one of the following cases holds:
  1. either there exists a production $A \to X_1 \dots X_k$ in P, where $A = t(x)$, for each $1 \leq i \leq k$, $X_i = t(x \cdot i)$ is a member of $V \cup \Sigma$, and for each $i \geq 1$, $x \cdot i$ is in T if and only if $i \leq k$;
  2. or $x \cdot i \in T$ if and only if $i = 1$, $t(x \cdot 1) = \varepsilon$ and $A \to \varepsilon$ is in P, where $A = t(x)$.
- Finally, *t satisfies the Müller acceptance condition* $\mathcal{F}$: for each infinite (maximal) path π of T the set

$$\text{InfLab}_t(\pi) = \{A \in V : A = t(x) \text{ for infinitely many } x \in \pi\}$$

  is a member of $\mathcal{F}$.

The *frontier* of a derivation tree t is the word $\text{fr}(t) = (U, \leq, \lambda)$ where $U \subseteq \text{dom}(t)$ is the set of those leaves of t which are labeled in $V \cup \Sigma$, $\leq$ is the restriction of the lexicographic order to U, and λ is the restriction of t onto U.

We write $X \Rightarrow_G^\infty u$ for a symbol $X \in V \cup \Sigma$ and a word $u \in (V \cup \Sigma)^\infty$ if there exists a derivation tree t with root symbol X and $\text{fr}(t) = u$. If there is a *finite* derivation tree with these properties, we write $X \Rightarrow_G^* u$. When G is clear from the context, we omit the subscripts. For a nonterminal $A \in V$, let $L^\infty(G, A)$ stand for the set $\{w \in \Sigma^\infty : A \Rightarrow^\infty w\}$. The *language generated by* G is $L^\infty(G) = L^\infty(G, S)$.

Definition 1. *A language* $L \subseteq \Sigma^\infty$ *is a* Müller context-free language *or an* MCFL *if* $L = L^\infty(G)$ *for some* MCFG $G = (V, \Sigma, P, S, \mathcal{F})$.

In [11], we studied *Büchi context-free languages*: a Büchi context-free grammar, or BCFG is a system $G = (V, \Sigma, P, S, F)$ where V, Σ, P and S are the same as in the case of an MCFG and $F \subseteq V$ is a Büchi acceptance condition. In this case, a derivation tree t of G has to satisfy the condition that $\mathrm{InfLab}(\pi) \cap F \neq \emptyset$ for each infinite path π of t. The Büchi context-free language, or BCFL generated by the above BCFG G is $L^\infty(G) = \{\mathrm{fr}(t) \in \Sigma^\infty : t \text{ is a derivation tree of } G\}$.

Example 1. The BCFG $G = (V, \Sigma, P, S, \{S\})$ where $P = \{S \to SS, S \to \varepsilon\} \cup \{S \to a : a \in \Sigma\}$ generates Σ^∞, the set of all countable words over Σ.

It is clear that the class of BCFLs is contained in the class of MCFLs; in [11], it has been shown that the inclusion is strict. Below we give two further examples that are MCFLs but not BCFLs.

Example 2. Let Σ be an alphabet and consider the MCFG

$$\begin{aligned} G &= (\{S, I\}, \Sigma, P, S, \{\{I\}\}) \text{ where} \\ P &= \{S \to a : a \in \Sigma \cup \{\varepsilon\}\} \cup \{S \to I\} \cup \{I \to SI\}. \end{aligned}$$

Then $L^\infty(G)$ is the set of all countable well-ordered words over Σ.

To see this, we show that for each countable ordinal α, the set of those words $u \in \Sigma^\infty$ having order type α is a subset of L. For $\alpha = 0$ and $\alpha = 1$ the statement holds, since $S \to \varepsilon$ and $S \to a$, $a \in \Sigma$ are productions of G. Assume the claim holds for each ordinal less than α and let $u \in \Sigma^\infty$ be a word having order type α. Since α can be written as a sum $\alpha_0 + \alpha_1 + \ldots$ of ordinals $\alpha_i < \alpha$, for all $i < \omega$, $u = \prod_{i<\omega} u_i$ for some (possibly empty) words u_i, each having order type less than α. Applying the induction hypothesis we get that $S \Rightarrow^\infty u_i$ for each $i < \omega$, and by $S \Rightarrow^\infty S^\omega$ we have $S \Rightarrow^\infty u$ proving the claim.

We do not show here that L contains well-ordered words only: in Section 4 we give a decision procedure using which one can check whether an MCFL given by an MCFG consists of well-ordered words only.

Example 3. Let Σ be an alphabet and consider the MCFG

$$\begin{aligned} G &= (\{S, I^-, I^+\}, \Sigma, P, S, \{\{I^-\}, \{I^+\}\}) \text{ where} \\ P &= \{S \to a : a \in \Sigma \cup \{\varepsilon\}\} \cup \{S \to I^- I^+\} \cup \{I^+ \to SI^+\} \cup \{I^- \to I^- S\}. \end{aligned}$$

Then $L^\infty(G)$ is the set of all countable scattered words over Σ.

By Corollary 5 in [11] it follows that neither the set of all well-ordered words, nor the set of all scattered words over Σ is a BCFL.

3 A Normal Form

In this section we introduce a notion of normal form for MCFGs, into which each MCFG can be transformed in polynomial space without increasing the size of the grammar.

Given an MCFG $G = (V, \Sigma, P, S, \mathcal{F})$, we say that a nonterminal $A \in V$ is

- *accessible from a nonterminal* B if $B \Rightarrow^\infty uAv$ for some words $u, v \in \Sigma^\infty$; it is called *accessible* if it is accessible from S;
- *(+)-productive* if $A \Rightarrow^\infty u$ for some (nonempty) $u \in \Sigma^\infty$;
- *useful* if it is either productive and accessible; or $A = S$ and $P \subseteq \{S \to \varepsilon\}$;
- *useless* if it is not useful.

Observe that $L^\infty(G)$ is empty for an MCFG $G = (V, \Sigma, P, S, \mathcal{F})$ with $P \not\subseteq \{S \to \varepsilon\}$ if and only if S is useless.

An MCFG $G = (V, \Sigma, P, S, \mathcal{F})$ is in *normal form* if it satisfies the following conditions:

- Either $V = \{S\}$ and $P \subseteq \{S \to \varepsilon\}$, or G only contains nonterminals which are both $+$-productive and accessible;
- If $A \in V$ is a nonterminal with $A \Rightarrow^\infty \varepsilon$, then $A \to \varepsilon$ is a production in P.

In view of the following Theorem 1, it is already PSPACE-hard to decide whether a nonterminal A of an MCFG G is useless or not.

Theorem 1. *The following problem is complete for* PSPACE*: given an* MCFG *G, does it hold that $L^\infty(G) = \emptyset$?*

This result easily follows from the PSPACE-completeness of the corresponding problem for Müller tree automata, cf. [13].

Remark 1. Note that the emptiness problem is solvable in polynomial time for BCFGs, cf. [11].

Since for an MCFG G in normal form we have that $L^\infty(G) = \emptyset$ if and only if its set of productions is empty, a polynomial space algorithm is the best one we can hope for, under standard assumptions of complexity theory, for computing a normal form for an MCFG G. Our next theorem states that this lower bound is achievable.

Theorem 2. *Given an arbitrary* MCFG *G, one can construct an equivalent* MCFG *G' in normal form using polynomial space, moreover, the size of the resulting G' is at most linear in terms of the size of G.*

4 Decision Problems

In this section we investigate the complexity of the following decision problems: given an MCFG G in normal form, decide whether $L^\infty(G)$ is empty; whether it contains at least one infinite word; whether it contains only well-ordered words; whether it contains only finite and ω-words; whether it contains only scattered words; whether it contains only dense words.

Each of these problems can be shown to be decidable by using the fact that languages recognized by Müller tree automata are effectively closed under certain operations such as the Boolean operations together with the fact that the emptiness problem is decidable for Müller tree automata. For example, consider the problem of deciding whether $L^\infty(G)$ contains only finite words, where $G = (V, \Sigma, P, S, \mathcal{F})$. One way of solving this problem is by showing that the tree language L of all derivation trees whose frontier is a finite word in Σ^* is Müller recognizable, and then checking whether $\overline{L} \cap L'$ holds, where L' is the collection of all derivation trees of G whose root is labeled S and whose frontier is in Σ^∞. However, this general method does not provide the simple characterizations obtained by the direct methods developed in this section and the simple polynomial time algorithms. Our main results give a nontrivial upper bound for the complexity of the above questions: all are solvable in polynomial time (in most cases even linear) if the grammar G is in normal form. By Theorem 2 this yields a polynomial space upper bound for these questions in the general case.

The first result of this section deals with the complexity of the emptiness problem, which is trivial when the grammar is in normal form, since if G is an MCFG is in normal form, then $L^\infty(G)$ is empty if and only if G has no productions. For the sake of completeness we record:

Proposition 1. *The following problem can be decided in polynomial time: given an* MCFG $G = (V, \Sigma, P, S, \mathcal{F})$ *with no useless nonterminals, is* $L^\infty(G)$ *empty?*

When $G = (V, \Sigma, P, S, \mathcal{F})$ is an MCFG, let $L^*(G)$ stand for the language $\{w \in \Sigma^* : S \Rightarrow^* w\}$. In general, $L^*(G) \neq L^\infty(G) \cap \Sigma^*$. However, when G is in normal form, equality holds:

Theorem 3. *If* $G = (V, \Sigma, P, S, \mathcal{F})$ *is an* MCFG *in normal form, then* $L^*(G) = L^\infty(G) \cap \Sigma^*$. *Hence, it can be decided in polynomial time whether an* MCFL *given by an* MCFG *in normal form contains at least one finite word.*

Theorem 3 has the following important consequence:

Corollary 1. *A language* $L \subseteq \Sigma^*$ *of finite words is context-free if and only if it is an* MCFL.

Next we describe a construction that will be often used in the sequel. Given an MCFG $G = (V, \Sigma, P, S, \mathcal{F})$, we define a finite edge-labeled multigraph Γ_G as follows. The set of vertices of Γ_G is $V \cup \Sigma$. The edge labels are pairs of the form (α, β) with $\alpha, \beta \in (V \cup \Sigma)^*$. There exists an edge from $A \in V \cup \Sigma$ to $B \in V \cup \Sigma$ labeled (α, β), in notation $A \xrightarrow{\alpha,\beta} B$ if and only if $A \to \alpha B \beta$ is in P.

For an arbitrary set $X \subseteq V \cup \Sigma$ of symbols let $\Gamma_G|_X$ stand for the restriction of Γ_G onto the set X, i.e., for the edge-labeled multigraph with vertex set X and edges $A \xrightarrow{\alpha,\beta} B$ with $A, B \in X$ and $A \to \alpha B \beta \in P$. Observe that a symbol B is accessible from a symbol A (i.e., $A \Rightarrow^* \alpha B \beta$ for some $\alpha, \beta \in (V \cup \Sigma)^*$) if and only if there exists a path from A to B in Γ_G.

Let us call a set $F \in \mathcal{F}$ of nonterminals *viable* if there exists a derivation tree t of G with root symbol S which has an infinite path π such that $\mathrm{InfLab}(\pi) = F$.

Our first observation concerning Γ_G is that viability of a set $F \in \mathcal{F}$ can be checked in polynomial time using well-known graph theoretic algorithms.

Proposition 2. *Let $G = (V, \Sigma, P, S, \mathcal{F})$ be an* MCFG *without useless nonterminals. A set $F \in \mathcal{F}$ is viable in G if and only if $\Gamma_G|_F$ is strongly connected. Hence, viability of a set $F \in \mathcal{F}$ can be decided in polynomial time.*

Based on that viability and accessibility are both decidable in polynomial time, we get polynomial time algorithms for deciding whether an MCFL contains an infinite word; whether it contains a word which is not well-ordered; or whether it contains an infinite word which is not an ω-word, as stated in the following characterization theorems.

Theorem 4. *Let $G = (V, \Sigma, P, S, \mathcal{F})$ be an* MCFG *in normal form and let $X \in V$ be a nonterminal. $L^\infty(G, X)$ contains an infinite word if and only if there exists an viable set $F \in \mathcal{F}$ such that some (hence each) member of F is accessible from X and there is an edge $A \xrightarrow{\alpha,\beta} B$ in $\Gamma_G|_F$ and $\alpha\beta \neq \varepsilon$.*

Hence, it can be decided in polynomial time whether an MCFL *given by an* MCFG *in normal form contains finite words only.*

Theorem 5. *Let $G = (V, \Sigma, P, S, \mathcal{F})$ be an* MCFG *in normal form. $L^\infty(G)$ contains a word which is not well-ordered if and only if there exists a viable set $F \in \mathcal{F}$ and an edge $A \xrightarrow{\alpha,\beta} B$ in $\Gamma_G|_F$ with $\beta \neq \varepsilon$.*

Hence, it can be decided in polynomial time whether an MCFL *given by an* MCFG *in normal form contains well-ordered words only.*

Theorem 6. *Let $G = (V, \Sigma, P, S, \mathcal{F})$ be an* MCFG *in normal form. $L^\infty(G)$ contains an infinite word which is not an ω-word if and only if there exists a viable set $F \in \mathcal{F}$, nonterminals $A, B \in V$ satisfying the following conditions:*

- *there exists an edge $A \xrightarrow{\alpha,\beta} B$ with $\beta \neq \varepsilon$;*
- *some (and hence each) member of F is accessible from B in Γ_G;*
- *there exists some edge $C \xrightarrow{\alpha',\beta'} D$ in $\Gamma_G|_F$ with $\alpha'\beta' \neq \varepsilon$.*

Hence, it can be decided in polynomial time whether an MCFL *given by an* MCFG *in normal form contains finite and ω-words only.*

It is also decidable in polynomial time whether an MCFL given by an MCFG in normal form contains scattered words only, as stated in the following theorem.

Theorem 7. *Let $G = (V, \Sigma, P, S, \mathcal{F})$ be an* MCFG *in normal form. The following are equivalent:*

1. *$L^\infty(G)$ contains a quasi-dense word (i.e., a word which is not scattered).*
2. *There is a derivation tree t (whose frontier is in Σ^∞) such that the full (infinite) binary tree can be embedded in t, and each infinite subtree of t contains a leaf labeled by a terminal.*
3. *There is a derivation tree t (whose frontier is in Σ^∞) such that the full (infinite) binary tree can be embedded in t.*
4. *There is a nonterminal A and a finite derivation tree t with root label A which has two leaves x_1 and x_2 labeled A with the following property: there is a set $F \in \mathcal{F}$ such that the set of labels of nonterminals along the path from the root to x_i is equal to F, for $i = 1, 2$.*

Hence, it is decidable in polynomial time whether $L^\infty(G)$ contains scattered words only.

Finally, it is also decidable in polynomial time whether an MCFG generates dense words only.

Theorem 8. *It can be decided in polynomial time whether an* MCFL *given by an* MCFG *in normal form contains dense words only.*

5 MCFLs of Scattered Words

For each countable ordinal α we define a set H_α of order types of scattered countable linear orders as follows. H_0 consists of the finite nonzero order types and for any $\alpha > 0$, H_α is the smallest set of order types which is closed under finite sums and contains all the order types of the form $\sum_{i \in I} \tau_i$, where I is either finite or one of the order types ω, $-\omega$ or $-\omega + \omega$, and each τ_i is in H_{β_i} for some ordinal $\beta_i < \alpha$.

It is known, cf. [16] that a nonempty countable linear order is scattered if and only if it is contained in some H_α for some (countable) ordinal α. Given a nonempty countable scattered linear order P, let $H(P)$ stand for its *rank*, i.e. the smallest ordinal α for which the order type of P belongs to H_α. When u is a scattered word, its rank $H(u)$ is defined as the rank of its underlying linear order.

This section aims at stating the following property of MCFLs: if L is an MCFL consisting of scattered words, then one of the following cases holds: either there exists an integer n such that the rank of each member of L is at most n, or L contains words having rank at least α for every countable ordinal α. Moreover, it can be decided in polynomial time whether i) or ii) holds, and if i) holds, a least such integer n can also be computed, still in polynomial time (if L is given by an MCFG in normal form).

In the rest of this section we consider MCFLs consisting of scattered words only. For an MCFG $G = (V, \Sigma, P, S, \mathcal{F})$ and symbols $X, Y \in V \cup \Sigma$, let us write $X \rightsquigarrow^\infty Y$ if $X \Rightarrow^\infty u$ for some $u \in (V \cup \Sigma)^\infty$ with Y occurring infinitely many times in u. We call a nonterminal $A \in V$ *reproductive* if $A \rightsquigarrow^\infty A$.

Our first result is that the relation $\rightsquigarrow^\infty$ is computable in polynomial time.

Proposition 3. *Suppose $G = (V, \Sigma, P, S, \mathcal{F})$ is an* MCFG *in normal form. Then $A \rightsquigarrow^\infty B$ holds for the symbols A and B if and only if there exists a viable set $F \in \mathcal{F}$ such that some (and hence each) member of F is accessible from A, moreover, B is accessible from some symbol occurring in at least one of the edge labels of $\Gamma_G|_F$.*

The next proposition states that if there exists a reproductive nonterminal, then words of arbitrarily large rank are contained in the generated language.

Proposition 4. *Suppose $G = (V, \Sigma, P, S, \mathcal{F})$ is an* MCFG *in normal form such that $L = L^\infty(G)$ contains only scattered words. If some nonterminal $A \in V$ is reproductive, then for each countable ordinal α, L contains a word u_α with $H(u_\alpha) \geq \alpha$.*

On the other hand, if there is no such a nonterminal, then there exists a finite and computable upper bound of the rank of the members of the language:

Proposition 5. *Suppose $G = (V, \Sigma, P, S, \mathcal{F})$ is an* MCFG *in normal form containing no reproductive nonterminals such that $L = L^\infty(G)$ contains scattered words only.*

Then the rank of each nonempty word in $L = L^\infty(G)$ is at most $|V|$. Moreover, for each symbol $X \in V \cup \Sigma$, the maximal rank $\#(X) = \max\{H(w) : w \in L^\infty(G, X)\}$ can be computed in polynomial time.

Propositions 4 and 5 immediately yield:

Theorem 9. *Suppose L is an* MCFL *containing only scattered words. Then one of the following cases holds:*

i) either there exists a finite bound n such that each word in L has rank at most n;
ii) or for any countable ordinal α, there exists a word in L with rank at least α.

Moreover, it can be decided in polynomial time whether i) or ii) holds, and if i) holds, the least such bound n can be computed in polynomial time, if L is given by an MCFG *in normal form.*

6 Closure Properties

Most of the closure properties of this section are rather standard, but we include them for completeness. Set theoretic operations on languages in Σ^∞ have their standard meaning. Suppose that $\Sigma = \{a_1, \ldots, a_n\}$, $L \subseteq \Sigma^\infty$ and $L_i \subseteq \Delta^\infty$ for all $1 \le i \le n$. Then the substitution $L[a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n]$ is defined in the expected way. Below we define some other operations.

Let $L, L_1, L_2, \ldots, L_m \subseteq \Sigma^\infty$. Then we define:

1. $L_1 L_2 = \{ab\}[a \leftarrow L_1, b \leftarrow L_2] = \{uv : u \in L_1, v \in L_2\}$.
2. $L^* = \{a\}^*[a \leftarrow L] = \{u_1 \ldots u_n : n < \omega,\ u_i \in L\}$.
3. $L^\omega = \{a^\omega\}[a \leftarrow L] = \{u_0 u_1 \ldots u_n \ldots : u_i \in L\}$.
4. $L^{-\omega} = \{a^{-\omega}\}[a \leftarrow L] = \{\ldots u_n \ldots u_1 u_0 : u_i \in L\}$.
5. $(L_1, \ldots, L_m)^\eta = \{(a_1, \ldots, a_m)^\eta\}[a_1 \leftarrow L_1, \ldots, a_m \leftarrow L_m]$.
6. $L^\infty = \{a\}^\infty[a \leftarrow L]$.

The above operations are respectively called *concatenation*, *star*, ${}^\omega$*-power*, ${}^{-\omega}$*-power*, ${}^\eta$*-power*, and ${}^\infty$*-power*.

The class of MCFLs enjoys closure properties similar to the class CFL of context-free languages (of finite words). In particular, it is closed under substitution (thus it is closed under the operations defined above).

Proposition 6. *The class of* MCFL*s is effectively closed under substitution, i.e., when $L \subseteq \{a_1, \ldots, a_n\}^\infty$, $L_1, \ldots, L_n \subseteq \Sigma^\infty$ are* MCFL*s each given by an* MCFG*, then an* MCFG *G with $L^\infty(G) = L[a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n]$ can be given effectively.*

In particular, the class of MCFL*s is closed under binary set union, concatenation, star, ${}^\omega$-, ${}^\eta$-, ${}^\infty$-, and $({}^{-\omega})$-power.*

It is well-known that CFL is neither closed under intersection nor under complementation. Hence, by Corollary 1 we have:

Proposition 7. *The class of* MCFL*s is neither closed under intersection nor under complementation.*

Proposition 8. *If L is an* MCFL*, then the languages $L^r = \{u^r : u \in L\}$,* $\mathrm{Pre}(L) = \{u : \exists v\ uv \in L\}$, $\mathrm{Suf}(L) = \{u : \exists v\ vu \in L\}$, $\mathrm{In}(L) = \{u : \exists v, w\ vuw \in L\}$ *and* $\mathrm{Sub}(L)$*, the collection of all words u such that there is an embedding $u \hookrightarrow v$ for some $v \in L$ are also* MCFL*s.*

7 Conclusion, Open Questions

We have defined Müller context-free grammars (MCFGs) generating languages of countable words, called MCFLs. The class of MCFLs is clearly closed under substitution and thus enjoys good closure properties. We have studied several decision problems for MCFLs, mainly motivated by order theoretic properties, and in each case we have found a polynomial time algorithm for MCFGs in normal form. The transformation of an arbitrary grammar into normal form requires polynomial space, but the size of the grammar produced by the algorithm is linear in the size of the input grammar. Among the decision problems, we showed that it is decidable in polynomial time whether an MCFG in normal form generates a language of well-ordered, or scattered, or dense words. We have established a limitedness property: If an MCFL contains only scattered words, then either the rank of each word of the language is bounded by a fixed integer n, or for each countable ordinal α there is a word in the language of rank at least α. Moreover, we have shown that it is decidable which of the two cases applies.

In an earlier paper we studied Büchi context-free languages, or BCFLs. While every BCFL is an MCFL, there exists an MCFL of scattered, or even well-ordered words that is not a BCFL. It remains for future research to answer the question whether there is an MCFL consisting of dense words that is not a BCFL. On the other hand, it is not difficult to show that every MCFL consisting of finite or ω-words is a BCFL. By a result in [11], it then follows that an ω-language is an MCFL if and only if it is context-free in the sense of Cohen and Gold [9].

The equality problem for BCFLs is undecidable, [11], thus it is also undecidable for MCFLs. We have not yet studied the question of deciding whether an MCFG generates a BCFL. Also, it would be interesting to know whether there is an MCFL of scattered words of rank bounded by an integer n that is not a BCFL.

References

1. Bedon, N.: Finite automata and ordinals. Theor. Comp. Sci. 156, 119–144 (1996)
2. Bès, A., Carton, O.: A Kleene theorem for languages of words indexed by linear orderings. In: De Felice, C., Restivo, A. (eds.) DLT 2005. LNCS, vol. 3572, pp. 158–167. Springer, Heidelberg (2005)

3. Bloom, S.L., Ésik, Z.: Deciding whether the frontier of a regular tree is scattered. Fund. Inform. 55, 1–21 (2003)
4. Boasson, L.: Context-free sets of infinite words. In: Theoretical Computer Science (Fourth GI Conf., Aachen, 1979). LNCS, vol. 67, pp. 1–9. Springer, Heidelberg (1979)
5. Bruyère, V., Carton, O.: Automata on linear orderings. J. Comput. System Sci. 73, 1–24 (2007)
6. Bruyere, V., Carton, O., Sénizergues, G.: Tree Automata and Automata on Linear Orderings. ITA 43, 321–338 (2009)
7. Büchi, J.R.: The monadic second order theory of ω_1. In: Decidable theories, II. LNM, vol. 328, pp. 1–127. Springer, Heidelberg (1973)
8. Choueka, Y.: Finite automata, definable sets, and regular expressions over ω^n-tapes. J. Comput. System Sci. 17(1), 81–97 (1978)
9. Cohen, R.S., Gold, A.Y.: Theory of ω-languages, parts one and two. Journal of Computer and System Science 15, 169–208 (1977)
10. Courcelle, B.: Frontiers of infinite trees. RAIRO Theor. Inf. 12, 319–337 (1978)
11. Ésik, Z., Iván, S.: Context-Free Languages of Countable Words. In: Leucker, M., Morgan, C. (eds.) Theoretical Aspects of Computing - ICTAC 2009. LNCS, vol. 5684, pp. 185–199. Springer, Heidelberg (2009)
12. Heilbrunner, S.: An algorithm for the solution of fixed-point equations for infinite words. RAIRO Theor. Inf. 14, 131–141 (1980)
13. Hunter, P., Dawar, A.: Complexity Bounds for Regular Games. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 495–506. Springer, Heidelberg (2005)
14. Montalbán, A.: On the equimorphism types of linear orderings. The Bulletin of Symbolic Logic 13(1), 71–99 (2007)
15. Nivat, M.: Sur les ensembles de mots infinis engendrés par une grammaire algébrique (French). RAIRO Inform. Théor. 12(3), 259–278 (1978)
16. Rosenstein, J.G.: Linear Orderings. Academic Press, London (1982)
17. Thomas, W.: Automata on Infinite Objects. In: Handbook of Theoretical Computer Science. Formal Models and Semantics, vol. B, pp. 133–192. Elsevier/MIT Press (1990)
18. Wojciechowski, J.: Classes of transfinite sequences accepted by finite automata. Fundamenta Informaticæ 7, 191–223 (1984)
19. Wojciechowski, J.: Finite automata on transfinite sequences and regular expressions. Fundamenta Informaticæ 8, 379–396 (1985)

Minimization of Deterministic Bottom-Up Tree Transducers

Sylvia Friese[1], Helmut Seidl[1], and Sebastian Maneth[2]

[1] Technische Universität München, Garching, Germany
[2] NICTA and University of New South Wales, Sydney, Australia

Abstract. We show that for every deterministic bottom-up tree transducer, a unique equivalent transducer can be constructed which is minimal. The construction is based on a sequence of normalizing transformations which, among others, guarantee that non-trivial output is produced as early as possible. For a deterministic bottom-up transducer where every state produces either none or infinitely many outputs, the minimal transducer can be constructed in polynomial time.

Keywords: Bottom-up Tree Transducers, Minimization, Normal form.

1 Introduction

Top-down and bottom-up tree transducers were invented in the 1970s by Rounds and Thatcher [21,23], and Thatcher [24], respectively. Their expressive powers are incomparable, both for nondeterministic and deterministic transducers [4], similar to the fact that left-to-right and right-to-left string transducers are incomparable (see Section IV.2 in [2]). In 1980 it was shown that equivalence for deterministic transducers is decidable both in the top-down [11] and bottom-up case [25]. Later, a polynomial-time algorithm for single-valued bottom-up transducers has been provided [22]. Recently, it was shown that for total deterministic top-down tree transducers, equivalence can be decided in polynomial time [10]. The proof relies on a new canonical normal form for such transducers, called the *earliest* normal form (inspired by the earliest string transducers of Mohri [20]). The question arises whether deterministic bottom-up tree transducers (btt's) also allow for a similar canonical normal. In this paper we give an affirmative answer to this question.

We show that for every btt there is a unique equivalent bottom-up transducer in normal form. The main idea is to unite states which are equivalent with respect to their behavior on contexts. There are several obstacles for this basic approach. Finite sets of output trees for a given state could be assembled in several different ways. Even if infinitely many outputs may occur, still bounded parts of it could be produced earlier or later. Due to the tree structure of outputs, the output for contexts of states could agree only for one particular pair of output subtrees. In order to remove these obstacles, we present a sequence of normal forms of increasing strength. Generating the unique normal form for a given btt therefore proceeds in four steps: (1) first, we make the transducer *proper*, i.e., we remove all output from states which only produce finitely many different outputs. The output for such states is postponed until a state with infinitely many different outputs or the final function at the root of the input tree. This is similar to the *proper normal form* of [1,8] (which removes states that produce finitely many outputs, using regular look-ahead). (2) We make the transducer *earliest*, i.e., every state with infinitely many outputs produces output as early as possible during the translation. (3) We remove pairwise *ground context unifiers*. It is only in step (4) that we can apply minimization in the usual way by merging states that are isomorphic. Steps (2)–(4) can be done in polynomial time, i.e., given a proper transducer, its unique minimal transducer is constructed in polynomial time. Constructing a proper transducer (Step 1) may take double-exponential time in the worst case.

Besides equivalence checking, there are many more applications of a canonical normal form. For instance, it allows for a Myhill-Nerode style theorem, which, in turn can be used to build a Gold-style learning algorithm; see [19] where both were done for deterministic top-down tree transducers (ttt's). As another example, the normal form can be used to decide certain (semantic) subclasses of btt's; e.g., we can decide whether a given btt is equivalent to a relabeling, using the normal form. This provides an alternative proof of [16], for the deterministic case.

Related Work. A valid generalization of both, btt's and ttt's is the deterministic top-down tree transducer with regular look-ahead [5]. Even though the equivalence problem for ttt's with regular look-ahead is easily reduced to the one for ttt's [10], it is an intriguing open problem whether ttt's with regular look-ahead have a canonical normal form. Another related model of transformation is the attribute grammar [17], seen as a tree transducer [13,15]. For attributed tree transducers, decidability of equivalence is still an open problem, but, for the special subclass of "nonnested, separated" attribute grammars (those which can be evaluated in one strict top-down phase followed by one strict bottom-up phase) equivalence is known to be decidable [3]. This class strictly includes ttt's (but not btt's [14]).

There are several other interesting incomparable classes of tree translations for which equivalence is known to be decidable, but no normal form (and no complexity) is known. For instance, MSO-definable tree translations [9]. This class coincides with single-use restricted attribute grammars or macro tree transducers with look-ahead [7]. Is there a canonical normal form for such transducers? Another interesting generalization are tree-to-*string* transducers. It is a long standing open problem [6] whether or not deterministic top-down tree-to-string transducers (ttst's) have decidable equivalence. Recently, for the subcase of *non-copying* ttst's, a unique normal form similar to the earliest normal form was presented [18]. Can their result be extended to the finite-copying case? Another recent result states that functional visibly pushdown transducers have decidable equivalence [12]. This class is closely related to non-copying ttst's. It raises the question whether our normal form for btt's can be extended to functional (but nondeterministic) bottom-up tree transducers.

2 Preliminaries

Bottom-up tree transducers work on ranked trees. In a ranked tree, the number of children of a node is determined by the *rank* of the symbol at that node. A ranked alphabet Σ consists of finitely many symbols. Each symbol $a \in \Sigma$ is equipped with a rank in

$\{0, 1, \dots\}$, where rank 0 indicates that a is the potential label of a leaf. We assume that a ranked alphabet contains at least one symbol of rank 0. The set $\mathcal{T}_\Sigma$ of ranked *trees* over Σ is the set of strings defined by the EBNF with rules $t ::= a(\underbrace{t, \dots, t}_{k \text{ times}})$ for all $k \geq 0$ and $a \in \Sigma$ of rank k. We also write a for the tree $a()$. Note that, since there is at least one symbol of rank 0, $\mathcal{T}_\Sigma \neq \emptyset$. We use the words *tree* and *term* interchangeably.

We consider trees possibly containing a dedicated variable $y \notin \Sigma$ (of rank 0). Let $\mathcal{T}_\Sigma(y)$ denote this set. On $\mathcal{T}_\Sigma(y)$, we define a binary operation "$\cdot$" by: $t_1 \cdot t_2 = t_1[t_2/y]$, i.e., the substitution by t_2 of every occurrence of the variable y in t_1. Note that the result is a ground tree, i.e., does not contain y, iff either $t_1 \in \mathcal{T}_\Sigma$ or $t_2 \in \mathcal{T}_\Sigma$. Moreover, the operation "$\cdot$" is associative with neutral element y. Therefore, the set $\mathcal{T}_\Sigma(y)$ together with the operation "$\cdot$" and y forms a monoid. Let $\hat{\mathcal{T}}_\Sigma(y)$ denote the sub-monoid consisting of all trees which contain at least one occurrence of y. Then $\mathcal{T}_\Sigma(y) = \hat{\mathcal{T}}_\Sigma(y) \cup \mathcal{T}_\Sigma$.

Proposition 1. *[6]*

1. *Let $s, s', t_1, t_2, t'_1, t'_2 \in \mathcal{T}_\Sigma(y)$ with $t_1 \neq t_2$ and $t'_1 \neq t'_2$. Assume that the two equalities $s \cdot t_1 = s' \cdot t'_1$ and $s \cdot t_2 = s' \cdot t'_2$ hold. Then one of the following two assertions are true:*
   (a) $s, s' \in \mathcal{T}_\Sigma$ and $s = s'$; or
   (b) Both trees s and s' contain an occurrence of y, i.e., are from $\hat{\mathcal{T}}_\Sigma(y)$, and $s \cdot u = s'$ or $s = s' \cdot u$ for some $u \in \hat{\mathcal{T}}_\Sigma(y)$.
2. *The sub-monoid $\hat{\mathcal{T}}_\Sigma(y)$ is free.*

Consider the set $\hat{\mathcal{T}}_\Sigma(y)_\perp = \hat{\mathcal{T}}_\Sigma(y) \cup \{\perp\}$ of all trees containing at least one occurrence of the variable y enhanced with an extra bottom element $\perp$ (not in $\Sigma \cup \{y\}$). On this set, we define a partial ordering by $\perp \sqsubseteq t$ for all t, and $t_1 \sqsubseteq t_2$ for $t_1, t_2 \in \hat{\mathcal{T}}_\Sigma(y)$ iff $t_1 = t' \cdot t_2$ for a suitable $t' \in \hat{\mathcal{T}}_\Sigma(y)$. The greatest element with respect to this ordering is y while the least element is given by $\perp$. With respect to this ordering, we observe:

1. Every $t \in \hat{\mathcal{T}}_\Sigma(y)$ has finitely many upper bounds.
2. For every $t_1, t_2 \in \hat{\mathcal{T}}_\Sigma(y)_\perp$, there exists a least upper bound $t_1 \sqcup t_2$ in $\hat{\mathcal{T}}_\Sigma(y)_\perp$.

Since $\hat{\mathcal{T}}_\Sigma(y)_\perp$ also has a least element, namely $\perp$, we conclude that $\hat{\mathcal{T}}_\Sigma(y)_\perp$ is a *complete* lattice satisfying the ascending chain condition, i.e., every set $X \subseteq \hat{\mathcal{T}}_\Sigma(y)_\perp$ has a least upper bound $t = \bigsqcup X$, and there are no infinite strictly ascending sequences $\perp \sqsubseteq t_1 \sqsubseteq t_2 \sqsubseteq \dots$. We call a tree $t \in \hat{\mathcal{T}}_\Sigma(y)$ *irreducible* if $t \neq y$ and $t \sqsubseteq t'$ only holds for $t' \in \{y, t\}$.

Let $\top \notin \Sigma \cup \{y\}$ be a new symbol. Assume that $c_1, c_2 \in \mathcal{T}_\Sigma(y)$ are trees, and that there are trees $s_1, s_2 \in \mathcal{T}_\Sigma(y) \cup \{\top\}$ such that $c_1 \cdot s_1 = c_2 \cdot s_2$. Note, that $c_i \cdot s_i = c_i$, if $c_i \in \mathcal{T}_\Sigma$. We call c_1, c_2 *unifiable* and $\langle s_1, s_2 \rangle$ a unifier of c_1, c_2.

We consider the set $\mathbb{D} = (\{y\} \times \hat{\mathcal{T}}_\Sigma(y)) \cup (\hat{\mathcal{T}}_\Sigma(y) \times \{y\}) \cup (\mathcal{T}_\Sigma \cup \{\top\})^2 \cup \{\perp\}$ of candidate unifiers. The set $\mathbb{D}$ forms a complete lattice w.r.t. the ordering $\leq$ defined by

– $\perp \leq d \leq \langle \top, \top \rangle$ for all $d \in \mathbb{D}$,
– $(d_1, d_2) \leq (d'_1, d'_2)$ if $d_i = d'_i \cdot s$ for all $i \in \{1, 2\}$ for some tree $s \in \mathcal{T}_\Sigma \cup \{y\}$, and
– $(d_1, d_2) \leq (d_1, \top)$ and $(d_1, d_2) \leq (\top, d_2)$ if $d_1, d_2 \in \mathcal{T}_\Sigma$.

The *most general unifier* $\mathsf{mgu}(c_1, c_2) \in \mathbb{D}$ for trees $c_1, c_2 \in \mathcal{T}_\Sigma(y)$ is the greatest unifier of c_1, c_2 w.r.t. the ordering $\leq$. It is $\perp$, if c_1, c_2 are not unifiable. Furthermore, for a set of pairs $C \subseteq \mathcal{T}_\Sigma(y)^2$ the most general unifier $\mathsf{mgu}(C)$ is the least upper bound of the unifiers of pairs in C, i.e., $\mathsf{mgu}(C) = \bigvee\{\mathsf{mgu}(c_1, c_2) \mid (c_1, c_2) \in C\}$.

For $k \in \{0, 1, \ldots\}$ we denote the set $\{x_1, \ldots, x_k\}$ of k distinct variables by $\mathcal{X}_k$. We consider trees with variables at leaves, i.e., trees in $\mathcal{T}_{\Sigma \cup \mathcal{X}_k}$ where each variable x_i has rank 0. Let $z \in \mathcal{T}_{\Sigma \cup \mathcal{X}_k}$ be such a tree. We abbreviate by $z[z_1, \ldots, z_k]$ the substitution $z[z_1/x_1, \ldots, z_k/x_k]$ of trees z_i for the variables x_i $(i = 1, \ldots, k)$ in the tree z.

2.1 Bottom-Up Tree Transducers

A *deterministic bottom-up tree transducer* (btt for short) is a tuple $T = (Q, \Sigma, \Delta, R, F)$, where

- Q is a finite set of states,
- Σ and Δ are ranked input and output alphabets, respectively, disjoint with Q,
- R is the (possibly partial) transition function, and
- $F : Q \to \mathcal{T}_\Delta(y)$ is a partial function mapping states to final outputs.

For every input symbol $a \in \Sigma$ of rank k and sequence $q_1, \ldots, q_k$ of states, the transition function R contains at most one transition, which is denoted by $a(q_1, \ldots, q_k) \to q(z)$ where $q \in Q$ and $z \in \mathcal{T}_{\Delta \cup \mathcal{X}_k}$.

For every input symbol $a \in \Sigma$ of rank k and sequence of states $q_1 \ldots q_k$ of Q, let $R(a, q_1 \ldots q_k)$ be the right-hand side of the transition for a and $q_1 \ldots q_k$, if it is defined, and let $R(a, q_1 \ldots q_k)$ be undefined otherwise.

The transducer is *total* if $R(a, q_1 \ldots q_k)$ is defined for all $k \geq 0$, $a \in \Sigma$ of rank k, and sequences of states $q_1 \ldots q_k$. The *size* of T, denoted by $|T|$, is the sum of sizes (= number of symbols) of its final outputs and of the left-hand sides and right-hand sides of its transitions.

Assume that $t \in \mathcal{T}_\Sigma(y)$ and $q \in Q$. The *result* $[\![t]\!]_q^T$ of a computation of T on input t when starting in state q at variable leaves y is defined by induction on the structure of t:

$$\begin{aligned} [\![y]\!]_q^T &= q(y) \\ [\![a(s_1, \ldots, s_k)]\!]_q^T &= q'(z[z_1, \ldots, z_k]) \\ &\quad \text{if } \forall i \ [\![s_i]\!]_q^T = q_i(z_i) \text{ and } R(a, q_1 \ldots q_k) = q'(z) \ . \end{aligned}$$

If $[\![t]\!]_q^T = q'(z')$, then z' is called the *output* produced for t. Note that the function $[\![\,.\,]\!]_q^T$ may not be defined for all trees t. The superscript ^{T} can be omitted if T is clear from the context. If $t \in \mathcal{T}_\Sigma$ we also omit the subscript q, i.e., we write $[\![t]\!]^T$ for $[\![t]\!]_q^T$.

The *image* $\tau_q^T(t)$ of the tree t is then defined by $\tau_q^T(t) = z' \cdot z$ iff $[\![t]\!]_q^T = q'(z)$ for some state q' with $F(q') = z'$. We omit the subscript q if the tree t does not contain the variable y.

We say that two btt's T and T' are *equivalent* when they describe the same transformation, i.e., for all $t \in \mathcal{T}_\Sigma$, $\tau^T(t)$ is defined iff $\tau^{T'}(t)$ is defined and are equal.

We also use the following notation. The *language* $\mathcal{L}^T(q)$ of a state q is the set of all ground input trees by which q is reached, i.e., $\mathcal{L}^T(q) = \{t \mid \exists s \in \mathcal{T}_\Delta : [\![t]\!]^T = q(s)\}$. A *context* c is a tree $c \in \hat{\mathcal{T}}_\Sigma(y)$ which contains exactly one occurrence of y. Let $\mathcal{C}_\Sigma$ be the set of all contexts. A tree $c \in \mathcal{C}_\Sigma$ is a context of a state q, if $\tau_q^T(c)$ is defined. Let $\mathcal{C}^T(q)$ denote the set of all contexts of state q. The *length* of a context c is the length of the path from the root to y. If context c has length n, then there are irreducible trees $c_1, \ldots, c_n$ such that $c = c_1 \cdot c_2 \cdot \ldots \cdot c_n$.

3 Trim Transducers

Transducers may contain useless transitions or states and we want to get rid of these while preserving the described transformation. A state q of a btt is *reachable*, if the language $\mathcal{L}^T(q)$ is non-empty. A state q is *meaningful*, if q has at least one context, i.e., $\mathcal{C}^T(q)$ is non-empty. Furthermore, the output at state q is *potentially useful*, if there is a context c of q such that the image $\tau_q^T(c)$ contains the variable y. Otherwise, the output at q is called *useless*. A bottom-up tree transducer T is called *trim* if T has the following properties: (1) every state is reachable, (2) every state is meaningful, (3) if the output at a state q is useless, then for each transition $a(q_1, \ldots, q_k) \to q(z)$ leading into state q, $z = *$. In this definition, $*$ is a special output symbol which does not occur in any image produced by T. It is well-known that each btt is equivalent to a trim btt.

Proposition 2. *For every bottom-up tree transducer T a bottom-up tree transducer T' can be constructed in polynomial time with the following properties:*

1. *$T'$is equivalent to T;*
2. *$|T'| \leq |T|$;*
3. *T' is trim.*

In the remainder of the paper we consider trim transducers only. For a trim transducer T with set Q of states, we denote by Q_* the set of states with useless output, i.e., for which the output is always $*$.

4 Proper Transducers

For a given trim transducer, there is not necessarily a unique minimal equivalent btt. Finite outputs of subtrees may be distributed over different states.

Example 3. Assume that $\Sigma = \{A, C, H, K, L\}$, $\Delta = \{a, b, h, l, *\}$, $Q_1 = \{q_0, q_1, q_2\}$, and $Q_2 = \{q_0', q_1', q_2'\}$. Consider the following transducers:

$$\begin{array}{ll|ll}
\multicolumn{2}{l|}{T_1 = (Q_1, \Sigma, \Delta, R_1, F_1) \text{ with}} & \multicolumn{2}{l}{T_2 = (Q_2, \Sigma, \Delta, R_2, F_2) \text{ with}} \\
A(q_1) \to q_0(x_1) & H \to q_1(b) & A(q_1') \to q_0'(b) & H \to q_1'(h) \\
A(q_2) \to q_0(b) & K \to q_1(a) & A(q_2') \to q_0'(a) & K \to q_2'(*) \\
C(q_1) \to q_0(h) & L \to q_2(*) & C(q_1') \to q_0'(x_1) & L \to q_1'(l) \\
C(q_2) \to q_0(l) & F_1(q_0) = y & C(q_2') \to q_0'(h) & F_2(q_0') = y
\end{array}$$

The transducer T_1 to the left produces different outputs for H and K while leading to the same state, and produces no output for L while leading into a different state. The transducer T_2 to the right, on the other hand, produces different outputs for H and L

leading into the same state and produces no output for K while leading into a different state. Both transducers are trim and describe the same transformation τ:

$$A(H) \to b \quad A(L) \to b \quad A(K) \to a \quad C(H) \to h \quad C(L) \to l \quad C(K) \to h$$

It is not clear how a unique normal form for τ with less than three states could look like. $\square$

Let T denote a trim transducer with set of states Q. A state $q \in Q$ is called *essential* if the set of results $\{[\![t]\!]^T \mid t \in \mathcal{L}^T(q)\}$ for input trees reaching q is infinite. Otherwise, q is called *inessential*. Note that all states of the transducers in Example 3 are inessential.

A *proper* transducer postpones outputs at inessential states. The trim transducer T is called *proper* if every inessential state does not produce any output, i.e., is in Q_*. For every trim transducer there exists an equivalent proper transducer:

Proposition 4. *[1,8] For every trim btt T a btt T' can be constructed with the following properties:*

1. *T' is equivalent to T;*
2. *T' is proper.*
3. *$|T'| \leq \Gamma \cdot |T|$*

where Γ is the sum of sizes of all outputs produced for inessential states of T.

In the worst case, an inessential state may have exponentially many outputs – even if the input alphabet has maximal rank 1. In case, that both input and output alphabets have symbols of ranks greater than 1, doubly exponentially many outputs of inessential states are possible.

Proof (of Proposition 4 - Sketch). The inessential states are determined by considering the *dependence graph* $G_T = (V, E)$ where V is the set of states of T, and $(q_i, q) \in E$ if there is a transition $a(q_1, \ldots, q_k) \to q(z)$ in T and x_i occurs in z. We split each inessential state q into new states $\langle q, z\rangle$ where $q(z)$ is a possible result for some input tree $t \in \mathcal{L}^T(q)$. If q occurs on a left-hand side of a transition as the state for the i-th argument where the state in the right-hand side is essential, a new transition is generated where q is replaced with $\langle q, z\rangle$ and the corresponding variable x_i is replaced with z. Also, the final function F' should be modified accordingly for inessential states. $\square$

Example 5. Consider again the transducer T_1 (to the left) of Example 3. The dependence graph G_{T_1} is $(\{q_0, q_1, q_2\}, \{(q_1, q_0)\})$. We determine that all states are inessential. The equivalent proper btt $T_1' = (Q_1', \Sigma, \Delta, R_1', F_1')$ has the following set of states:

$$Q_1' = \{\langle q_0, a\rangle, \langle q_0, b\rangle, \langle q_0, h\rangle, \langle q_0, l\rangle, \langle q_1, a\rangle, \langle q_1, b\rangle, \langle q_2, *\rangle\} .$$

Since every state of Q_1' is inessential, the output is postponed to the final function. Whereas, the right-hand sides of the transitions R_1' are of the form $\langle q, z\rangle\,(*)$, e.g.,

$$H \to \langle q_1, b\rangle\,(*) \quad A(\langle q_1, b\rangle) \to \langle q_0, b\rangle\,(*) \quad C(\langle q_1, b\rangle) \to \langle q_0, h\rangle\,(*) .$$

For the final function, we get

$$F_1'(\langle q_0, b\rangle) = b \qquad F_1'(\langle q_0, a\rangle) = a \qquad F_1'(\langle q_0, h\rangle) = h \qquad F_1'(\langle q_0, l\rangle) = l .$$

If we construct T_2' for the transducer T_2 to the right of Example 3, we get an isomorphic transducer. Both transducers are proper and realize the transformation τ^{T_1}. $\square$

5 Earliest Transducers

Assume that we are given a proper btt T. We now want this transducer to produce the output at essential states as *early* as possible. Thereto, we compute the *greatest common suffix* of all non-ground images of contexts for a state q and produce it at q directly.

For an essential state q, let $\mathcal{D}(q)$ denote the set of images $z \in \hat{\mathcal{T}}_\Delta(y)$ produced for contexts of q. Thus, every tree in $\mathcal{D}(q)$ contains an occurrence of the variable y. The *greatest common suffix* of all trees in $\mathcal{D}(q)$ is denoted by $\mathsf{gcs}(q)$, i.e.,

$$\mathsf{gcs}(q) = \bigsqcup \mathcal{D}(q)$$

with respect to the order $\sqsubseteq$ on $\hat{\mathcal{T}}_\Delta(y)$ from Section 2.

Proposition 6. *For a proper btt T, the trees $\mathsf{gcs}(q)$ for all essential states q of T can be computed in polynomial time.*

Proof. Assume that $z \in \mathcal{T}_\Delta(\mathcal{X}_k)$ and that x_i occurs in z. Then $\mathsf{suff}_i(z)$ denotes the largest subtree z_i of $z[y/x_i]$ with the following properties:

- y is the only variable occurring in z_i, i.e., $z_i \in \hat{\mathcal{T}}_\Delta(y)$;
- $z[y/x_i] = z' \cdot z_i$ for some z', i.e., $z' \in \hat{\mathcal{T}}_{\Delta \cup \mathcal{X}_k \setminus \{x_i\}}(y)$.

Then the trees $\mathsf{gcs}(q)$ are the least solution of the inequations

$$\begin{array}{lll} \mathsf{gcs}(q_i) \sqsupseteq \mathsf{suff}_i(\mathsf{gcs}(q) \cdot z), & & a(q_1, \ldots, q_k) \to q(z) \in R \text{ and } x_i \text{ occurs in } z, \\ \mathsf{gcs}(q) \ \sqsupseteq z, & & F(q) = z \text{ and } y \text{ occurs in } z. \end{array}$$

Since T is proper, this system contains inequations only for essential states q. Since the right-hand sides are monotonic, the system has a unique least solution. Since the complete lattice $\hat{\mathcal{T}}_\Delta(y)_\perp$ satisfies the ascending chain condition, this least solution can effectively be computed. Using a standard worklist algorithm, it can be shown that each inequation is evaluated at most $\mathcal{O}(|T|)$ times. If we represent elements from $\hat{\mathcal{T}}_\Delta(y)$ as sequences of *irreducible* trees, then each right-hand side also can be evaluated in polynomial time. This proves the complexity bound stated in the proposition. $\square$

A proper bottom-up tree transducer T is called *earliest* if the greatest common suffix of every essential state q equals y.

Theorem 7. *For each proper tree transducer T, a tree transducer T' can be constructed in polynomial time with the following properties:*

- *T' is equivalent to T;*
- *T' is earliest.*

Proof (Sketch). Let T be a proper transducer. According to Proposition 6, we can compute the greatest common suffix $\mathsf{gcs}(q)$ for every essential state q of T. The corresponding earliest transducer T' has the same set of states as T as well as the same input and output alphabets, but only differs in the transition function and the final function.

For a right-hand side $q(z)$ of a transition in T, we construct the output of the corresponding transition in T' in two steps. First, we add to z the greatest common suffix corresponding to q, i.e., we define $\bar{z} = \mathsf{gcs}(q) \cdot z$. Then we remove from $\bar{z}$ the greatest common suffices of all states corresponding of all variables occurring in $\bar{z}$ (and z). $\square$

Example 8. Assume that $\Sigma = \{A, B, C, E\}$ and $\Delta = \{d, e\}$. Consider the proper btt $T = (Q, \Sigma, \Delta, R, F)$ with set of (essential) states $Q = \{q_1, q_2\}$ where the final function is $F = \{q_1 \mapsto d(d(y, e), d(y, e))\}$ and the transition function R is given by:

$$\begin{array}{rlrl} A(q_1, q_2) & \to q_1(d(x_2, d(x_1, e))) & E & \to q_1(e) \\ B(q_2) & \to q_2(d(x_1, d(d(e, e), e))) & C & \to q_2(e) \end{array}$$

To compute the greatest common suffices, we consider the following inequations:

$$\begin{array}{lll} \mathsf{gcs}(q_1) \sqsupseteq \mathsf{suff}_1(\mathsf{gcs}(q_1) \cdot d(x_2, d(x_1, e))) & = d(y, e) \\ \mathsf{gcs}(q_2) \sqsupseteq \mathsf{suff}_2(\mathsf{gcs}(q_1) \cdot d(x_2, d(x_1, e))) & = y \\ \mathsf{gcs}(q_2) \sqsupseteq \mathsf{suff}_1(\mathsf{gcs}(q_2) \cdot d(x_1, d(d(e, e), e))) & = \mathsf{gcs}(q_2) \cdot d(y, d(d(e, e), e)) \\ \mathsf{gcs}(q_1) \sqsupseteq F(q_1) & = d(d(y, e), d(y, e)) \end{array}$$

For q_2, we obtain $\mathsf{gcs}(q_2) = y$. Moreover since $d(y, e) \sqcup d(d(y, e), d(y, e)) = d(y, e)$, we have $\mathsf{gcs}(q_1) = d(y, e)$. The final function of the earliest btt for T' thus is given by $F' = \{q_1 \mapsto d(y, y)\}$. In order to construct the new transition function, first consider the right-hand side for $A(q_1, q_2)$ in R' where $R(A(q_1, q_2)) = q_1(d(x_2, d(x_1, e)))$. In the first step, we construct

$$\bar{z} = \mathsf{gcs}(q_1) \cdot d(x_2, d(x_1, e)) = d(y, e) \cdot d(x_2, d(x_1, e)) = d(d(x_2, d(x_1, e)), e) .$$

From this tree, we remove the suffices for q_1 and q_2 at the variables x_1 and x_2, respectively. This results in the tree $u = d(d(x_2, x_1), e)$. Therefore, we obtain the transition

$$A(q_1, q_2) \to q_1(d(d(x_2, x_1), e)) .$$

Analogously, we obtain the transitions

$$E \to q_1(d(e, e)) \qquad B(q_2) \to q_2(d(x_1, d(d(e, e), e))) \qquad C \to q_2(e) .$$

6 Unified Transducers

For an earliest btt, contexts of states may disagree except for a pair of output trees.

Example 9. Assume that $\Sigma = \{A, \ldots, E, G\}$ and $\Delta = \{b, d, e, f, g, \bot\}$. Consider the earliest btt $T = (Q, \Sigma, \Delta, R, F)$ with $Q = \{q_0, q_1, q_1', q_2, q_2', q_3\}$ and R, F given by:

$$\begin{array}{lll} A \to q_0(b) & B(q_0) \to q_0(e(x_1)) & F(q_1) = f(y, b) \\ C(q_0) \to q_1(x_1) & D(q_0) \to q_1'(x_1) & F(q_1') = f(e(y), y) \\ E(q_0) \to q_2(x_1) & G(q_0) \to q_2'(x_1) & F(q_2) = y \\ C(q_2) \to q_1(e(b)) & C(q_2') \to q_1'(b) & F(q_2') = y \\ D(q_1) \to q_3(d(g, x_1)) & D(q_1') \to q_3(d(g, e(x_1))) & F(q_3) = y \end{array}$$

For each context c of q_2, i.e., $c \in \{C(y), D(C(y))\}$, the two states q_2 and q_2' induce the same image: $\tau_{q_2}^T(c) = \tau_{q_2'}^T(c)$. But unfortunately, the successor states q_1 and q_1' do not have this property. Both states are essential and have the same contexts. The images of the context $D(y)$, $\tau_{q_1}^T(D(y)) = d(g, y)$ and $\tau_{q_1'}^T(D(y)) = d(g, e(y))$, differ only in the suffix $e(y)$. The images of the context y are $\tau_{q_1}^T(y) = f(y, b)$ and $\tau_{q_1'}^T(y) = f(e(y), y)$. If y is substituted by b in the image at q_1' and $e(b)$ at q_1, they become equal. Thus, for each context c of q_1, we get $\tau_{q_1}^T(c) \cdot e(b) = \tau_{q_1'}^T(c) \cdot b$. □

Assume that $T = (Q, \Sigma, \Delta, R, F)$ is an earliest btt and that $q_1, q_2 \in Q$ are states. Assume that q_1 and q_2 have the same contexts, i.e., $\mathcal{C}^T(q_1) = \mathcal{C}^T(q_2)$. Then, we define the *most general unifier* of q_1, q_2, $\mathsf{mgu}(q_1, q_2)$, as the most general unifier of the set $C = \{(\tau^T_{q_1}(c), \tau^T_{q_2}(c)) \mid c \in \mathcal{C}^T(q_1)\}$ of pairs of images of contexts of q_1 and q_2, i.e., $\mathsf{mgu}(q_1, q_2) = \mathsf{mgu}(C)$. Otherwise, we set $\mathsf{mgu}(q_1, q_2) = \bot$. We call q_1, q_2 *unifiable* if the most general unifier is not $\bot$.

The most general unifier $\mathsf{mgu}(q_1, q_2) = \langle s_1, s_2 \rangle$ for unifiable states q_1, q_2 has the following properties:

– If q_i is inessential, then for every context c of q_i, $\tau^T_{q_i}(c) \in \mathcal{T}_\Delta$. Therefore, $s_i = \top$.
– Moreover, s_1 contains y iff s_2 contains y. If both s_1 and s_2 contain y, the mgu must equal $\langle y, y \rangle$, otherwise T would not be earliest.

A ground term s is called *realizable* in a state q if s is contained in the set of outputs of q. Note that the ground terms s occurring in most general unifiers of states are, however, not necessarily realizable. The earliest btt T is called *unified earliest* if no ground term in most general unifiers of states of T is realizable. In the following, we show that for every earliest btt, a unified earliest btt can be constructed in polynomial time. For this construction, we require the following observation.

Theorem 10. *Assume that T is an earliest bottom-up tree transducer. Then all most general unifiers* $\mathsf{mgu}(q_1, q_2)$ *can be constructed in polynomial time.*

Assume now that we are given the most general unifier of an earliest btt T. Then we can construct a unified earliest transducer T' which is equivalent to T. We have:

Theorem 11. *For each earliest btt T, a btt T' can be constructed in polynomial time with the following properties:*

– *T' is equivalent to T;*
– *T' is unified earliest.*

Proof (Sketch). Let T be an earliest btt. We construct the unified earliest btt T'.

First, we introduce new states. Whenever an output s of an input t at state q is produced by T which will contribute to a ground unifier of q, then the computation on t is redirected to a new state $\langle q, s \rangle$ which memorizes s and does produce $*$ only. Instead, the output s is delayed to the images of the contexts. This implies that the new state $\langle q, s \rangle$ is inessential. Furthermore, for states q' used to evaluate subtrees of t whose outputs s' may contribute to s, further states $\langle q', s' \rangle$ should be introduced.

Some of the new states $\langle q, s \rangle$ now may be unreachable. The transducer T' therefore is defined as the trim transducer, obtained according to Proposition 2. □

Example 12. Consider again the transducer $T = (Q, \Sigma, \Delta, R, F)$ of Example 9. The most general unifiers are

$$\mathsf{mgu}(q_1, q_1') = \langle e(b), b \rangle\,, \qquad \mathsf{mgu}(q_2, q_2') = \langle \top, \top \rangle\,,$$

and $\mathsf{mgu}(q, q') = \bot$, otherwise. We get the set $S = \{e(b), b, \bot\}$ of subterms of terms occurring as ground unifiers of states or $\bot$. All states of $Q \times S$ are possible new states. Except from $\langle q_3, b \rangle$ and $\langle q_3, e(b) \rangle$ all are reachable.

Starting with left-hand side A, we get the new transition $A \to \langle q_0, b\rangle\,(*)$, because $b \in S$. Furthermore, for the transition $B(q_0) \to q_0(e(x_1))$ we get the transition $B(\langle q_0, b\rangle) \to \langle q_0, e(b)\rangle\,(*)$, because $e(x_1)[b/x_1] = e(b) \in S$. Now, consider the left-hand side $B(\langle q_0, e(b)\rangle)$. The potential output $e(x_1)[e(b)/x_1] = e(e(b))$ is not in S. Thus, the right-hand side should be $\langle q_0, \bot\rangle\,(e(e(b)))$. And for the left-hand side $B(\langle q_0, \bot\rangle)$, we get the transition $B(\langle q_0, \bot\rangle) \to \langle q_0, \bot\rangle\,(e(x_1))$. □

7 Minimal Transducers

Last, we merge equivalent states by preserving the properties of an unified earliest btt. Let $\sim'$ denote the smallest equivalence relation with the following properties:

- If $\mathsf{mgu}(q, q') = \langle y, y\rangle$ or $\mathsf{mgu}(q, q') = \langle \top, \top\rangle$ then $q \sim' q'$;
- Assume that $\mathsf{mgu}(q, q_1) = \langle \top, s_1\rangle$ for some ground term s_1. If for all q_2 with $\mathsf{mgu}(q, q_2) = \langle \top, s_2\rangle$ for some $s_2 \neq \top$, $\mathsf{mgu}(q_1, q_2) = \langle y, y\rangle$ holds then $q \sim' q_1$.

The relation $\sim$ is the greatest equivalence relation which is a refinement of $\sim'$ such that, $q_1 \sim q_2$ whenever for every symbol $a \in \Sigma$ of rank k, every $1 \leq i \leq k$, and all states $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k \in Q$, the following holds. There is a transition $a(p_1, \ldots, p_{i-1}, q_1, p_{i+1}, \ldots, p_k) \to q_1'(z_1)$ in R iff there is a transition in R of the form $a(p_1, \ldots, p_{i-1}, q_2, p_{i+1}, \ldots, p_k) \to q_2'(z_2)$. If such two transitions exist then $q_1' \sim q_2'$.

A unified earliest transducer $T = (Q, \Sigma, \Delta, R, F)$ is said to be *minimal* iff all distinct states $q_1, q_2 \in Q$ are not equivalent, i.e., $q_1 \not\sim q_2$.

Theorem 13. *For each unified earliest btt T a unified earliest btt T' can be constructed in polynomial time with the following properties:*

- *T' is equivalent to T;*
- *T' is minimal;*
- $|T'| \leq |T|$.

Proof. Let $T = (Q, \Sigma, \Delta, R, F)$ be an earliest btt. By fixpoint iteration, we compute the equivalence relation $\sim$ on Q. Now, we build a transducer T' with the equivalence classes of $\sim$ as states. Let $[q] = \{q' \mid q \sim q'\}$ the equivalence class of q. We call $[q]$ inessential, if all states in $[q]$ are inessential. Otherwise, it is called essential. For each class $[q]$ we mark a representative state $p_q \in Q$, which is essential iff $[q]$ is essential.

Formally, we get $T' = (Q', \Sigma, \Delta, R', F')$ with $Q' = \{[q] \mid q \in Q\}$. The function F' is given by $F'([q]) = F(p_q)$. And for R', assume that $q_1, \ldots, q_k \in Q$ are representatives of their equivalence classes and that $a(q_1, \ldots, q_k) \to q(z) \in R$.

- If q is essential, then $a([q_1], \ldots, [q_k]) \to [q](z) \in R'$.
- If p_q is inessential, then $a([q_1], \ldots, [q_k]) \to [q](z) \in R'$.
- Otherwise, if $\mathsf{mgu}(q, p_q) = \langle \top, s\rangle$, then $a([q_1], \ldots, [q_k]) \to [q](s) \in R'$.

By induction on the depth of input trees t and length of contexts c, we obtain:

- $\forall t \in \mathcal{T}_\Sigma$: $[\![t]\!]^{T'} = [q](z)$ iff $\exists q' \in [q]$ with $[\![t]\!]^{T} = \begin{cases} q'(z) & \mathsf{mgu}(q', p_q) = \langle y, y\rangle \\ q'(z) & \mathsf{mgu}(q', p_q) = \langle \top, \top\rangle \\ q'(*) & \mathsf{mgu}(q', p_q) = \langle \top, z\rangle \end{cases}$

– $\forall c \in \hat{\mathcal{T}}_\Sigma(y) : \tau^{T'}_{[q]}(c) = z$ iff $\tau^{T}_{p_q}(c) = z$.

It follows that $\tau^T = \tau^{T'}$ and that T' is trim, proper, earliest, and unified. □

In the following, we show that the equivalent minimal btt for a given earliest unified btt is unique. Let $T_1 = (Q_1, \Sigma, \Delta, R_1, F_1)$ and $T_2 = (Q_2, \Sigma, \Delta, R_2, F_2)$ be two equivalent minimal btt's, i.e. $\tau^{T_1} = \tau^{T_2}$. We abbreviate the output of a tree $t \in \mathcal{T}_\Sigma$ in a transducer T_i as $out^{T_i}(t)$, i.e., it exists a state q in Q_i with $[\![t]\!]^{T_i} = q(out^{T_i}(t))$.

For each state $q \in Q_1$ a state $r_q \in Q_2$ is said to be *related* to q, if both are reached by at least one same input tree, i.e., $\exists t \in \mathcal{L}^{T_1}(q) \cap \mathcal{L}^{T_2}(r_q)$. If the transducers are trim, for each state $q \in Q_1$ exists at least one related state r_q: If q is reachable, there exists $t \in \mathcal{L}^{T_1}(q)$. Since q is meaningful, there should also exist a state $r_q \in Q_2$ with $t \in \mathcal{L}^{T_2}(r_q)$. We will show that there exists exactly one related state for each $q \in Q_1$. That will give us a mapping from T_1 to T_2.

Lemma 14. *Assume T_1 and T_2 are two minimal transducers which are equivalent. Then for each state q of T_1 there exists exactly one related state r_q in T_2. And the following holds:*

– *Every context c of q is a context of r_q and $\tau^{T_1}_q(c) = \tau^{T_2}_{r_q}(c)$;*
– *$\mathcal{L}^{T_1}(q) = \mathcal{L}^{T_2}(r_q)$ and for each input tree t holds $[\![t]\!]^{T_1} = q(z)$ iff $[\![t]\!]^{T_2} = r_q(z)$.*

Theorem 15. *The minimal transducer T for a transformation τ is unique.*

Proof (Sketch). Assume T_1 and T_2 are minimal transducers with $\tau^{T_1} = \tau^{T_2}$, and define a mapping $\varphi : Q_1 \to Q_2$ by $\varphi(q) = r_q$ where r_q is the related state of q. By the previous lemma, this mapping is well-defined and bijective. It remains to show that φ is an isomorphism w.r.t. the transition and final functions, i.e.,

1. $F_1(q)$ is defined iff $F_2(\varphi(q))$ is defined, and if they are defined, $F_1(q) = F_2(\varphi(q))$,
2. $a(q_1, \ldots, q_k) \to q_0(z_0) \in R_1 \Leftrightarrow a(\varphi(q_1), \ldots, \varphi(q_k)) \to \varphi(q_0)(z_0) \in R_2$. □

Summarizing, we obtain from Propositions 2, 4 and Theorems 7, 11, 13 and 15:

Theorem 16. *For each btt T an equivalent minimal transducer can be constructed which is unique up to renaming of states. If the btt T is already proper, the construction can be performed in polynomial time.* □

8 Conclusion

We have provided a normal form for deterministic bottom-up tree transducers which is unique up to renaming of states. In case that the btt is already proper, i.e., does only produce output at essential states, the construction can be performed in polynomial time — given that we represent right-hand sides compactly. Though similar in spirit as the corresponding construction for top-down deterministic transducers, the given construction for btt's is amazingly involved and relies on a long sequence of transformations of the original transducer to rule out anomalies in the behavior of the transducer.

It remains to future work to evaluate in how far our novel normal-form can be applied, e.g., in the context of learning tree-to-tree transformations.

References

1. Aho, A.V., Ullman, J.D.: Translations on a context-free grammar. Inform. and Control 19, 439–475 (1971)
2. Berstel, J.: Transductions and Context-Free Languages. Teubner, Stuttgart (1979)
3. Courcelle, B., Franchi-Zannettacci, P.: On the equivalence problem for attribute systems. Inform. and Control 52, 275–305 (1982)
4. Engelfriet, J.: Bottom-up and top-down tree transformations — a comparison. Math. Systems Theory 9, 198–231 (1975)
5. Engelfriet, J.: Top-down tree transducers with regular look-ahead. Math. Systems Theory 10, 289–303 (1977)
6. Engelfriet, J.: Some open questions and recent results on tree transducers and tree languages. In: Book, R.V. (ed.) Formal language theory; perspectives and open problems. Academic Press, New York (1980)
7. Engelfriet, J., Maneth, S.: Macro tree transducers, attribute grammars, and MSO definable tree translations. Inform. and Comput. 154, 34–91 (1999)
8. Engelfriet, J., Maneth, S.: Macro tree translations of linear size increase are MSO definable. SIAM J. Comput. 32, 950–1006 (2003)
9. Engelfriet, J., Maneth, S.: The equivalence problem for deterministic MSO tree transducers is decidable. Inform. Proc. Letters 100, 206–212 (2006)
10. Engelfriet, J., Maneth, S., Seidl, H.: Deciding equivalence of top-down XML transformations in polynomial time. J. Comput. Syst. Sci. 75(5), 271–286 (2009)
11. Ésik, Z.: Decidability results concerning tree transducers I. Acta Cybernetica 5, 1–20 (1980)
12. Filiot, E., Raskin, J.-F., Reynier, P.-A., Servais, F., Talbot, J.-M.: Properties of visibly pushdown transducers (submitted 2010)
13. Fülöp, Z.: On attributed tree transducers. Acta Cybernetica 5, 261–279 (1981)
14. Fülöp, Z., Vágvölgyi, S.: Attributed tree transducers cannot induce all deterministic bottom-up tree transformations. Inform. and Comput. 116, 231–240 (1995)
15. Fülöp, Z., Vogler, H.: Syntax-Directed Semantics – Formal Models based on Tree Transducers. In: Brauer, W., Rozenberg, G., Salomaa, A. (eds.) EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg (1998)
16. Gazdag, Z.: Decidability of the shape preserving property of bottom-up tree transducers. Int. J. Found. Comput. Sci. 17(2), 395–414 (2006)
17. Knuth, D.E.: Semantics of context-free languages. Math. Systems Theory 2, 127–145 (1968); Corrections in Math. Systems Theory 5, 95–96 (1971)
18. Laurence, G., Lemay, A., Niehren, J., Staworko, S., Tommasi, M.: Linear top-down tree-to-word transducers: Characterization and minimization. In: RTA (to appear 2010)
19. Lemay, A., Maneth, S., Niehren, J.: A learning algorithm for top-down XML transformations. In: PODS (to appear 2010)
20. Mohri, M.: Minimization algorithms for sequential transducers. Theor. Comput. Sci. 234, 177–201 (2000)
21. Rounds, W.C.: Mappings and grammars on trees. Math. Systems Theory 4, 257–287 (1970)
22. Seidl, H.: Single-valuedness of tree transducers is decidable in polynomial time. Theor. Comput. Sci. 106(1), 135–181 (1992)
23. Thatcher, J.W.: Generalized^2 sequential machine maps. J. Comp. Syst. Sci. 4, 339–367 (1970)
24. Thatcher, J.W.: Tree automata: an informal survey. In: Aho, A.V. (ed.) Currents in the Theory of Computing, pp. 143–172. Prentice Hall, Englewood Cliffs (1973)
25. Zachar, Z.: The solvability of the equivalence problem for deterministic frontier-to-root tree transducers. Acta Cybernetica 4, 167–177 (1980)

Two-Way Unary Automata versus Logarithmic Space

Viliam Geffert[1,⋆] and Giovanni Pighizzini[2,⋆⋆]

[1] Department of Computer Science, P. J. Šafárik University
Jesenná 5, SK-04001 Košice, Slovakia
viliam.geffert@upjs.sk

[2] Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano
via Comelico 39, I-20135 Milano, Italy
pighizzini@dico.unimi.it

Abstract. We show that each n-state unary 2NFA (a two-way nondeterministic finite automaton) can be simulated by an equivalent 2UFA (an unambiguous 2NFA) with a polynomial number of states. Moreover, if L = NL (the classical logarithmic space classes), then each unary 2NFA can be converted into an equivalent 2DFA (a deterministic two-way automaton), still keeping polynomial the number of states. This shows a connection between the standard logarithmic space complexity and the state complexity of two-way unary automata: it indicates that L could be separated from NL by proving a superpolynomial gap, in the number of states, for the conversion from unary 2NFAs to 2DFAs.

Keywords: unary regular languages, finite automata, state complexity, logarithmic space, space complexity.

1 Introduction

Already in 1959, in the period of the early research on finite automata, Rabin and Scott [14] and Shepherdson [19] independently discovered that the capability of moving the input head in both directions does not increase the recognition power of finite memory devices, namely, of two-way deterministic or nondeterministic finite automata (2DFAs or 2NFAs, for short). Both these devices can recognize only regular languages, exactly as their one-way deterministic and nondeterministic counterparts (1DFAs and 1NFAs). Since then, several results comparing the succinctness of the description of regular languages by different kinds of automata have been obtained. (Further references can be found in [2]).

However, despite all this effort, the question posed by Sakoda and Sipser in 1978 [17] about the existence of a polynomial simulation of 2NFAs by 2DFAs

⋆ Supported by the Slovak Grant Agency for Science (VEGA) under contract 1/0035/09 "Combinatorial Structures and Complexity of Algorithms".

⋆⋆ Partially supported by the Italian MIUR under project "Aspetti matematici e applicazioni emergenti degli automi e dei linguaggi formali: metodi probabilistici e combinatori in ambito di linguaggi formali".

is still open.[1] Several authors attacked this problem, obtaining some results for restricted models (e.g, sweeping automata [13,20], oblivious automata [6], unary automata [4], deterministic moles [8]). However, a solution to the general problem seems to be very far.

In this paper we consider two-way *unary* automata, i.e., two-way automata with a single-letter input alphabet. We first show how to reduce the membership problem for a unary language, represented by a 2NFA, to the *graph accessibility problem* (GAP), which is the problem of deciding whether a directed graph contains a path connecting two designated vertices. GAP is known to be a complete problem for logarithmic space [18]. That is, GAP belongs to NL, the class of languages accepted by $O(\log n)$ space bounded nondeterministic Turing machines, and its membership in L, the analogous class for deterministic machines, would imply L = NL, solving a longstanding open problem in space complexity. Thus, our reduction provides a bridge between the state complexity of two-way unary automata and the classical space complexity. This allows us to derive the main contributions.

One of our contributions concerns two-way *unambiguous* automata (2UFAs). We recall that a nondeterministic device is called unambiguous if it presents at most one accepting computation on each input. Comparisons between deterministic, unambiguous, and nondeterministic computational models have been widely considered in the literature. ([11,15,16] are just three examples.) Unambiguous representation of languages is not only of theoretical interest, but plays also an important role in applications, e.g., in parser construction. In the context of nonuniform space complexity, Reinhardt and Allender [16] proved that nondeterministic computations in logarithmic space can be made unambiguous. Combining this with our reduction from 2NFA languages to GAP, we obtain a polynomial simulation of unary 2NFAs by 2UFAs. In other words, each unary 2NFA can be made unambiguous with a polynomial increasing in the number of the states.

The other contribution presented in our paper relates the above mentioned Sakoda-Sipser's question to the $L \overset{?}{=} NL$ problem. To the best of authors' knowledge, the only connection at the moment seems to be the one stated by Berman and Lingas in 1977 [1]:

> If L = NL, then there exists a polynomial p such that, for each integer m and each k-state 2NFA A, there exists a $p(mk)$-state 2DFA A' accepting a language $L(A') \subseteq L(A)$ such that $L(A')$ and $L(A)$ agree on all strings of length smaller than or equal to m, i.e., $L(A') \cap \Sigma^{\leq m} = L(A) \cap \Sigma^{\leq m}$.

In the invited paper of Kapoutsis at DLT'09 [9, p.51], the same result is presented by replacing "k-state 2NFA A" with an automaton recognizing the language C_k, presented in the paper of Sakoda and Sipser [17]. This language (actually, this sequence of languages) is "complete" for the reduction of 2NFAs to 2DFAs. (Kapoutsis calls this language TWO-WAY-LIVENESS$_k$):

[1] Unless otherwise stated, "polynomial simulation" means polynomial in the number of the states throughout the paper.

> If L = NL, then there exists a polynomial p such that, for each integer k, there exists a $p(k)$-state 2DFA able to decide the TWO-WAY-LIVENESS$_k$ correctly on every $p(k)$-long input.

In the paper of Sipser on sweeping automata [20], a similar condition is presented for ONE-WAY-LIVENESS$_k$, which is a complete problem for the reduction from 1NFAs to 2DFAs, and sweeping automata:

> If L = NL, then there exists a polynomial p such that, for each integer k, there exists a $p(k)$-state sweeping automaton accepting the set of strings of length at most k in ONE-WAY-LIVENESS$_k$.

All these conditions can be rewritten as sufficient conditions for L $\neq$ NL. The resulting conditions require the separation of 2DFAs from 2NFAs of polynomial size on "short" inputs (of polynomial length).[2]

In this paper, we shall prove that L = NL would imply a polynomial simulation of unary 2NFAs by 2DFAs. Hence, also this result gives a sufficient condition for L $\neq$ NL. This condition, compared with that of Berman and Lingas [1], restricts the set of candidates to *unary automata*, but completely removes the restriction on the input length. So it seems not to be comparable with the previous conditions, but it seems more natural and understandable.

In [4], it was shown that each n-state unary 2NFA can be simulated by a 2DFA with $O(n^{\lfloor \log(n+1)+3 \rfloor})$ states. Hence, the cost of the simulation of unary 2NFAs by 2DFAs is subexponential. In the light of our new result, an argument showing the tightness of this upper bound or a different (but still superpolynomial) lower bound would imply L $\neq$ NL. At the moment, the best known lower bound is quadratic [3].

2 Preliminaries

Let us start by briefly recalling some basic definitions from automata theory. For a detailed exposition, we refer the reader to [5]. 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 $A = (Q, \Sigma, \delta, q_{\rm I}, F)$, in which Q is a finite set of states, Σ 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, $q_{\rm I} \in Q$ 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, A reads an input symbol, changes its state, and moves the input head one position forward, backward, or keeps it stationary depending on whether δ returns $+1$, -1, or 0, respectively. The machine accepts the input, if there exists a computation path from the initial state $q_{\rm I}$ with the head on the left endmarker to some final state $q \in F$.

[2] Kapoutsis [8,9] observes that even a separation with exponentially long strings appears to be hard.

The language accepted by A, denoted by $L(A)$, consists of all input strings that are accepted.

The 2NFA A is said to be *deterministic* (2DFA), whenever $|\delta(q, \sigma)| \leq 1$, for any $q \in Q$ and $\sigma \in \Sigma \cup \{\vdash, \dashv\}$. As an intermediate model between 2NFA and 2DFA, the 2NFA A is called *unambiguous* (2UFA), if there exists at most one accepting computation path for each input. It is said to be *sweeping* [20], if it is deterministic and input head reversals are allowed only at the endmarkers. This last notion can be relaxed by allowing nondeterministic choices at the endmarkers, but keeping the other restrictions. In this case the 2NFA is said to be *quasi sweeping* [12].

An automaton A is *almost equivalent* to an automaton A', if the languages accepted by A and A' coincide, with the possible exception of a finite number of strings. If there are no exceptions, then A and A' are *(fully) equivalent.*

An automaton working over a single letter alphabet is called *unary.* For such automata, we shall fix the input alphabet to $\Sigma = \{a\}$.

We assume that the reader is familiar with the standard Turing machine model and the basic facts from space complexity theory. Our Turing machine (both deterministic and nondeterministic) is equipped with a finite state control, a two-way read-only input tape, with input enclosed in between two endmarkers, and a separate semi-infinite two-way read-write worktape, initially empty. Such machine is $s(n)$*-space bounded*, if no computation uses more than $s(n)$ worktape cells, for each input of length n.

The class of languages accepted in $s(n) \leq O(\log n)$ space by deterministic Turing machines is denoted by L, while the corresponding classes for nondeterministic and unambiguous machines by NL and UL.

A *deterministic transducer* is an ordinary $s(n)$-space bounded Turing machine, equipped with an additional one-way write-only output tape, initially empty.

We also assume that the reader is familiar with basic notions from *graph theory*, in particular, with the notion of *labeled* and *directed graph* and the notion of a *path* in a directed graph.

The *graph accessibility problem* is the problem of deciding, for a given directed graph $G = (V, E)$, with $V = \{v_1, \ldots, v_N\}$, whether there exists a path from v_1 to v_N. We assume that the graph is presented on the input tape of a Turing machine in the form of a binary adjacency matrix, written row by row, in which $g_{i,j}$ (the element at row i and column j) is equal to 0 or 1 depending on whether $(v_i, v_j) \in E$. Clearly, the size of this matrix is $N \times N$, its length is N^2, and $g_{i,j}$ is written at the bit position $(i-1) \cdot N + j$. As customary for decision problems, the set of all directed graphs coded in this way, for which the graph accessibility problem has a positive answer, will be denoted by GAP.

It is an open problem whether L = NL. In [18], it was shown that GAP is an NL-complete problem. Hence, GAP $\in$ NL and, moreover, GAP $\in$ L if and only if L = NL.

3 Reducing Unary 2NFA Acceptance to GAP

In this section we show how to reduce the membership problem for a language accepted by a given unary 2NFA to the graph accessibility problem in a given directed graph. To this aim, we shall use the following result, which allows us to consider 2NFAs in a special form:

Theorem 1 ([4, Thm. 2]). *Each n-state unary 2NFA A can be transformed into a 2NFA M such that:*

- *M has at most $N \leq 2n+2$ states.*
- *M is quasi-sweeping (i.e., input head reversals and nondeterminism are performed only at the endmarkers).*
- *M and A are almost equivalent, in particular, the accepted languages coincide for all inputs of length greater than $5n^2$.*

By Theorem 1, an accepting computation of M is a sequence of deterministic *traversals* of the input, ending in a final state. Nondeterministic decisions are taken only when the input head is visiting one of the endmarkers. Actually, the statement of Theorem 1 does not show all the features of the automaton M. We now present further details about its structure. These details, shown in [4], will turn out to be useful to state our results.

Let M be denoted as a quintuple $(Q, \{a\}, \delta, q_{\text{I}}, \{q_{\text{F}}\})$. (As explained below, M has a unique final state.) According to Theorem 1, n denotes the number of the states in the original automaton A from which M is derived. Thus, $|Q| = N \leq 2n+2$.

The set of states of M is the union of $s+1$ disjoint nonempty sets $\{q_{\text{I}}, q_{\text{F}}\}$, $Q_1, \ldots, Q_s$, for some s, such that:

- The set $\{q_{\text{I}}, q_{\text{F}}\}$ consists of the initial and final states only, with $q_{\text{I}} \neq q_{\text{F}}$.
- Each of the sets $Q_1, \ldots, Q_s$ represents either a "positive" or a "negative" loop, used to traverse the input tape from the left endmarker (right, respectively) to the opposite endmarker. Essentially, during each traversal of the input, one of these sets is used to count the input length modulo one integer. More precisely, for each $i = 1, \ldots, s$, there exist some $\ell_i > 0$ and $d_i \in \{-1, +1\}$ such that:
  - $Q_i = \{q_{i,0}, q_{i,1}, \ldots, q_{i,\ell_i - 1}\}$,
  - $\delta(q_{i,j}, a) = \{(q_{i,(j+1) \bmod \ell_i}, d_i)\}$, for $j = 0, \ldots, \ell_i - 1$.

  Note that the transitions from the states in Q_i are deterministic in the middle of the input. However, at the endmarkers, there may exist other transitions from these states. If $d_i = +1$ ($d_i = -1$, respectively), the set Q_i is called a *positive* loop of length ℓ_i (*negative*, respectively) and its states are called *positive* (*negative*).
- Let Q^+ and Q^-, respectively, denote the sets of positive and negative states, i.e., $Q^+ = \bigcup_{d_i=+1} Q_i$ and $Q^- = \bigcup_{d_i=-1} Q_i$. In [4], it was shown that $|Q^+| \leq n$ and $|Q^-| \leq n$.

Now we can give further details concerning the transition function of M and the structure of accepting computations:

– The initial state q_{I} is used only at the beginning of the computation. From this state, scanning the left endmarker, the machine switches to some positive state, to start its first left-to-right traversal. Hence, $\delta(q_{\mathrm{I}}, \sigma) \ni (q, d)$ implies that $\sigma = \vdash$, $q \in Q^+$, and $d = +1$.
– As already described, in a left-to-right traversal, only states from some positive loop $Q_i \subseteq Q^+$ are used, until the machine reaches the right endmarker. At this endmarker, by a nondeterministic transition, the machine chooses a negative state and moves the head one position to the left, to start a right-to-left traversal. Hence, for $q \in Q^+$, $\delta(q, \sigma) \ni (p, d)$ implies that either $\sigma = a$, with $p \in Q^+$ and $d = +1$ (deterministic transition in the course of a left-to-right traversal), or $\sigma = \dashv$, with $p \in Q^-$ and $d = -1$ (switching from a left-to-right traversal to some right-to-left traversal at the right endmarker).
– In a right-to-left traversal, one negative loop is used in a similar way. The only difference is at the left endmarker, where the machine can also switch to the final state q_{F} without moving the head, in order to accept the input. Hence, for $q \in Q^-$, $\delta(q, \sigma) \ni (p, d)$ implies one of the following three cases: $\sigma = a$, with $p \in Q^-$ and $d = -1$ (deterministic transition in the course of a right-to-left traversal), or $\sigma = \vdash$, with $p \in Q^+$ and $d = +1$ (switching from a right-to-left traversal to some left-to-right traversal at the left endmarker), or $\sigma = \vdash$, with $p = q_{\mathrm{F}}$ and $d = 0$ (acceptance at the left endmarker).
– For completeness, no moves are defined from the final state q_{F}, in which the machine halts and accepts. Note also that q_{F} can only be reached at the left endmarker.

By the previous description, one can observe that the behavior of M while scanning "real" input symbols is very restricted. In order to formalize this point and, at the same time, to simplify the discussion, it is useful to introduce a further notation.

Let p, q be two states of a unary two-way automaton M in the "normal form" introduced by Theorem 1. Then, for an integer $m \geq 0$, we write $p \overset{m}{\rhd} q$ ($q \overset{m}{\lhd} p$, respectively) if and only if M, starting from the state p with the input head scanning the left (right, respectively) endmarker, can reach the state q with the head scanning first symbol to the right (to the left) of a^m, by a sequence of moves that keep the head inside the tape segment $\vdash a^m$ (inside $a^m \dashv$) in the meantime.

Lemma 2. *Given $p, q \in Q$ and an integer $m \geq 0$, the following statements hold:*

– *$p \overset{m}{\rhd} q$ if and only if $p \in Q^- \cup \{q_{\mathrm{I}}\}$, $q \in Q_i$ for some positive loop $Q_i \subseteq Q^+$ of length ℓ_i, and $p \overset{r}{\rhd} q$, where $r = m \bmod \ell_i$.*
– *$q \overset{m}{\lhd} p$ if and only if $p \in Q^+$, $q \in Q_i$ for some negative loop $Q_i \subseteq Q^-$ of length ℓ_i, and $q \overset{r}{\lhd} p$, where $r = m \bmod \ell_i$.*

We can now introduce a directed labeled graph $G = (Q, E, \Psi)$ which provides a different description of M and of its computations. This will play a fundamental role in proving the main results.

– The vertex set of G coincides with the set Q of the states in M, the set of edges is $E = Q \times Q$.
– The edge (p, q) is labeled by a pair $\Psi_{p,q} = (\ell, R)$, for some integer $\ell \geq 1$ and some $R \subseteq \{0, \ldots, \ell-1\}$.

We first describe the labeling function in an informal way. For our purposes, the "significant" part of E is restricted to $(\{q_{\rm I}\} \times Q^+) \cup (Q^+ \times Q^-) \cup (Q^- \times Q^+) \cup (Q^- \times \{q_{\rm F}\})$. Given $p, q \in Q$, we want to use the label $\Psi_{p,q}$ to characterize the set of all integers m such that M, scanning the left/right endmarker of the input tape $\vdash a^m \dashv$ in the state p, can reach the state q with the head scanning one of the endmarkers again, by a sequence of moves that do not visit any of the endmarkers in the meantime. Typically, p, q are placed at the opposite endmarkers and hence $\Psi_{p,q}$ characterizes the set of all integers m such that $p \overset{m}{\rhd} q$ or $q \overset{m}{\lhd} p$, depending on whether $p \in Q^- \cup \{q_{\rm I}\}$ or $p \in Q^+$, respectively, i.e., on whether we characterize a left-to-right or right-to-left traversal. By Lemma 2, such integers m are exactly those satisfying a condition of the form $m \bmod \ell = r$, where ℓ depends only on q while r depends on both p and q. Namely, if $q \in Q_i$, then $\ell = \ell_i$. However, transitions from p at one of the endmarkers may be nondeterministic, possibly choosing several different states within the same loop Q_i. Hence, there may exist several different values of r for the same pair p, q. For these reasons, $\Psi_{p,q}$ keeps a single value ℓ but uses a set of possible values $r \in R$.

As a special case, for $\delta(p, \vdash) \ni (q_{\rm F}, 0)$ and $q = q_{\rm F}$, the label $\Psi_{p,q}$ does not depend on the length of the input and should characterize the set of all integers m. This can be accomplished by taking $\Psi_{p,q} = (1, \{0\})$, since $m \bmod 1 = 0$ for each $m \geq 0$. Similarly, if M can never get from p to q by a computation not visiting any of the endmarkers in the meantime (as an example, if both p and q belong to Q^+), we can take $\Psi_{p,q} = (1, \emptyset)$, since $m \bmod 1 \notin \emptyset$ for any m.

Formally, Ψ is defined as follows:

$$\Psi_{p,q} = \begin{cases} (\ell_i, \{r \in \{0, \ldots, \ell_i-1\} \mid p \overset{r}{\rhd} q\}), & \text{if } p \in Q^- \cup \{q_{\rm I}\} \text{ and } q \in Q_i \subseteq Q^+, \\ (\ell_i, \{r \in \{0, \ldots, \ell_i-1\} \mid q \overset{r}{\lhd} p\}), & \text{if } p \in Q^+ \text{ and } q \in Q_i \subseteq Q^-, \\ (1, \{0\}), & \text{if } p \in Q^-,\ q = q_{\rm F}, \text{ and } \delta(p, \vdash) \ni (q_{\rm F}, 0), \\ (1, \emptyset), & \text{otherwise.} \end{cases}$$

Notice that we can also obtain $\Psi_{p,q} = (\ell_i, \emptyset)$ if, from the state p at one of the endmarkers, it is not possible to switch to any state in the loop Q_i containing q.

The main property of the graph G is given in the following lemma, which is an immediate consequence of Lemma 2, combined with the definition of Ψ:

Lemma 3. *Given $p, q \in Q$ and an integer $m \geq 0$, the automaton M, scanning one of the endmarkers of the input tape $\vdash a^m \dashv$ in the state p, can reach the state q with the head scanning one of the endmarkers again, by a sequence of moves that do not visit any of the endmarkers in the meantime, if and only if $\Psi_{p,q} = (\ell, R)$ and $m \bmod \ell \in R$.*

As a consequence of Lemma 3, computations of M can be described by paths in the graph G, where only some of the edges are allowed, depending on the input length, and vice versa.

For each given input a^m, we can restrict G to the edges that are valid for a^m. To this aim, we now introduce another graph $G(m) = (Q, E(m))$, depending on the input a^m. The graph $G(m)$ uses the same set of vertices, namely, the state set Q. The edges are not labeled, but filtered as follows:

$$E(m) = \{(p,q) \in Q \times Q \mid \Psi_{p,q} = (\ell, R) \text{ with } m \bmod \ell \in R\}\,.$$

Theorem 4. *For each $m \geq 0$, the input a^m is accepted by M if and only if q_{F} is reachable from q_{I} in the graph $G(m)$.*

Proof. Consider an accepting computation of M on the input tape $\vdash a^m \dashv$ and the sequence of all the states $q_{\mathrm{I}}, p_1, \ldots, p_t, q_{\mathrm{F}}$ reached with the input head visiting one of the endmarkers. With the exception of the final state q_{F}, we can observe that the state p_h is reached at the left endmarker, if the value of h is even, while it is reached at the right endmarker, if h is odd, taking $q_{\mathrm{I}} = p_0$. Since the final state can be reached from p_t only via a stationary move at the left endmarker, t must be even and $p_t \in Q^-$. Summing up:

- The initial state q_{I} coincides with p_0.
- For each $h = 1, \ldots, t$, M gets from the state p_{h-1} to p_h by making a traversal across the input a^m, and hence $p_{h-1} \overset{m}{\rhd} p_h$ or $p_h \overset{m}{\lhd} p_{h-1}$, depending on whether h is odd or even, respectively. In both cases, by Lemma 3, we get $(p_{h-1}, p_h) \in E(m)$.
- $\delta(p_t, \vdash) \ni (q_{\mathrm{F}}, 0)$. Hence, $\Psi_{p_t, q_{\mathrm{F}}} = (1, \{0\})$, which gives $(p_t, q_{\mathrm{F}}) \in E(m)$, for each m.

So, we easily conclude that there is a path from q_{I} to q_{F} in $G(m)$.

Conversely, given a path $q_{\mathrm{I}}, p_1, \ldots, p_t, q_{\mathrm{F}}$ in the graph $G(m)$, using Lemma 3 again, we can observe that, for each $h = 1, \ldots, t$, the edge (p_{h-1}, p_h) represents a left-to-right or a right-to-left traversal of the input, depending on the parity of h. Combining these traversals together, we get an accepting computation of M for the input a^m. $\square$

In Theorem 4, we provided a reduction from $L(M)$ to GAP. This is enough for our purposes. However, for the sake of completeness, we mention how to get a reduction to GAP for the *original* 2NFA A. Recall that M, obtained by the use of Theorem 1, is only almost equivalent to the original A: the languages accepted by A and M may differ in finitely many "exceptions", the length of which is bounded by $5n^2$, where n denotes the number of states in A. Hence, for $m \leq 5n^2$, we can replace the graph $G(m)$, defined above, by a graph consisting just of two vertices $q_{\mathrm{I}}, q_{\mathrm{F}}$. Depending on whether $a^m \in L(A)$, they are (are not) connected by the edge $(q_{\mathrm{I}}, q_{\mathrm{F}})$.

We conclude this section by observing that the existence of each edge in $E(m)$ can be determined using a finite control with at most n states:

Lemma 5. *For each two states p, q of M, there exists a unary deterministic finite automaton $A_{p,q}$ with at most n states, such that for each $m \geq 0$, $A_{p,q}$ accepts a^m if and only if $(p,q) \in E(m)$.*

4 Polynomial Deterministic Simulation (Assuming L=NL)

Under the hypothesis L = NL, we prove here that each unary n-state 2NFA A can be simulated by an equivalent 2DFA with a number of states polynomial in n. Roughly speaking, we shall use a 2DFA which computes the reduction presented in Section 3 and simulates, in its finite control, a deterministic logarithmic space bounded machine accepting GAP. The next lemma describes such a simulation.

Here we use the same notation as in Section 3, in particular, n and N denote, respectively, the number of states in the original 2NFA A and in the almost equivalent M obtained according to Theorem 1, with $N \leq 2n+2$. The set of states in M will be enumerated by $Q = \{v_1, \ldots, v_N\}$, with $q_{\rm I} = v_1$ and $q_{\rm F} = v_N$.

Lemma 6. *If* L = NL*, then there exists a polynomial p such that the 2*NFA *M can be simulated by a 2*DFA *M' with at most $p(N)$ states.*

Proof. If L = NL, there must exist $D_{\rm GAP}$, a deterministic logarithmic space bounded Turing machine accepting GAP. We can assume that $D_{\rm GAP}$ has a two-way read-only input tape which contains a representation of a graph with N vertices, given by its adjacency matrix.

Now, our goal is to devise a 2DFA M' such that, for a given input string a^m, $m \geq 0$, it decides whether or not $a^m \in L(M)$. To this aim, consider the graph $G(m)$. By Theorem 4, $a^m \in L(M)$ if and only if $G(m) \in$ GAP. Thus, by presenting the adjacency matrix of $G(m)$ as an input to the machine $D_{\rm GAP}$, M' can correctly decide whether $a^m \in L(M)$.

Since the length of $G(m)$ coded by the adjacency matrix is N^2, the machine $D_{\rm GAP}$ works in space $O(\log(N^2)) \leq K{\cdot}\log N$, for a suitable constant K. However, N does not depend on the input length m, and hence *this amount of space is fixed.* This allows us to encode the entire worktape of $D_{\rm GAP}$ in the control of a finite state machine.

The critical point of the simulation is that the input tape of M' contains the input string a^m, while the input tape of the simulated machine $D_{\rm GAP}$ must contain the adjacency matrix of $G(m)$, which uses N^2 bits. Keeping this matrix in the finite control of M' would require an exponential number of states.

To overcome this problem, M' does not keep the matrix itself in its finite control, but only the input head position of $D_{\rm GAP}$, which is an integer $h \in \{0, \ldots, N^2{+}1\}$. The corresponding bit in the adjacency matrix of $G(m)$ is computed "on demand", each time the simulation of a single step of $D_{\rm GAP}$ needs it. After that, this bit can be "forgotten", to make room for another bit. Recall that the demanded bit at the input position h corresponds to a fixed entry $g_{i,j}$ in the adjacency matrix for $G(m)$, with i, j satisfying $h = (i{-}1){\cdot}N + j$, which in turns corresponds to a fixed pair of states in M, namely, to v_i, v_j. Hence, the hth input bit is 1 if and only if $(v_i, v_j) \in E(m)$.

Thus, in order to compute the hth input bit and to simulate the next step of $D_{\rm GAP}$, the machine M' uses the corresponding automaton A_{v_i,v_j} as a subroutine (see Lemma 5), where $i = 1 + \lfloor h/N \rfloor$ and $j = 1 + (h \bmod N)$.

Summing up, the 2DFA M' keeps the following information in its final control:

– the input head position of D_{GAP} ($2+N^2$ possible values),
– one bit of $G(m)$, currently scanned by the input head of D_{GAP} (2 possible values),
– the worktape head position ($2+K\cdot\log N$ possible values),
– the worktape content ($2^{K\cdot\log N} \leq N^K$ possible values),
– the finite control of D_{GAP} (some H states, a fixed constant not depending on N),
– the finite control for each automaton A_{v_i,v_j} (at most $N^2\cdot n \leq N^3$ states).

By multiplying all these amounts, we conclude that the total number of states in M' is bounded by $O(N^{K+5}\log N)$. □

Note that $O(N^{K+5}\log N) \leq O(n^{K+5}\log n)$. That is, the deterministic simulation presented in the lemma above is polynomial in the number of states of the original 2NFA A. The degree of this polynomial depends on K, a constant specifying $K\cdot\log N$, the upper bound on the number of cells required by the (hypothetical) deterministic Turing machine D_{GAP} on its *binary* worktape. However, M and M' agree with A only on input strings of length greater than $5n^2$. (See Theorem [1].)

Now we are able to state one of our main results:

Theorem 7. *If* L = NL, *then each n-state unary 2NFA A can be simulated by an equivalent deterministic 2DFA with a polynomial number of states.*

Proof. A 2DFA M'' fully equivalent to the original 2NFA A can now be obtained by a small modification of M' presented in Lemma [6]. First, M'' makes a scan of the input and checks whether its length does exceed $5n^2$. Short inputs are accepted or rejected directly, according to the membership in $L(A)$. For sufficiently long inputs, the membership is resolved by the use of M'. Clearly, the number of the states in M'' is $5n^2+1+O(n^{K+5}\log n) \leq O(n^{K+5}\log n)$. □

5 Polynomial Unambiguous Simulation (Unconditional)

Finally, we are ready to show that each unary n-state 2NFA A can be simulated by an unambiguous 2NFA with a number of states polynomial in n. Compared with Theorem [7], unambiguous 2NFAs are more powerful devices than 2DFAs, but this simulation does not require any additional assumptions, such as L = NL.

Reinhardt and Allender [16] proved that, in the context of nonuniform complexity, nondeterministic logarithmic space bounded computations can be made unambiguous. Our simulation combines this result with the reduction from a unary 2NFA language to GAP.

Given a complexity class $\mathcal{C}$, let us denote [10] by $\mathcal{C}$/poly the class of languages L for which there exist a sequence of binary "advice" strings $\{\alpha(n)\,|\,n\geq 0\}$ of polynomial length and a language $B \in \mathcal{C}$ such that $L = \{x \mid (x, \alpha(|x|)) \in B\}$.

Theorem 8 ([16]). NL $\subseteq$ UL/poly.

As a consequence of this theorem, there exists a nondeterministic Turing machine U_{GAP} such that:

– U_{GAP} works in logarithmic space and has at most one accepting path on each input string,
– there exists a sequence of binary strings $\{\alpha(n) \mid n \geq 0\}$ and a polynomial q, such that $|\alpha(n)| \leq q(n)$ for each $n \geq 0$, and
– for each graph G with N vertices, encoded in the form of the binary adjacency matrix, U_{GAP} accepts the string $G \sharp \alpha(N^2)$ if and only if $G \in$ GAP. Here $\sharp \notin \{0,1\}$ denotes a new separator symbol.

Theorem 9. *Each n-state unary 2*NFA *A can be simulated by an equivalent unambiguous 2*UFA *with a polynomial number of states.*

Proof. (outline) The argument is very similar to that of Lemma 6. For the given N-state 2NFA M obtained by Theorem 1, we construct a 2NFA M' that for input a^m simulates the machine U_{GAP} on input $G(m) \sharp \alpha(N^2)$. M' does not keep the input tape of U_{GAP} in its finite control but only the input head position of U_{GAP}. M' computes the corresponding input symbol "on demand" each time the simulation of the next step of U_{GAP} requires it. However, since the advice $\alpha(N^2)$ is *fixed*, that is, it does not depend on the input of M, only on N, the number of states in M, the information about bits from $\alpha(N^2)$ can be kept directly in the transition function of M'. ☐

6 Final Remarks

In Section 4, we proved that L = NL implies the existence of a polynomial conversion from unary 2NFAs to 2DFAs. It is natural to ask whether the converse implication holds as well. Here we are able to prove a partially different implication. On one hand, even a polynomial conversion from *one-way* unary 1NFAs to 2DFAs (hence, also from unary 2NFAs to 2DFAs) implies L = NL. However, as an additional assumption, such conversion must be *constructive* and, moreover, we must be able to construct the resulting 2DFA by the use of a deterministic transducer working with $O(\log n)$ space.

We recall that the membership problem for unary 1NFAs is known to be NL-complete [7]. As a consequence, we can prove the following:

Theorem 10. *If there exists a deterministic $O(\log n)$ space bounded transducer T transforming each given n-state unary 1*NFA *into an equivalent $n^{O(1)}$-state 2*DFA*, then* L = NL*.*

As shown by Chrobak [3], each n-state unary 1NFA can be transformed into a 2DFA with $O(n^2)$ states. Actually, *provided that the given 1*NFA *is in a special form*, the Chrobak normal form, such transformation is very simple and can be computed by an $O(\log n)$ space bounded transducer. Hence, the "difficult" part in the conversion from unary 1NFAs to 2DFAs is the transformation of a unary 1NFA into the Chrobak normal form. As remarked in [21], such transformation cannot be done in logarithmic space, unless that L = NL.

References

1. Berman, J., Lingas, A.: On the complexity of regular languages in terms of finite automata. Tech. Rep. 304. Polish Academy of Sciences (1977)
2. Birget, J.-C.: State-complexity of finite-state devices, state compressibility and incompressibility. Math. Syst. Theory 26, 237–269 (1993)
3. Chrobak, M.: Finite automata and unary languages. Theoret. Comput. Sci. 47, 149–158 (1986); Corrigendum: IBID 302, 497–498 (2003)
4. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic automata into simpler automata. Theoret. Comput. Sci. 295, 189–203 (2003)
5. Hopcroft, J., Ullman, J.: Introduction to automata theory, languages, and computation. Addison-Wesley, Reading (1979)
6. 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)
7. Jones, N.: Space-bounded reducibility among combinatorial problems. J. Comput. System Sci. 11, 68–85 (1975)
8. Kapoutsis, C.A.: Deterministic moles cannot solve liveness. J. Aut. Lang. Combin. 12, 215–235 (2007)
9. Kapoutsis, C.A.: Size complexity of two-way finite automata. In: DLT 2009. LNCS, vol. 5583, pp. 47–66. Springer, Heidelberg (2009)
10. Karp, R., Lipton, R.: Turing machines that take advice. Enseign. Math. 28, 191–209 (1982)
11. Leung, H.: Descriptional complexity of nfa of different ambiguity. Int. J. Found. Comput. Sci. 16, 975–984 (2005)
12. Mereghetti, C., Pighizzini, G.: Two-way automata simulations and unary languages. J. Aut. Lang. Combin. 5, 287–300 (2000)
13. Micali, S.: Two-way deterministic finite automata are exponentially more succinct than sweeping automata. Inform. Process. Lett. 12, 103–105 (1981)
14. Rabin, M., Scott, D.: Finite automata and their decision problems. IBM J. Res. Develop. 3, 114–125 (1959)
15. Ravikumar, B., Ibarra, O.H.: Relating the type of ambiguity of finite automata to the succinctness of their representation. SIAM J. Comput. 18, 1263–1282 (1989)
16. Reinhardt, K., Allender, E.: Making nondeterminism unambiguous. SIAM J. Comput. 29, 1118–1131 (2000)
17. Sakoda, W., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Proc. 10th ACM Symp. Theory of Comput (STOC), pp. 275–286 (1978)
18. Savitch, M.: Relationships between nondeterministic and deterministic tape complexities. J. Comput. System Sci. 4, 177–192 (1970)
19. Shepherdson, M.: The reduction of two-way automata to one-way automata. IBM J. Res. Develop. 3, 198–200 (1959)
20. Sipser, M.: Lower bounds on the size of sweeping automata. J. Comput. System Sci. 21, 195–202 (1980)
21. To, A.W.: Unary finite automata vs. arithmetic progressions. Information Processing Letters 109, 1010–1014 (2009)

On the Periodicity of Morphic Words

Vesa Halava[1], Tero Harju[1], Tomi Kärki[1,⋆], and Michel Rigo[2]

[1] Department of Mathematics, University of Turku,
FI-20014 Turku, Finland
{vehalava,harju,topeka}@utu.fi

[2] Institute of Mathematics, University of Liège,
Grande Traverse 12 (B 37), B-4000 Liège, Belgium
M.Rigo@ulg.ac.be

Abstract. Given a morphism h prolongable on a and an integer p, we present an algorithm that calculates which letters occur infinitely often in congruent positions modulo p in the infinite word $h^{\omega}(a)$. As a corollary, we show that it is decidable whether a morphic word is ultimately p-periodic. Moreover, using our algorithm we can find the smallest similarity relation such that the morphic word is ultimately relationally p-periodic. The problem of deciding whether an automatic sequence is ultimately weakly R-periodic is also shown to be decidable.

Keywords: automatic sequence, decidability, morphic word, periodicity, similarity relation.

1 Introduction

Periodicity is one of the main concepts in combinatorics on words and an important subject also from the application point of view [11]. The most famous periodicity results in word combinatorics are probably the theorem of Fine and Wilf, the Critical Factorization Theorem, and the theorem of Morse and Hedlund. The theorem of Fine and Wilf considers two simultaneously occurring periods in one word [6], the Critical Factorization Theorem relates the period of a word with the local repetitions [3,5], and the theorem of Morse and Hedlund characterizes ultimately periodic words in terms of subword complexity [9].

It is an interesting and important question how to recognize periodicity and ultimate periodicity of infinite words. This depends heavily on the way the infinite word is generated. For example, in 1986, J. Honkala proved that it is decidable whether a given automatic sequence is ultimately periodic [10]. Such a sequence, produced by a finite automaton with output, can also be generated using a uniform morphism and a coding. Recently, a new method for solving the ultimate periodicity problem for automatic sequences was given by J.-P. Allouche, N. Rampersad and J. Shallit [1]. A more general result showing the decidability of the ultimate periodicity question for pure morphic words was given by T. Harju and M. Linna [8] and, independently, by J.-J. Pansiot [12].

⋆ Supported by Finnish Cultural Foundation, Varsinais-Suomi Regional fund.

However, if we replace pure morphic words of the form $h^{\omega}(a)$ by general morphic words of the form $g(h^{\omega}(a))$ where g is a coding, then the decidability status of the ultimate periodicity problem remains unknown.

In this paper, we solve the decision problem of the ultimate p-periodicity of morphic words. For given integer p and morphism h prolongable on a, i.e., $h(a) = ax$ and $h^n(x) \neq \varepsilon$ for every $n \geq 0$, we present an algorithm that calculates which letters occur infinitely often in congruent positions modulo p in the given infinite word $h^{\omega}(a)$. It follows from this that, for any coding g, it is decidable whether $g(h^{\omega}(a))$ is ultimately p-periodic. We then consider relational periodicity of morphic words. Relational periods were introduced in [7] as a generalization of periods in partial words. Using our algorithm we can find the smallest similarity relation such that the infinite word is ultimately relationally p-periodic. Finally, we show that given a similarity relation R it is decidable whether an automatic sequence is ultimately weakly R-periodic.

2 Preliminaries

Let $\mathcal{A}$ be a finite alphabet and denote the empty word by ε. The free monoid $\mathcal{A}^*$ is the set of finite words over $\mathcal{A}$ with the operation of concatenation. The length of a finite word w is denoted by $|w|$. The set of letters occurring in a word w is denoted by $\mathrm{Alph}(w)$. An infinite word over $\mathcal{A}$ is a sequence $x = (x_n)_{n \geq 0}$ where $x_n \in \mathcal{A}$ for every $n \geq 0$. Denote the set of all infinite words over $\mathcal{A}$ by $\mathcal{A}^{\omega}$. Let also $\mathcal{A}^{\infty} = \mathcal{A}^* \cup \mathcal{A}^{\omega}$. A word v is a *factor* of $w \in \mathcal{A}^{\infty}$ if $w = xvy$ for some word $x \in \mathcal{A}^*$ and $y \in \mathcal{A}^{\infty}$. The factor v is called a *prefix* of w if $x = \varepsilon$ and it is called a *suffix* of w if $y = \varepsilon$. The set of factors of length n of a word w is denoted by $F_n(w)$.

A mapping $h\colon \mathcal{A}^* \to \mathcal{A}^*$ is a morphism if $h(uv) = h(u)h(v)$ for every $u, v \in \mathcal{A}^*$. A morphism h on $\mathcal{A}^*$ is *prolongable* on a letter a if $h(a) = ay$ and $h^n(y) \neq \varepsilon$ for all integers $n \geq 0$. In this case, $h^n(a)$ is a prefix of $h^{n+1}(a)$ and the following fixed point of h exists:

$$h^{\omega}(a) = \lim_{n \to \infty} h^n(a) = ayh(y)h^2(y) \cdots .$$

An infinite word of the above form $x = h^{\omega}(a)$ is called a *pure morphic word. A morphic word* is an image of a pure morphic words by a coding, i.e., it is of the form $g(h^{\omega}(a))$ for some morphism h prolongable on a and for some coding, i.e., a letter-to-letter morphism $g\colon \mathcal{A}^* \to \mathcal{B}^*$.

An infinite word x is ultimately periodic if it is of the form $x = uv^{\omega} = uvvv\cdots$, where u and v are finite words. The length $|u|$ is a *preperiod* and the length $|v|$ is a *period* of x. An infinite word x is *ultimately p-periodic* if $|v| = p$. The smallest period of x is called *the period* of x.

A *similarity relation* R is a relation on finite words induced by a reflexive and symmetric relation on letters. A word $u = u_1 \cdots u_m$ is *R-similar* to a word $v = v_1 \cdots v_n$ if $n = m$ and $u_i \, R \, v_i$ for all letters $u_i, v_i \in \mathcal{A}$. Note that a similarity relation need not be transitive. For example, if $a \, R \, c$, $b \, R \, c$ and $(a, b) \notin R$, then $abba \, R \, cbca$, but $abba$ and $caca$ are not R-similar since the second letters are not related.

Three kinds of relational periods with respect to a given similarity relation R were introduced in [7]. An infinite word $x = (x_n)_{n\geq 0}$ is *weakly R-periodic* if $x_i \, R \, x_{i+p}$ for every $i \geq 0$ and for some integer $p > 0$, which is called a *weak R-period* of x. It is *externally* R-periodic with an *external R-period* $p > 0$ if there exists a word $v = v_0 \cdots v_{p-1}$ such that $x_i \, R \, v_j$ if $i \equiv j \pmod p$. The word x is *strongly* R-periodic with a *strong R-period* $p > 0$ if $i \equiv j \pmod p$ implies $x_i \, R \, x_j$. It is shown in [7] that all these relational periods are different, and any strong R-period of x is also a weak and an external R-period. An infinite word x is ultimately weakly (resp. externally, strongly) R-periodic if $x = uv$ where u is a finite word (a preperiod) and the infinite word v is weakly (resp. externally, strongly) R-periodic. We say that an infinite word is *ultimately relationally p-periodic* if $p \in \mathbb{N}$ is a weak, external or strong R-period of some suffix of the word.

Let $k \geq 2$ be an integer. An infinite word $x = (x_n)_{n\geq 0} \in \mathcal{B}^\omega$ is *k-automatic* if there exists a finite deterministic automaton with output $M = (Q, q_0, \Sigma_k, \delta, \mathcal{B}, \tau)$ such that $\tau(\delta(q_0, \mathrm{rep}_k(n))) = x_n$ for all $n \geq 0$. Here $\mathrm{rep}_k(n)$ denotes the base-k representation of the integer n, Q is the finite set of states, q_0 is the initial state, $\Sigma_k = \{0, \dots, k-1\}$ is the input alphabet, δ is the transition function and $\tau \colon Q \to \mathcal{B}$ is the output function. By the result of Cobham [4], an infinite word is k-automatic if and only if it is of the form $g(h^\omega(a))$ for a k-uniform morphism h prolongable on a and a coding g. A morphism $h \colon \mathcal{A}^* \to \mathcal{A}^*$ is *k-uniform* if $|h(b)| = k$ for all letters $b \in \mathcal{A}$.

3 Algorithm on *k*-Sets

Let $x = (x_n)_{n\geq 0}$ be an infinite word over $\mathcal{A}$. Let $k \in \{0, 1, \dots, p-1\}$. We say that a letter $a \in \mathcal{A}$ belongs to the *k-set of x modulo p* if there exist infinitely many integers n such that

$$x_n = a \quad \text{and} \quad n \equiv k \pmod p.$$

In this section, we show that given integers k and p, it is decidable whether a letter a belongs to the k-set of a given purely morphic word x modulo p. Consequently, we give an algorithm that counts all k-sets of x modulo p. First, let us prove the following lemma concerning iteration of morphisms.

Lemma 1. *Let $h \colon \mathcal{A}^* \to \mathcal{A}^*$ be a morphism, and let p be a positive integer. There exist integers r and $q > 0$ such that*

$$|h^r(b)| \equiv |h^{r+q}(b)| \pmod p \tag{1}$$

for all $b \in \mathcal{A}$.

Proof. If $p = 1$, then the claim is trivial. Let $p > 1$. Let $M = M(h)$ be the incidence matrix of the morphism h on the alphabet $\mathcal{A} = \{a_1, a_2, \dots, a_d\}$, i.e.,

$$M = (m_{i,j})_{1\leq i,j\leq d},$$

where $m_{i,j}$ denotes the number of occurrences of a_i in $h(a_j)$. Now

$$|h^n(a_j)| = v_t M^n v_j^T,$$

for $v_t = (1, \dots, 1)$ and $v_j = (\delta_{ij})_{1 \le i \le d}$, where $\delta_{jj} = 1$ and $\delta_{ij} = 0$ for $i \ne j$. There are only finitely many matrices $M^n \bmod p$, where the entries are the residues modulo p. Hence the infinite sequence $(u_{j,n})_{n \ge 0}$ of the lengths $u_{j,n} = |h^n(a_j)| \bmod p$ is ultimately periodic. Denote the preperiod of $(u_{j,n})_{n \ge 0}$ by r_j and the period of $(u_{j,n})_{n \ge 0}$ by q_j. Set $r = \max r_j$ and $q = \operatorname{lcm}(q_1, \dots, q_d)$. By the periodicity of the sequences $(u_{j,n})_{n \ge 0}$, we conclude that $|h^r(a_j)| \equiv |h^{r+q}(a_j)| \pmod{p}$ for any $a_j \in \mathcal{A}$. □

Using the above lemma, we now prove our main result.

Theorem 1. *Let $x = (x_n)_{n \ge 0} = h^\omega(a) \in \mathcal{A}^\omega$ for a morphism h prolongable on a, and let b be a letter in $\mathcal{A}$. Given positive integers k and p, it is decidable whether there exist infinitely many n such that $x_n = b$ and $n \equiv k \pmod{p}$.*

Proof. Let r and q be the integers satisfying (1) for the morphism h and the given integer p. Consider the directed graph $G_h = (V, E)$ where the set of vertices V is $\{(a, i) \mid a \in \mathcal{A},\ 0 \le i < p\}$ and there is an edge from (c, i) to (d, j) if there exist a letter $b \in \mathcal{A}$ and integers m and m' such that

$$h^r(b) = y_1 \cdots y_n \text{ and } y_m = c \text{ with } 1 \le m \le n, \tag{2}$$
$$h^q(c) = z_1 \cdots z_{n'} \text{ and } z_{m'} = d \text{ with } 1 \le m' \le n', \tag{3}$$
$$j - i \equiv |h^q(y_1 \cdots y_{m-1})| + m' - m \pmod{p}. \tag{4}$$

By Lemma 1, this means that if c is at position $i \bmod p$ in x and it is the mth letter of the image $h^r(b)$ for some b in x, then there is d at position $j \bmod p$ of x and it is the m'th letter of the image $h^q(c)$; see Figure 1. Namely, consider the position of d modulo p. If $x_l = b$, then

$$i \equiv |h^r(x_0 \cdots x_{l-1})| + m - 1 \pmod{p}, \tag{5}$$
$$j \equiv |h^{r+q}(x_0 \cdots x_{l-1})| + |h^q(y_1 \cdots y_{m-1})| + m' - 1 \pmod{p}. \tag{6}$$

By (1), we have $|h^{r+q}(x_0 \cdots x_{l-1})| \equiv |h^r(x_0 \cdots x_{l-1})| \pmod{p}$, which together with (5) and (6) implies (4).

Recall that h is prolongable on a. We say that a vertex $(c, i) \in V$ is an *initial vertex* if there exists a letter $b = x_l$ such that $0 \le l < |h^r(a)|$, c is the mth letter of $h^r(b)$ and i satisfies (5). The set of the initial vertices is denoted by V_I. Moreover, if there exist infinitely many paths in G_h starting from some vertex of V_I and ending in $v \in V$, we say that v is a *recurrent vertex.* The set of recurrent vertices is denoted by V_R.

Let $(v_0, v_1, \dots, v_n)$ be a path from some vertex $v_0 \in V_I$ to v_n, where $v_i = (b_i, k_i)$ for $i = 0, \dots, n$. By induction, there exists $l < |h^r(a)|$ such that b_i occurs in the image $h^{r+iq}(x_l)$, and in x there is a position of b_i between $|h^{r+iq}(x_0 \cdots x_{l-1})|$ and $|h^{r+iq}(x_0 \cdots x_l)| + 1$, which is congruent to k_i modulo p. Note that if b_i occurs in the image $h^{r+iq}(x_l)$, then it occurs in the image $h^r(x_m)$ for some letter x_m occurring in the image of $h^{iq}(x_l)$. The letter x_m corresponds to b occurring in (2) in the definition of the graph G_h. Note also that, for any position n, we can find i and $l < |h^r(a)|$ such that x_n is in a position between $|h^{r+iq}(x_0 \cdots x_{l-1})|$ and

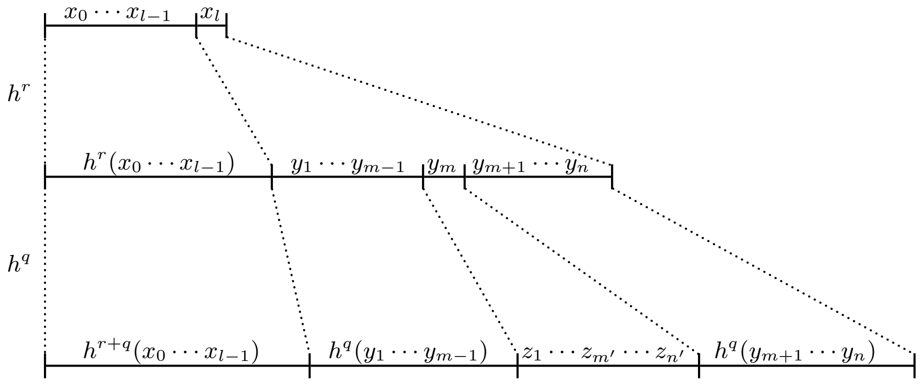

$x_0 \cdots x_{l-1}$
x_l
h^r
$h^r(x_0 \cdots x_{l-1})$
$y_1 \cdots y_{m-1}$
y_m
$y_{m+1} \cdots y_n$
h^q
$h^{r+q}(x_0 \cdots x_{l-1})$
$h^q(y_1 \cdots y_{m-1})$
$z_1 \cdots z_{m'} \cdots z_{n'}$
$h^q(y_{m+1} \cdots y_n)$

Fig. 1. Image of x_l

$|h^{r+iq}(x_0 \cdots x_l)| + 1$. Hence, we conclude that (b, k) is in V_R if and only if there exist infinitely many n such that $x_n = b$ and $n \equiv k \pmod{p}$. This proves the claim. □

The proof of the previous theorem gives us an algorithm for finding the k-sets of $h^\omega(a)$ modulo p.

Algorithm. INPUT: morphism h prolongable on a and integer $p > 0$.

1. Find r and q satisfying (1).
2. Using r and q, construct the graph $G_h = (V, E)$ defined in Theorem 1.
3. Find the set of recurrent vertices V_R.
4. For each letter b and integer $k \in \{0, 1, \ldots, p-1\}$, set $b \in C_k$ if $(b, k) \in V_R$.

OUTPUT: k-sets C_k of $h^\omega(a)$ modulo p.

4 Periodicity Results

In this section we show how the algorithm of the previous section can be used for solving two periodicity problems. First, let us consider a decidability question on morphic words.

Theorem 2. *Given a positive integer p, it is decidable whether a morphic word $g(h^\omega(a))$ is ultimately p-periodic.*

Proof. Consider a morphic word $y = (y_n)_{n \geq 0} = g(h^\omega(a))$, where $h \colon \mathcal{A}^* \to \mathcal{A}^*$ is a morphism prolongable on a and $g \colon \mathcal{A}^* \to \mathcal{B}^*$ is a coding. Let $x = (x_n)_{n \geq 0} = h^\omega(a)$. By Theorem 1, all k-sets of x modulo p can be found algorithmically. We claim that the word y is ultimately p-periodic if and only if $g(b) = g(c)$ for all pairs of letters (b, c) such that b and c belong to the same k-set of x modulo p.

First assume that $g(b) = g(c)$ for all pairs (b, c) belonging to some k-set of x modulo p. Consider the subsequence $(x_{pn+k})_{n \geq 0}$ for some $k \in \{0, 1, \ldots, p-1\}$.

After a finite prefix the subsequence contains only letters belonging to the k-set of x modulo p. Now g maps all these letters to a common letter $b_k \in \mathcal{B}$. Since this holds for every $k \in \{0, 1, \ldots, p-1\}$, it follows that $y = g(x) = uv^\omega$ where $v = (b_0 \cdots b_{p-1})^\omega$ and u is some finite word.

Conversely, assume that b and c belong to some k-set of x modulo p and $g(b) \neq g(c)$. Thus, there exist two increasing sequences $(i_n)_{n\geq 1}$ and $(j_n)_{n\geq 1}$ such that $x_{i_n} = b$, $x_{j_n} = c$ and $i_n \equiv k \equiv j_n \pmod{p}$. This means that, for any N, there are positions $i, j > N$ such that $y_i \neq y_j$ and $i \equiv j \pmod{p}$. Hence, y is not ultimately p-periodic. □

Another application of Theorem 1 is the following results concerning relational periods.

Theorem 3. *Given a positive integer p, a morphic word y and a similarity relation R, it is decidable whether y is ultimately strongly (resp. externally, weakly) R-periodic with period p.*

Proof. Let $y = (y_n)_{n\geq 0} = g(h^\omega(a))$, where $h\colon \mathcal{A}^* \to \mathcal{A}^*$ is a morphism prolongable on a and $g\colon \mathcal{A}^* \to \mathcal{B}^*$ is a coding. Denote $x = (x_n)_{n\geq 0} = h^\omega(a)$. By Theorem 1, all k-sets of x modulo p can be found algorithmically.

Similarly to the proof of the previous theorem, we can show that y is ultimately strongly R-periodic with period p if and only if $g(b)\,R\,g(c)$ for all pairs of letters (b, c) belonging to some k-set of x modulo p.

The case of external R-periodicity is a little bit different. We claim that y is ultimately externally R-periodic with period p if and only if for each $k \in \{0, 1, \ldots, p-1\}$ there exists a letter $b_k \in \mathcal{B}$ such that $b_k\,R\,g(c)$ for every letter c in the k-set of x modulo p.

First assume that such letters b_k can be found. Then b_k is related to all letters occurring infinitely many times in the subsequence $(y_{pn+k})_{n\geq 0} = (g(x_{pn+k}))_{n\geq 0}$. Hence, $(y_{pn+k})_{n\geq 0}$ is ultimately externally R-periodic with period 1. Since this holds for every $k \in \{0, 1, \ldots, p-1\}$, we conclude that after a finite prefix the morphic word y is R-similar to $(b_0 \ldots b_{p-1})^\omega$, i.e., y is ultimately externally R-periodic with period p.

Assume next that y is ultimately externally R-periodic with period p. Then for each $k \in \{0, 1, \ldots, p-1\}$, there exists a letter v_k such that $y_i\,R\,v_k$ if $i \equiv k \pmod{p}$ and $i > N$ for some integer N large enough. Take a letter c from the k-set of x modulo p. By definition, there are infinitely many positions i such that $x_i = c$ and $i \equiv k \pmod{p}$. Hence, there are infinitely many position i such that $y_i = g(c)$ and $i \equiv k \pmod{p}$. By assumption, we have $g(c)\,R\,v_k$ and the claim holds.

Finally, consider the weak R-periodicity of y. Here it suffices to find out all words of length $p+1$ that occur infinitely often in y. Denote the set of these factors by W_{p+1}. The word y is ultimately weakly R-periodic with period p if and only if the first and the last letter of every $w \in W_{p+1}$ are R-similar. The set W_{p+1} can be constructed effectively. By the result of Cobham [2, Theorem 7.5.1], we may assume that h is non-erasing.

First, we find the set of all factors of x of length at most $p+1$. Set

$$U_n = \bigcup_{j=1}^{p+1} F_j(h^n(a))$$

for $n \geq 0$. Since $h^n(a)$ is a prefix of $h^{n+1}(a)$ and $x = h^\omega(a)$, we have $U_n \subseteq U_{n+1} \subseteq \cup_{j=1}^{p+1} F_j(x)$ for every $n \geq 0$. Since there are only finitely many factors of length at most $p+1$, there must be an integer N such that $U_N = U_{N+1}$. Since h is non-erasing, a factor v of x of length at most $p+1$ is a factor of $h(u)$ for some word $u \in F_j(x)$ with $j \leq p+1$. This implies that $U_n = U_N$ for every $n \geq N$ and, consequently,

$$F_{p+1} = U_N \cap \mathcal{A}^{p+1}.$$

Let V_{p+1} be the set of factors of x of length $p+1$ that occur infinitely often in x. To find the subset V_{p+1} of F_{p+1} we suggest a graph construction similar to that in Theorem 1. Let $G = (V, E)$ be a directed graph, where $V = F_{p+1}$ and there is an edge from u to v if v is a factor of $h(u)$. By the definition of the graph, a word v belongs to V_{p+1} if and only if there exist a vertex u such that there are infinitely long paths from u to v. The existence of such u can be easily verified. Hence, we can construct the set V_{p+1} and, clearly, we have $W_{p+1} = g(V_{p+1})$. □

For a relation R on the alphabet $\mathcal{A}$, let

$$\mathrm{sz}(R) = |\{(a, b) \mid a\, R\, b\}|.$$

A relation R is smaller than a relation S if $\mathrm{sz}(R) < \mathrm{sz}(S)$. We can measure the degree of periodicity of an infinite word $x \in \mathcal{A}^\omega$ by considering the smallest similarity relation $R_{x,p}$ such that x is ultimately strongly (resp. externally, weakly) $R_{x,p}$-periodic with period p. It is clear that such a relation is unique. We have the following obvious corollary of Theorem 3.

Corollary 1. *Given a positive integer p and a morphic word x, we can effectively find the smallest similarity relation $R_{x,p}$ such that x is ultimately strongly (resp. externally, weakly) $R_{x,p}$-periodic with period p.*

Unfortunately, the general ultimate periodicity problem (without specifying the period p) for morphic words remains unsolved. Note that this very difficult and challenging problem is a special case of the *ultimate relational periodicity problem*:

> *Given a pure morphic word x and a similarity relation R, decide whether x is ultimately strongly (resp. externally, weakly) R-periodic.*

The ultimate periodicity problem for a morphic word $g(h^\omega(a))$ is the above problem where the relation R is defined by the coding g as follows: $a\, R\, b$ if an only if $g(a) = g(b)$. Note that in this case R is an equivalence relation and the definitions of strong, weak and external periods coincide.

However, using the method from [1] we can solve the ultimate relational periodicity problem restricted to automatic sequences and weak relational periods.

The proof of Theorem 4 is similar to corresponding result in [1]. For completeness sake, we give the proof.

Theorem 4. *Given a k-automatic sequence x over $\mathcal{A}$ and a similarity relation R on $\mathcal{A}$, it is decidable whether x is ultimately weakly R-periodic.*

Proof. Let $x = (x_n)_{n \geq 0}$ be an infinite k-automatic sequence. The sequence x is ultimately weakly R-periodic if there exists $P \geq 1$, $N \geq 0$ such that $x_i \, R \, x_{i+P}$ for all $i \geq N$.

First, construct a nondeterministic finite automaton (NFA) that on input (P, N) successively "guesses" the base-k digits of I starting with the least significant digit and accepts if $I \geq N$ and $(x_I, x_{I+P}) \notin R$. Let $M = (Q, q_0, \Sigma_k, \delta, \mathcal{B}, \tau)$ be the finite automaton with output that generates the sequence x, i.e., $x_n = \tau(\delta(q_0, \mathrm{rep}_k(n)))$ for all $n \geq 0$. Then define $M_1 = (Q', q_0', \Sigma_k \times \Sigma_k, \delta', F')$, where

$$\begin{aligned} Q' &= \{<, =, >\} \times \{0,1\} \times Q \times Q, \\ q_0' &= [=, 0, q_0, q_0], \\ F' &= \{[b, 0, q, r] \mid b \in \{>, =\} \text{ and } (\tau(q), \tau(r)) \notin R\}. \end{aligned}$$

Here F' is the set of accepting states, and the input is actually a sequence of pairs

$$(p_0, n_0)(p_1, n_1)(p_2, n_2) \cdots (p_j, n_j),$$

where $p_j p_{j-1} \cdots p_0$ is a base-k representation of P and $n_j n_{j-1} \cdots n_0$ is a base-k representation of N, either one or both padded with leading zeros to ensure that their lengths are equal.

In order to verify that $I \geq N$, we have a flag $\{<, =, >\}$ that tells us how the digits of the guessed I seen so far are in relation to the digits of the input N read so far. Let i' be the next guessed digit of I and let n' be the next read digit of N. We update the flag as follows:

$$u(<, i', n') = \begin{cases} <, \text{ if } i' \leq n'\,; \\ >, \text{ if } i' > n'\,; \end{cases}$$

$$u(=, i', n') = \begin{cases} <, \text{ if } i' < n'\,; \\ =, \text{ if } i' = n'\,; \\ >, \text{ if } i' > n'\,; \end{cases}$$

$$u(>, i', n') = \begin{cases} <, \text{ if } i' < n'\,; \\ >, \text{ if } i' \geq n'\,. \end{cases}$$

In a state $[b, c, q, r] \in Q'$ the component b is the flag, c is the carry bit in the computation $I + P$, q is the state in M reached by the digits of I guessed so far, and r is the state in M reached by the digits of $I + P$ calculated so far. Hence, the transitions $\delta' \colon Q' \times (\Sigma_k \times \Sigma_k) \to 2^{Q'}$ are defined by

$$\delta'([b, c, q, r](p', n')) = \left\{ \left[u(b, i', n'), \left\lfloor \frac{i' + p' + c}{k} \right\rfloor, \delta(q, i'), \delta(r, (i' + p' + c) \bmod k) \right] \middle| \, 0 \leq i' < k \right\}.$$

We must ensure that the acceptance of (P, N) does not depend on the number of leading zeros and that the calculation is correct in the case where the guessed I is longer than the input. Hence, we modify the accepting states of M_1 by constructing a new NFA $\hat{M}_1 = (Q', q'_0, \Sigma_k \times \Sigma_k, \delta', \hat{F})$ where a state $[b, c, q, r]$ belongs to $\hat{F}$ if there exists $j \geq 0$ such that $\delta'([b, c, q, r], (0, 0)^j)$ contains a state in F'.

Then convert $\hat{M}_1$ to a deterministic finite automaton (DFA) M_2 using the subset construction. By interchanging accepting and non-accepting states we obtain a DFA M_3 that accepts (P, N) if and only if $x_I \, R \, x_{I+P}$ for all $I \geq N$. Now x is ultimately weakly R-periodic if and only if M_3 accepts some input (P, N) with $P \geq 1$. This can be checked by creating a DFA M_4 that accepts $\Sigma_k^*(\Sigma_k \setminus \{0\})\Sigma_k^* \times \Sigma_k^*$ and forming the direct product DFA M_5 that accepts exactly the words accepted by both M_3 and M_4. Thus, the word x is ultimately weakly R-periodic if and only if the language accepted by M_5 is not empty. This can be easily checked. □

References

1. Allouche, J.-P., Rampersad, N., Shallit, J.: Periodicity, repetitions, and orbits of an automatic sequence. Theoret. Comput. Sci. 410, 2795–2803 (2009)
2. Allouche, J.-P., Shallit, J.: Automatic sequences: Theory, Applications, Generalizations. Cambridge University Press, Cambridge (2003)
3. Césari, Y., Vincent, M.: Une caractérisation des fonctions périodiques. C. R. Acad. Sci. Paris 286(A), 1175–1177 (1978)
4. Cobham, A.: Uniform tag sequences. Math. Systems Theory 6, 164–192 (1972)
5. Duval, J.-P.: Périodes et répétitions des mots du monoïde libre. Theoret. Comput. Sci. 9, 17–26 (1979)
6. Fine, N.J., Wilf, H.S.: Uniqueness theorem for periodic functions. Proc. Amer. Math. Soc. 16, 109–114 (1965)
7. Halava, V., Harju, T., Kärki, T.: Interaction properties of relational periods. Discrete Math. Theor. Comput. Sci. 10, 87–112 (2008)
8. Harju, T., Linna, M.: On the periodicity of morphisms on free monoids. Theoret. Inform. Appl. 20, 47–54 (1986)
9. Hedlund, G.A., Morse, M.: Symbolic dynamics II: Sturmian trajectories. Amer. J. Math. 62, 1–42 (1940)
10. Honkala, J.: A decision method for the recognizability of sets defined by number systems. Theoret. Inform. Appl. 20, 395–403 (1986)
11. Mignosi, F., Restivo, A.: Periodicity. In: Lothaire, M. (ed.) Algebraic Combinatorics on Words, Encyclopedia of Mathematics and its Applications, vol. 90, pp. 269–311. Cambridge University Press, Cambridge (2002)
12. Pansiot, J.-J.: Decidability of periodicity for infinite words. Theoret. Inform. Appl. 20, 43–46 (1986)

Compressed Conjugacy and the Word Problem for Outer Automorphism Groups of Graph Groups

Niko Haubold, Markus Lohrey, and Christian Mathissen

Institut für Informatik, Universität Leipzig, Germany
{haubold,lohrey,mathissen}@informatik.uni-leipzig.de

Abstract. It is shown that for graph groups (right-angled Artin groups) the conjugacy problem as well as a restricted version of the simultaneous conjugacy problem can be solved in polynomial time even if input words are represented in a compressed form. As a consequence it follows that the word problem for the outer automorphism group of a graph group can be solved in polynomial time.

1 Introduction

Automorphism groups and *outer automorphism groups* of *graph groups* received a lot of interest in the past few years. A graph group $\mathbb{G}(\Sigma, I)$ is given by a finite undirected graph (Σ, I) (without self-loops). The set Σ is the set of generators of $\mathbb{G}(\Sigma, I)$ and every edge $(a, b) \in I$ gives rise to a commutation relation $ab = ba$. Graph groups are also known as *right-angled Artin groups* or *free partially commutative groups*. Graph groups interpolate between finitely generated free groups and finitely generated free Abelian groups. The automorphism group of the free Abelian group $\mathbb{Z}^n$ is $\mathsf{GL}(n, \mathbb{Z})$ and hence finitely generated. By a classical result of Nielsen, also automorphism groups of free groups are finitely generated, see e.g. [14]. For graph groups in general, it was shown by Laurence [10] (building up on previous work by Servatius [19]) that their automorphism groups are finitely generated. Only recently, Day [4] has shown that $\mathsf{Aut}(\mathbb{G}(\Sigma, I))$ is always finitely presented. An overwiew on structural results on automorphism groups of graph groups can be found in [1].

In this paper, we continue the investigation of algorithmic aspects of automorphism groups of graph groups. In [13] it was shown that the word problem for $\mathsf{Aut}(\mathbb{G}(\Sigma, I))$ can be solved in polynomial time. The proof of this result used compression techniques. It is well-known that the word problem for $\mathbb{G}(\Sigma, I)$ can be solved in linear time. In [13], a compressed (or succinct) version of the word problem for graph groups was studied. In this variant of the word problem, the input word is represented succinctly by a so-called *straight-line program*. This is a context free grammar $\mathbb{A}$ that generates exactly one word $\mathsf{val}(\mathbb{A})$, see Sec. 2.1. Since the length of this word may grow exponentially with the size (number of productions) of the SLP $\mathbb{A}$, SLPs can be seen indeed as a succinct string representation. SLPs turned out to be a very flexible compressed representation of strings, which are well suited for studying algorithms for compressed data, see e.g. [6,11,17]. In [13,18] it was shown that the word problem for the automorphism group $\mathsf{Aut}(G)$ of a group G can be reduced in polynomial time to the *compressed word problem* for G, where the input word is succinctly given by an SLP. In [18], it was shown that the compressed word problem for a finitely generated free group F can be solved in polynomial time and in [13] this result was extended to graph groups. It follows that the word problem for $\mathsf{Aut}(\mathbb{G}(\Sigma, I))$ can be solved in polynomial time. Recently, Macdonald [15] has shown that also the compressed word problem for every fully residually free group can be solved in polynomial time.

It is not obvious to carry over these complexity results from $\mathsf{Aut}(\mathbb{G}(\Sigma, I))$ to the *outer* automorphism group $\mathsf{Out}(\mathbb{G}(\Sigma, I)) = \mathsf{Aut}(\mathbb{G}(\Sigma, I))/\mathrm{Inn}(\mathbb{G}(\Sigma, I))$ (see Sec. 2.3 for the definition). Nevertheless, Schleimer proved in [18] that the word problem for the outer automorphism group of a finitely generated free group can be decided in polynomial time. For this, he used a compressed variant of the simultaneous conjugacy problem in free groups. In this paper, we generalize Schleimer's result to graph groups: For every graph (Σ, I), the word problem for $\mathsf{Out}(\mathbb{G}(\Sigma, I))$ can be solved in polynomial time. Analogously to Schleimer's approach for free groups, we reduce the word problem for $\mathsf{Out}(\mathbb{G}(\Sigma, I))$ to a compressed variant of the simultaneous conjugacy problem in $\mathbb{G}(\Sigma, I)$. In this problem, we are given an SLP $\mathbb{A}_a$ for every generator $a \in \Sigma$, and the question is whether there exists $x \in \mathbb{G}(\Sigma, I)$ such that $a = x\,\mathsf{val}(\mathbb{A}_a)\,x^{-1}$ for all $a \in \Sigma$. A large part of this paper develops a polynomial time algorithm for this problem. Moreover, we also present a polynomial time algorithm for the compressed version of the classical conjugacy problem in graph groups: In this problem, we are given two SLPs $\mathbb{A}$ and $\mathbb{B}$ and we ask whether there exists $x \in \mathbb{G}(\Sigma, I)$ such that $\mathsf{val}(\mathbb{A}) = x\,\mathsf{val}(\mathbb{B})x^{-1}$ in $\mathbb{G}(\Sigma, I)$. For the non-compressed version of this problem, a linear time algorithm was presented in [21] based on [12]. In [3] this result was generalized to various subgroups of graph groups.

Missing proofs can be found in the full version [9] of this extended abstract.

2 Preliminaries

Let Σ be a finite alphabet. For a word $s = a_1 \cdots a_m$ ($a_i \in \Sigma$) let $|s| = m$, $\mathsf{alph}(s) = \{a_1, \ldots, a_m\}$, $s[i] = a_i$ for $1 \leq i \leq m$, and $|s|_a = |\{k \mid s[k] = a\}|$ for $a \in \Sigma$. We use $\Sigma^{-1} = \{a^{-1} \mid a \in \Sigma\}$ to denote a disjoint copy of Σ and let $\Sigma^{\pm 1} = \Sigma \cup \Sigma^{-1}$. Define $(a^{-1})^{-1} = a$. This defines an involution $^{-1} : \Sigma^{\pm 1} \to \Sigma^{\pm 1}$, which can be extended to an involution on $(\Sigma^{\pm 1})^*$ by setting $(a_1 \cdots a_n)^{-1} = a_n^{-1} \cdots a_1^{-1}$.

2.1 Straight-Line Programs

We are using straight-line programs as a succinct representation of strings with reoccurring subpatterns. A *straight-line program (SLP) over the alphabet Γ* is a context free grammar $\mathbb{A} = (V, \Gamma, S, P)$, where V is the set of *nonterminals*, Γ is the set of *terminals*, $S \in V$ is the *initial nonterminal*, and $P \subseteq V \times (V \cup \Gamma)^*$ is the set of *productions* such that (i) for every $X \in V$ there is exactly one $\alpha \in (V \cup \Gamma)^*$ with $(X, \alpha) \in P$ and (ii) there is no cycle in the relation $\{(X, Y) \in V \times V \mid \exists \alpha : (X, \alpha) \in P, Y \in \mathsf{alph}(\alpha)\}$. These conditions ensure that the language generated by the straight-line program $\mathbb{A}$ contains exactly one word $\mathsf{val}(\mathbb{A})$. Moreover, every nonterminal $X \in V$ generates exactly one word that is denoted by $\mathsf{val}_{\mathbb{A}}(X)$, or briefly $\mathsf{val}(X)$, if $\mathbb{A}$ is clear from the context. The size of $\mathbb{A}$ is $|\mathbb{A}| = \sum_{(X,\alpha) \in P} |\alpha|$. An SLP can be transformed in polynomial time into an equivalent SLP in *Chomsky normal form*, which means that all productions have the form $A \to BC$ or $A \to a$ with $A, B, C \in V$ and $a \in \Gamma$. For an SLP $\mathbb{A}$ over $\Sigma^{\pm 1}$

(w.l.o.g. in Chomsky normal form) we denote with $\mathbb{A}^{-1}$ the SLP that has for each terminal rule $A \to a$ from $\mathbb{A}$ the terminal rule $A \to a^{-1}$ and for each nonterminal rule $A \to BC$ from $\mathbb{A}$ the nonterminal rule $A \to CB$. Clearly, $\mathsf{val}(\mathbb{A}^{-1}) = \mathsf{val}(\mathbb{A})^{-1}$. Let us state some simple algorithmic problems that can be easily solved in polynomial time:

– Given an SLP $\mathbb{A}$, calculate $|\mathsf{val}(\mathbb{A})|$ and $\mathsf{alph}(\mathsf{val}(\mathbb{A}))$.
– Given an SLP $\mathbb{A}$ and a number $i \in \{1, \ldots, |\mathsf{val}(\mathbb{A})|\}$, calculate $\mathsf{val}(\mathbb{A})[i]$.
– Given an SLP $\mathbb{A}$ (let $\mathsf{val}(\mathbb{A}) = a_1 \cdots a_n$) and two numbers $1 \leq i \leq j \leq n$, compute and SLP $\mathbb{B}$ with $\mathsf{val}(\mathbb{B}) = a_i \cdots a_j$.

In [17], Plandowski presented a polynomial time algorithm for testing whether $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$ for two given SLPs $\mathbb{A}$ and $\mathbb{B}$. A cubic algorithm was presented by Lifshits [11]. In fact, Lifshits gave an algorithm for compressed pattern matching: given SLPs $\mathbb{A}$ and $\mathbb{B}$, is $\mathsf{val}(\mathbb{A})$ a factor of $\mathsf{val}(\mathbb{B})$? His algorithm runs in time $O(|\mathbb{A}| \cdot |\mathbb{B}|^2)$.

2.2 Trace Monoids and Graph Groups

We introduce some notions from trace theory, see [5] for more details. An *independence alphabet* is a pair (Σ, I) where Σ is a finite alphabet and $I \subseteq \Sigma \times \Sigma$ is an irreflexive and symmetric relation. The complementary graph (Σ, D) with $D = (\Sigma \times \Sigma) \setminus I$ is called a *dependence alphabet*. The *trace monoid* $\mathbb{M}(\Sigma, I)$ is defined as the quotient $\mathbb{M}(\Sigma, I) = \Sigma^*/\{ab = ba \mid (a,b) \in I\}$ with concatenation as operation and the empty word as the neutral element. This monoid is cancellative and its elements are called *traces*. The trace represented by the word $s \in \Sigma^*$ is denoted by $[s]_I$. For $a \in \Sigma$ let $I(a) = \{b \in \Sigma \mid (a,b) \in I\}$ be the letters that commute with a. For traces u, v we denote with uIv the fact that $\mathsf{alph}(u) \times \mathsf{alph}(v) \subseteq I$. For $\Gamma \subseteq \Sigma$ we say that Γ is *connected* if the subgraph of (Σ, D) induced by Γ is connected. For a trace u let $\max(u) = \{a \mid u = va \text{ for } a \in \Sigma, v \in \mathbb{M}(\Sigma, I)\}$ be the set of possible last letters of u and $\min(u) = \{a \mid u = av \text{ for } a \in \Sigma, v \in \mathbb{M}(\Sigma, I)\}$ be the set of possible first letters.

A convenient representation for traces are *dependence graphs*, which are node-labeled directed acyclic graphs. For a word $w \in \Sigma^*$ the dependence graph D_w has vertex set $\{1, \ldots, |w|\}$ where the node i is labeled with $w[i]$. There is an edge from vertex i to j if and only if $i < j$ and $(w[i], w[j]) \in D$. It is easy to see that for two words $w, w' \in \Sigma^*$ we have $[w]_I = [w']_I$ if and only if D_w and $D_{w'}$ are isomorphic. Hence, we can speak of *the* dependence graph of a trace.

For background in combinatorial group theory see [14]. The *free group* generated by Σ can be defined as the quotient monoid $F(\Sigma) = (\Sigma^{\pm 1})^*/\{aa^{-1} = \varepsilon \mid a \in \Sigma^{\pm 1}\}$. For an independence alphabet (Σ, I) the *graph group* $\mathbb{G}(\Sigma, I)$ is the quotient group $\mathbb{G}(\Sigma, I) = F(\Sigma)/\{ab = ba \mid (a,b) \in I\}$. From the independence alphabet (Σ, I) we derive the independence alphabet $(\Sigma^{\pm 1}, \{(a^i, b^j) \mid i, j \in \{-1, 1\}, (a,b) \in I\})$. Abusing notation, we denote the independence relation of this alphabet again with I. Note that $(a,b) \in I$ implies $a^{-1}b = ba^{-1}$ in $\mathbb{G}(\Sigma, I)$. Thus, we have $\mathbb{G}(\Sigma, I) = \mathbb{M}(\Sigma^{\pm 1}, I)/\{aa^{-1} = \varepsilon \mid a \in \Sigma^{\pm 1}\}$. Graph groups are also known as right-angled Artin groups and free partially commutative groups.

2.3 (Outer) Automorphism Groups

The *automorphism group* $\mathsf{Aut}(G)$ of a group G is the set of all automorphisms of G with composition as operation and the identity mapping as the neutral element. An automorphism φ is called *inner* if there is $x \in G$ such that $\varphi(y) = xyx^{-1}$ for all $y \in G$. The set of all inner automorphisms of G forms the *inner automorphism group* $\mathsf{Inn}(G)$ of G. This is easily seen to be a normal subgroup of $\mathsf{Aut}(G)$ and the quotient group $\mathsf{Out}(G) = \mathsf{Aut}(G)/\mathsf{Inn}(G)$ is called the *outer automorphism group* of G.

Assume that $\mathsf{Aut}(G)$ is finitely generated (which, in general, won't be the case, even if G is finitely generated) and let $\Psi = \{\psi_1, \ldots, \psi_k\}$ be a monoid generating set for $\mathsf{Aut}(G)$, i.e., every automorphism of G can be composed from the automorphisms in Ψ. Then Ψ also generates $\mathsf{Out}(G)$ where we identify ψ_i with its coset $\psi_i \cdot \mathsf{Inn}(G) \in \mathsf{Out}(G)$ for $i \in \{1, \ldots, k\}$. Then the *word problem* for the outer automorphism group can be viewed as the following decision problem:

INPUT: A word $w \in \Psi^*$.
QUESTION: Does $w = 1$ in $\mathsf{Out}(G)$?

Since an automorphism belongs to the same coset (w.r.t. $\mathsf{Inn}(G)$) as the identity if and only if it is inner, we can rephrase the word problem for $\mathsf{Out}(G)$ as follows:

INPUT: A word $w \in \Psi^*$.
QUESTION: Does w represent an element of $\mathsf{Inn}(G)$ in $\mathsf{Aut}(G)$?

Building on results from [19], Laurence has shown in [10] that automorphism groups of graph groups are finitely generated. Recently, Day [4] proved that automorphism groups of graph groups are in fact finitely presented. In this paper, we present a polynomial time algorithm for the word problem for $\mathsf{Out}(\mathbb{G}(\Sigma, I))$.

3 Main Results

In this section we will present the main results of this paper, the proofs of which are subject to the rest of the paper. Our group theoretical main result is:

Theorem 1. *Let (Σ, I) be a fixed independence alphabet. Then, the word problem for the group $\mathsf{Out}(\mathbb{G}(\Sigma, I))$ can be solved in polynomial time.*

In order to solve the word problem for $\mathsf{Out}(\mathbb{G}(\Sigma, I))$ in polynomial time, we will consider (following Schleimer's approach for free groups [18]) compressed conjugacy problems in $\mathbb{G}(\Sigma, I)$. The most general of these compressed conjugacy problems is the *simultaneous compressed conjugacy problem* for $\mathbb{G}(\Sigma, I)$:

INPUT: SLPs $\mathbb{A}_1, \mathbb{B}_1, \ldots, \mathbb{A}_n, \mathbb{B}_n$ over $\Sigma^{\pm 1}$.
QUESTION: $\exists x \in (\Sigma^{\pm 1})^* \; \forall i \in \{1, \ldots, n\} : \mathsf{val}(\mathbb{A}_i) = x \, \mathsf{val}(\mathbb{B}_i) x^{-1}$ in $\mathbb{G}(\Sigma, I)$?

The simultaneous (non-compressed) conjugacy problem also appears in connection with group-based cryptography [16]. Unfortunately, we don't know, whether the simultaneous compressed conjugacy problem can be solved in polynomial time. But, in order to deal with the word problem for $\mathsf{Out}(\mathbb{G}(\Sigma, I))$, a restriction of this problem suffices, where the SLPs $\mathbb{B}_1, \ldots, \mathbb{B}_n$ from the simultaneous compressed conjugacy problem are the letters from Σ. We call this problem the *restricted simultaneous compressed conjugacy problem*, briefly $\mathsf{RSCCP}(\Sigma, I)$:

INPUT: SLPs $\mathbb{A}_a$ ($a \in \Sigma$) over $\Sigma^{\pm 1}$.
QUESTION: $\exists x \in (\Sigma^{\pm 1})^* \; \forall a \in \Sigma : \mathsf{val}(\mathbb{A}_a) = xax^{-1}$ in $\mathbb{G}(\Sigma, I)$?

An x such that $\mathsf{val}(\mathbb{A}_a) = xax^{-1}$ in $\mathbb{G}(\Sigma, I)$ for all $a \in \Sigma$ is called a *solution* of the $\mathsf{RSCCP}(\Sigma, I)$-instance. The following theorem will be shown in Sec. 5:

Theorem 2. *Let (Σ, I) be a fixed independence alphabet. Then, $\mathsf{RSCCP}(\Sigma, I)$ can be solved in polynomial time. Moreover, in case a solution exists, one can compute an SLP for a solution in polynomial time.*

Proof of Thm. 1 using Thm. 2. Fix a finite monoid generating set Φ for $\mathsf{Aut}(\mathbb{G}(\Sigma, I))$. Let $\varphi = \varphi_1 \cdots \varphi_n$ with $\varphi_1, \ldots, \varphi_n \in \Phi$ be the input. By [18] we can compute in polynomial time SLPs $\mathbb{A}_a$ ($a \in \Sigma$) over $\Sigma^{\pm 1}$ with $\mathsf{val}(\mathbb{A}_a) = \varphi(a)$ in $\mathbb{G}(\Sigma, I)$ for all $a \in \Sigma$. The automorphism φ is inner iff there exists x such that $\mathsf{val}(\mathbb{A}_a) = xax^{-1}$ in $\mathbb{G}(\Sigma, I)$ for all $a \in \Sigma$. This can be decided in polynomial time by Thm. 2. $\square$

Finally, we will also consider a compressed variant of the classical conjugacy problem for $\mathbb{G}(\Sigma, I)$. Recall that the *conjugacy problem* for a finitely generated group G asks, whether two given elements $g, h \in G$ are *conjugated*, i.e., whether there exists $x \in G$ with $g = xhx^{-1}$. The *compressed conjugacy problem* for the graph group $\mathbb{G}(\Sigma, I)$, $\mathsf{CCP}(\Sigma, I)$ for short, is the following problem:

INPUT: SLPs $\mathbb{A}$ and $\mathbb{B}$ over $\Sigma^{\pm 1}$.
QUESTION: Are $\mathsf{val}(\mathbb{A})$ and $\mathsf{val}(\mathbb{B})$ conjugated in $\mathbb{G}(\Sigma, I)$?

Theorem 3. *Let (Σ, I) be a fixed independence alphabet. Then, $\mathsf{CCP}(\Sigma, I)$ can be solved in polynomial time.*

We will prove Thm. 3 in Sec. 7. It is important in Thm. 1–3 that we fix the independence alphabet (Σ, I). It is open whether these results also hold if (Σ, I) is part of the input.

4 Further Facts for Traces

In this section, we state some simple facts on trace monoids, which will be needed later. Fix the trace monoid $\mathbb{M}(\Sigma, I)$. A trace u is said to be a *prefix* of a trace w, briefly $u \preceq w$, if $uv = w$ for some trace v. The prefixes of a trace w correspond to the downward-closed node sets of the dependence graph of w. Analogously, a trace v is a *suffix* of a trace w if $uv = w$ for some trace u. For two traces u, v, the *infimum* $u \sqcap v$ is the largest trace s w.r.t. $\preceq$ such that $s \preceq u$ and $s \preceq v$; it always exists [2]. With $u \setminus v$ we denote the unique trace t such that $u = (u \sqcap v)t$; uniqueness follows from the fact that $\mathbb{M}(\Sigma, I)$ is cancellative. Note that $u \setminus v = u \setminus (u \sqcap v)$. The *supremum* $u \sqcup v$ of two traces u, v is the smallest trace s w.r.t. $\preceq$ such that $u \preceq s$ and $v \preceq s$ if any such trace exists. We can define the supremum of several traces $w_1, \ldots, w_n$ by induction: $w_1 \sqcup \cdots \sqcup w_n = (w_1 \sqcup \cdots \sqcup w_{n-1}) \sqcup w_n$.

Lemma 4 ([2])**.** *The trace $u \sqcup v$ exists if and only if $(u \setminus v)\, I\, (v \setminus u)$, in which case we have $u \sqcup v = (u \sqcap v)\,(u \setminus v)\,(v \setminus u)$.*

Example 5. We consider the following independence alphabet (Σ, I) and the corresponding dependence alphabet (Σ, D):

$$(\Sigma, I) \quad \begin{tikzcd}[column sep=tiny, row sep=tiny] c \arrow[rr, dash] & & a \arrow[dl, dash] \arrow[dr, dash] & & & \\ & e \arrow[rr, dash] & & d \arrow[rr, dash] & & b \end{tikzcd} \qquad (\Sigma, D) \quad \begin{tikzcd}[column sep=tiny, row sep=tiny] a \arrow[dr, dash] & & e \arrow[dl, dash] \arrow[dr, dash] & & & \\ & b \arrow[rr, dash] & & c \arrow[rr, dash] & & d \end{tikzcd}$$

Consider the words $u = aeadbacdd$ and $v = eaabdcaeb$. The dependence graphs D_u and D_v look as follows, where we label the vertex i with the letter $u[i]$ (resp. $v[i]$):

$$\begin{tikzcd}
 & a \arrow[r] & a \arrow[dr] & & a & & \\
D_u & e \arrow[rr] & & b \arrow[ur] \arrow[dr] & & & \\
 & d \arrow[rrr] & & & c \arrow[r] & d \arrow[r] & d
\end{tikzcd}
\qquad
\begin{tikzcd}
 & a \arrow[r] & a \arrow[dr] & & a \arrow[rr] & & b \\
D_v & e \arrow[rr] & & b \arrow[ur] \arrow[dr] & & e \arrow[ur] & \\
 & d \arrow[rrr] & & & c \arrow[ur] & &
\end{tikzcd}$$

Then we have $u \sqcap v = aeadbac =: p$. Since $u \setminus p = dd$ and $v \setminus p = eb$ we have $(u \setminus p)I(v \setminus p)$ and hence the supremum $s = u \sqcup v = aeadbacddeb$ is defined. The dependence graphs for p and s are:

$$\begin{tikzcd}
 & a \arrow[r] & a \arrow[dr] & & a \\
D_p & e \arrow[rr] & & b \arrow[ur] \arrow[dr] & \\
 & d \arrow[rrr] & & & c
\end{tikzcd}
\qquad
\begin{tikzcd}
 & a \arrow[r] & a \arrow[dr] & & a \arrow[rr] & & b \\
D_s & e \arrow[rr] & & b \arrow[ur] \arrow[dr] & & e \arrow[ur] & \\
 & d \arrow[rrr] & & & c \arrow[ur] \arrow[r] & d \arrow[r] & d
\end{tikzcd}$$

The following lemma is a basic statement for traces, see for example [5, Sec. 1.3]:

Lemma 6 (Levi's Lemma). *Let u_1, u_2, v_1, v_2 be traces with $u_1u_2 = v_1v_2$. Then there exist traces x, y_1, y_2, z such that $y_1 I y_2$, $u_1 = xy_1$, $u_2 = y_2z$, $v_1 = xy_2$, and $v_2 = y_1z$.*

A *trace rewriting system* R over $\mathbb{M}(\Sigma, I)$ is just a finite subset of $\mathbb{M}(\Sigma, I) \times \mathbb{M}(\Sigma, I)$ [5]. The *one-step rewrite relation* $\to_R \subseteq \mathbb{M}(\Sigma, I) \times \mathbb{M}(\Sigma, I)$ is defined as: $x \to_R y$ if and only if there are $u, v \in \mathbb{M}(\Sigma, I)$ and $(\ell, r) \in R$ such that $x = u\ell v$ and $y = urv$. A trace u is *R-irreducible* if no trace v with $u \to_R v$ exists. The set of all R-irreducible traces is denoted by $\mathsf{IRR}(R)$. If R is Noetherian and confluent (see e.g. [5, Sec. 5.1] for definitions), then for every trace u, there exists a unique *normal form* $\mathsf{NF}_R(u) \in \mathsf{IRR}(R)$ such that $u \xrightarrow{*}_R \mathsf{NF}_R(u)$.

Let us now work in the trace monoid $\mathbb{M}(\Sigma^{\pm 1}, I)$. For a trace $u = [a_1 \cdots a_n]_I \in \mathbb{M}(\Sigma^{\pm 1}, I)$ we denote with u^{-1} the trace $u^{-1} = [a_n^{-1} \cdots a_1^{-1}]_I$. It is easy to see that this definition is independent of the chosen representative $a_1 \cdots a_n$ of the trace u. It follows that we have $[\mathsf{val}(\mathbb{A})]_I^{-1} = [\mathsf{val}(\mathbb{A}^{-1})]_I$ for an SLP $\mathbb{A}$. For the rest of the paper, we fix the trace rewriting system $R = \{([aa^{-1}]_I, [\varepsilon]_I) \mid a \in \Sigma^{\pm 1}\}$ over the trace monoid $\mathbb{M}(\Sigma^{\pm 1}, I)$. This system is Noetherian (trivial) and, by [5,20], also confluent. For traces $u, v \in \mathbb{M}(\Sigma^{\pm 1}, I)$ we have $u = v$ in $\mathbb{G}(\Sigma, I)$ if and only if $\mathsf{NF}_R(u) = \mathsf{NF}_R(v)$. Using these facts, it was shown in [5,20] that the word problem for $\mathbb{G}(\Sigma, I)$ can be solved in linear time (on the RAM model).

We close this section with some results concerning SLP-compressed traces. A simple observation is that for given SLPs $\mathbb{A}$ and $\mathbb{B}$ one can decide in polynomial time whether $[\mathsf{val}(\mathbb{A})]_I = [\mathsf{val}(\mathbb{B})]_I$. The projection lemma for traces [5, Cor. 1.4.8] allows to reduce this question to equality testing for SLP-compressed strings [17]. Much harder to prove is:

Theorem 7 ([13]). *For a given SLP $\mathbb{A}$ over $\Sigma^{\pm 1}$ one can compute in polynomial time an SLP $\mathbb{B}$ with $[\mathsf{val}(\mathbb{B})]_I = \mathsf{NF}_R([\mathsf{val}(\mathbb{A})]_I)$.*

Thm. 7 implies that the compressed word problem for a graph group can be solved in polynomial time.

Theorem 8 ([13]). *For given SLPs $\mathbb{A}_0$ and $\mathbb{A}_1$ over $\Sigma^{\pm 1}$, we can compute in polynomial time SLPs $\mathbb{P}$, $\mathbb{D}_0$, $\mathbb{D}_1$ with $[\mathsf{val}(\mathbb{P})]_I = [\mathsf{val}(\mathbb{A}_0)]_I \sqcap [\mathsf{val}(\mathbb{A}_1)]_I$ and $[\mathsf{val}(\mathbb{D}_i)]_I = [\mathsf{val}(\mathbb{A}_i)]_I \setminus [\mathsf{val}(\mathbb{A}_{1-i})]_I$ for $i \in \{0, 1\}$.*

An immediate corollary of Thm. 8 and Lemma 4 is:

Corollary 9. *Let r be a fixed constant. Then, for given SLPs $\mathbb{V}_1, \ldots, \mathbb{V}_r$ over $\Sigma^{\pm 1}$, one can check in polynomial time whether $[\mathsf{val}(\mathbb{V}_1)]_I \sqcup \cdots \sqcup [\mathsf{val}(\mathbb{V}_r)]_I$ exists, and in case it exists one can compute an SLP $\mathbb{S}$ with $[\mathsf{val}(\mathbb{S})]_I = [\mathsf{val}(\mathbb{V}_1)]_I \sqcup \cdots \sqcup [\mathsf{val}(\mathbb{V}_r)]_I$ in polynomial time.*

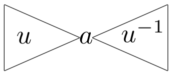
It is important that we fix the number r of SLPs in Cor. 9: Each application of Thm. 8 increase the size of the SLP polynomially. Hence, a non-fixed number of applications might lead to an exponential blow-up.

5 Restricted Simultaneous Compressed Conjugacy

A *double a-cone* ($a \in \Sigma^{\pm 1}$) is an R-irreducible trace of the form uau^{-1} with $u \in \mathbb{M}(\Sigma^{\pm 1}, I)$. We first state several results on double a-cones, which will be used in the proof of Thm. 2. The following characterization can be easily shown:

Lemma 10. *A trace uau^{-1} is a double a-cone if and only if $u \in \mathsf{IRR}(R)$ and $\max(u) \cap (\{a, a^{-1}\} \cup I(a)) = \emptyset$.*

It follows that every letter in a double a-cone either lies before or after the central letter a. Its dependence graph always has the following form:



By the following lemma, each double a-cone has a unique factorization of the form $u_1 b u_2$ with $|u_1| = |u_2|$.

Lemma 11. *Let $v = uau^{-1}$ be a double a-cone and let $v = u_1 b u_2$ with $b \in \Sigma^{\pm 1}$ and $|u_1| = |u_2|$. Then $a = b$, $u_1 = u$ and $u_2 = u^{-1}$.*

Lemma 11 together with standard techniques for SLP-compressed strings (in particular, the polynomial equality test for SLP-compressed strings [11,17]) implies:

Lemma 12. *For a given SLP $\mathbb{A}$ with $[\mathsf{val}(\mathbb{A})]_I \in \mathsf{IRR}(R)$ and $a \in \Sigma^{\pm 1}$, one can check in polynomial time whether $[\mathsf{val}(\mathbb{A})]_I$ is a double a-cone, and in case it is, one can compute in polynomial time an SLP $\mathbb{V}$ with $[\mathsf{val}(\mathbb{A})]_I = [\mathsf{val}(\mathbb{V})\, a\, \mathsf{val}(\mathbb{V}^{-1})]_I$.*

Lemma 13. *Let $w \in \mathbb{M}(\Sigma^{\pm 1}, I)$ be R-irreducible and $a \in \Sigma^{\pm 1}$. Then there exists $x \in \mathbb{M}(\Sigma^{\pm 1}, I)$ with $w = xax^{-1}$ in $\mathbb{G}(\Sigma, I)$ if and only if w is a double a-cone.*

Lemma 13 can be shown by induction on the number of R-rewrite steps from xax^{-1} to $w \in \mathsf{IRR}(R)$. Finally, our main lemma on double a-cones is:

Lemma 14. *Let $w_a, v_a \in \mathbb{M}(\Sigma^{\pm 1}, I)$ ($a \in \Sigma$) be R-irreducible such that $w_a = v_a a v_a^{-1}$ in $\mathbb{M}(\Sigma^{\pm 1}, I)$ for all $a \in \Sigma$ (thus, every w_a is a double a-cone). If there is a trace $x \in \mathbb{M}(\Sigma^{\pm 1}, I)$ with $\forall a \in \Sigma : xax^{-1} = w_a$ in $\mathbb{G}(\Sigma, I)$, then $s = \bigsqcup_{a \in \Sigma} v_a$ exists and $sas^{-1} = w_a$ in $\mathbb{G}(\Sigma, I)$ for all $a \in \Sigma$.*

Now we are in the position to prove Thm. 2: Let $\mathbb{A}_a$ ($a \in \Sigma$) be the input SLPs. We have to check whether there exists x such that $\mathsf{val}(\mathbb{A}_a) = xax^{-1}$ in $\mathbb{G}(\Sigma, I)$ for all $a \in \Sigma$. Thm. 7 allows us to assume that $[\mathsf{val}(\mathbb{A}_a)]_I \in \mathsf{IRR}(R)$ for all $a \in \Sigma$. We first check whether every trace $[\mathsf{val}(\mathbb{A}_a)]_I$ is a double a-cone. By Lemma 12 this is possible in polynomial time. If there exists $a \in \Sigma$ such that $[\mathsf{val}(\mathbb{A}_a)]_I$ is not a double a-cone, then we can reject by Lemma 13. Otherwise, we can compute (using again Lemma 12) SLPs $\mathbb{V}_a$ ($a \in \Sigma$) such that $[\mathsf{val}(\mathbb{A}_a)]_I = [\mathsf{val}(\mathbb{V}_a)\, a \,\mathsf{val}(\mathbb{V}_a^{-1})]_I$ in $\mathbb{M}(\Sigma^{\pm 1}, I)$. Finally, by Lemma 14, it suffices to check whether $s = \bigsqcup_{a\in\Sigma}[\mathsf{val}(\mathbb{V}_a)]_I$ exists and whether $\mathsf{NF}_R(sas^{-1}) = [\mathsf{val}(\mathbb{A}_a)]_I$ for all $a \in \Sigma$. This is possible in polynomial time by Thm. 7 and Cor. 9 (recall that $|\Sigma|$ is a constant in our consideration). Moreover, if the supremum s exists, then we can compute in polynomial time an SLP $\mathbb{S}$ with $[\mathsf{val}(\mathbb{S})]_I = s$, which is a solution for our $\mathsf{RSCCP}(\Sigma, I)$-instance. □

6 A Pattern Matching Algorithm for Connected Patterns

For the proof of Thm. 3 we need a pattern matching algorithm for compressed traces. For traces v, w we say that v is a factor of w if there are traces x, y with $w = xvy$. We consider the following problem and show that it can be solved in polynomial time if the independence alphabet (Σ, I) satisfies certain conditions.

INPUT: An independence alphabet (Σ, I) and two SLPs $\mathbb{T}$ and $\mathbb{P}$ over Σ.
QUESTION: Is $[\mathsf{val}(\mathbb{P})]_I$ a factor of $[\mathsf{val}(\mathbb{T})]_I$?

We write $\mathsf{alph}(\mathbb{T})$ and $\mathsf{alph}(\mathbb{P})$ for $\mathsf{alph}(\mathsf{val}(\mathbb{T}))$ and $\mathsf{alph}(\mathsf{val}(\mathbb{P}))$, respectively. We assume in the following that the SLPs $\mathbb{T} = (V, \Sigma, S, P)$ and $\mathbb{P}$ are in Chomsky normal form. Let $\Gamma \subseteq \Sigma$. We denote by π_Γ the homomorphism $\pi_\Gamma : \mathbb{M}(\Sigma, I) \to \mathbb{M}(\Gamma, I \cap (\Gamma \times \Gamma))$ with $\pi_\Gamma(a) = a$ for $a \in \Gamma$ and $\pi_\Gamma(a) = \varepsilon$ for $a \in \Sigma \setminus \Gamma$. Let $V^\Gamma = \{X^\Gamma \mid X \in V\}$ be a disjoint copy of V. For each production $p \in P$ define a new production p^Γ as follows. If p is of the form $X \to a$ ($a \in \Sigma$), then let $p^\Gamma = (X^\Gamma \to \pi_\Gamma(a))$. If $p \in P$ is of the form $X \to YZ$ ($Y, Z \in V$) define $p^\Gamma = (X^\Gamma \to Y^\Gamma Z^\Gamma)$. We denote with $\mathbb{T}^\Gamma$ the SLP $(V^\Gamma, \Gamma, S^\Gamma, P^\Gamma)$ where $P^\Gamma = \{p^\Gamma \mid p \in P\}$. Obviously, $\mathsf{val}(\mathbb{T}^\Gamma) = \pi_\Gamma(\mathsf{val}(\mathbb{T}))$.

In order to develop a polynomial time algorithm for the problem stated above we need a succinct representation for an occurrence of $\mathbb{P}$ in $\mathbb{T}$. Since $[\mathsf{val}(\mathbb{P})]_I$ is a factor of $[\mathsf{val}(\mathbb{T})]_I$ iff there is a prefix $u \preceq [\mathsf{val}(\mathbb{T})]_I$ such that $u[\mathsf{val}(\mathbb{P})]_I \preceq [\mathsf{val}(\mathbb{T})]_I$, we will in fact compute prefixes with the latter property and represent a prefix u by its Parikh image $(|u|_a)_{a\in\Sigma}$. Hence, we say a sequence $O = (O_a)_{a\in\Sigma} \in \mathbb{N}^\Sigma$ is an *occurrence* of a trace v in a trace w iff there is a prefix $u \preceq w$ such that $uv \preceq w$, and $O = (|u|_a)_{a\in\Sigma}$. Note that our definition of an occurrence of $\mathbb{P}$ in $\mathbb{T}$ does not exactly correspond to the intuitive notion of an occurrence as a convex subset of the dependence graph of $[\mathsf{val}(\mathbb{T})]_I$. In fact, to a convex subset of the dependence graph of $[\mathsf{val}(\mathbb{T})]_I$, which is isomorphic to the dependence graph of $[\mathsf{val}(\mathbb{P})]_I$, there might correspond several occurrences O, since for an $a \in \Sigma$ that is independent of $\mathsf{alph}(\mathbb{P})$ we might have several possibilities for the value O_a. However, if we restrict to letters that are dependent on $\mathsf{alph}(\mathbb{P})$, then our definition of an occurrence coincides with the intuitive notion. For $\Gamma \subseteq \Sigma$ we write $\pi_\Gamma(O)$ for the restriction $(O_a)_{a\in\Gamma}$. Furthermore, we say that O is an occurrence of $\mathbb{P}$ in $\mathbb{T}$ if O is an occurrence of $[\mathsf{val}(\mathbb{P})]_I$ in $[\mathsf{val}(\mathbb{T})]_I$.

Let X be a nonterminal of $\mathbb{T}$ with production $X \to YZ$ and let O be an occurrence of $[\mathsf{val}(\mathbb{P})]_I$ in $[\mathsf{val}(X)]_I$. If there are $a, b \in \mathsf{alph}(\mathbb{P})$ with $O_a < |\mathsf{val}(Y)|_a$ and $O_b + |\mathsf{val}(\mathbb{P})|_b > |\mathsf{val}(Y)|_b$, then we say that O is an occurrence of $\mathbb{P}$ *at the cut* of X. Assume w.l.o.g. that $|\mathsf{val}(\mathbb{P})| \geq 2$, otherwise we simply have to check whether a certain letter occurs in $\mathsf{val}(\mathbb{T})$. By this assumption, $[\mathsf{val}(\mathbb{P})]_I$ is a factor of $[\mathsf{val}(\mathbb{T})]_I$ iff there is a nonterminal X of $\mathbb{T}$ for which there is an occurrence of $\mathbb{P}$ at the cut of X.

Lemma 15 ([12]). *Let $v, w \in \mathbb{M}(\Sigma, I)$. A sequence $(n_a)_{a\in\Sigma} \in \mathbb{N}^\Sigma$ is an occurrence of v in w iff (n_a, n_b) is an occurrence of $\pi_{\{a,b\}}(v)$ in $\pi_{\{a,b\}}(w)$ for all $(a, b) \in D$.*

An *arithmetic progression* is a subset of $\mathbb{N}^\Sigma$ of the form $\{(i_a)_{a\in\Sigma} + k \cdot (d_a)_{a\in\Sigma} \mid 0 \leq k \leq \ell\}$. This set can be represented by the triple $((i_a)_{a\in\Sigma}, (d_a)_{a\in\Sigma}, \ell)$. The *descriptional size* $|((i_a)_{a\in\Sigma}, (d_a)_{a\in\Sigma}, \ell)|$ of the arithmetic progression $((i_a)_{a\in\Sigma}, (d_a)_{a\in\Sigma}, \ell)$ is $\log_2(\ell) + \sum_{a\in\Sigma}(\log_2(i_a) + \log_2(d_a))$. We will use Lemma 15 in order to compute the occurrences of $\mathbb{P}$ in $\mathbb{T}$ in form of a family of arithmetic progressions. To this aim, we follow a similar approach as Genest and Muscholl for message sequence charts [7]. In particular Lemma 16 below was inspired by [7, Prop. 1]. For the rest of this section we make the following assumption:

$$\mathsf{alph}(\mathbb{P}) = \mathsf{alph}(\mathbb{T}) = \Sigma \text{ is connected.} \tag{1}$$

Whereas $\mathsf{alph}(\mathbb{P}) = \mathsf{alph}(\mathbb{T})$ is a real restriction, the assumption that $\Sigma = \mathsf{alph}(\mathbb{T})$ is connected is not a real restriction; we simply solve several pattern matching instances if it is not satisfied. Let X be a nonterminal of $\mathbb{T}$ and let O be an occurrence of $\mathbb{P}$ at the cut of X. Since the pattern is connected there must be some $(a, b) \in D$ such that $\pi_{\{a,b\}}(O)$ is at the cut of $X^{\{a,b\}}$. We will therefore compute occurrences of $\pi_{\{a,b\}}(\mathsf{val}(\mathbb{P}))$ at the cut of $X^{\{a,b\}}$. It is well known that the occurrences of $\pi_{\{a,b\}}(\mathsf{val}(\mathbb{P}))$ at the cut of $X^{\{a,b\}}$ form an arithmetic progression $((i_a, i_b), (d_a, d_b), \ell)$ and that $\pi_{\{a,b\}}(\mathsf{val}(\mathbb{P}))$ is of the form $u^n v$ for some $n \geq \ell$ and strings $u, v \in \{a, b\}^*$ with $v \preceq u$, $|u|_a = d_a$ and $|u|_b = d_b$. Moreover, the arithmetic progression $((i_a, i_b), (d_a, d_b), \ell)$ can be computed in time $|\mathbb{T}|^2|\mathbb{P}|$ (see [11][1]). Now suppose we have computed the occurrences of $\pi_{\{a,b\}}(\mathsf{val}(\mathbb{P}))$ at the cut of $X^{\{a,b\}}$ in form of an arithmetic progression. The problem now is how to find (for the possibly exponentially many occurrences in the arithmetic progression) matching occurrences of projections onto all other pairs in D. The following lemma states that either there is a pair $(a, b) \in D$ such that the projection onto $\{a, b\}$ is the first or the last element of an arithmetic progression, or all projections lie at the cut of the same nonterminal.

Lemma 16. *Let X be a nonterminal of $\mathbb{T}$ and let O be an occurrence of $\mathbb{P}$ at the cut of X. Then either (i) $\pi_{\{a,b\}}(O)$ is at the cut of $X^{\{a,b\}}$ for all $(a, b) \in D$ with $a \neq b$, or (ii) there are $(a, b) \in D$ such that $\pi_{\{a,b\}}(O)$ is the first or last element of the arithmetic progression of occurrences of $\pi_{\{a,b\}}(\mathsf{val}(\mathbb{P}))$ at the cut of $X^{\{a,b\}}$.*

Lemma 16 motivates that we partition the set of occurrences into two sets. Let O be an occurrence of $\mathbb{P}$ in $\mathbb{T}$ at the cut of X. We call O *single* (for X) if there is $(a, b) \in D$ such that the projection $\pi_{\{a,b\}}(O)$ is the first or the last element of the arithmetic progression of occurrences of $\pi_{\{a,b\}}(\mathsf{val}(\mathbb{P}))$ at the cut of $X^{\{a,b\}}$. Otherwise, we call O *periodic*

[1] In fact, in [11] it was shown that the arithmetic progression $(i_a{+}i_b, d_a{+}d_b, \ell)$ can be computed in polynomial time. From this the arithmetic progression, $((i_a, i_b), (d_a, d_b), \ell)$ can easily be computed.

(for X). By Lemma 16, if O is periodic, then $\pi_{\{a,b\}}(O)$ is an element of the arithmetic progression of occurrences of $\mathsf{val}(\mathbb{P}^{\{a,b\}})$ at the cut of $X^{\{a,b\}}$ for all $(a,b) \in D$ (but neither the first nor the last element). Prop. 17 below shows that we can decide in polynomial time whether there are single occurrences of $\mathbb{P}$ in $\mathbb{T}$. The basic idea is that due to assumption (1), an occurrence of $\mathsf{val}(\mathbb{P})$ in $\mathsf{val}(\mathbb{T})$ is completely determined as soon as we have determined the position of a single node of the dependence graph of $[\mathsf{val}(\mathbb{P})]_I$ in the dependence graph of $[\mathsf{val}(\mathbb{T})]_I$.

Proposition 17. *Given $(a,b) \in D$, a nonterminal X of $\mathbb{T}$ and an occurrence (O_a, O_b) of $\pi_{\{a,b\}}(\mathsf{val}(\mathbb{P}))$ at the cut of $X^{\{a,b\}}$, one can decide in time $(|\mathbb{T}| + |\mathbb{P}|)^{O(1)}$ whether this occurrence is a projection of an occurrence of $\mathbb{P}$ at the cut of X.*

It remains to show that for every nonterminal X of $\mathbb{T}$ we can compute in polynomial time the periodic occurrences. To this aim we define the amalgamation of arithmetic progressions. Let $\Gamma, \Gamma' \subseteq \Sigma$ with $\Gamma \cap \Gamma' \neq \emptyset$. Consider two arithmetic progressions $p = ((i_a)_{a\in\Gamma}, (d_a)_{a\in\Gamma}, \ell)$ and $p' = ((i'_a)_{a\in\Gamma'}, (d'_a)_{a\in\Gamma'}, \ell')$. The *amalgamation* of p and p' is $p \otimes p' = \{v = (v_a)_{a\in\Gamma\cup\Gamma'} \mid \pi_\Gamma(v) \in p \text{ and } \pi_{\Gamma'}(v) \in p'\}$. The following lemma follows from elementary facts about simultaneous congruences:

Lemma 18. *Let $\Gamma, \Gamma' \subseteq \Sigma$ with $\Gamma \cap \Gamma' \neq \emptyset$, and let $p = ((i_a)_{a\in\Gamma}, (d_a)_{a\in\Gamma}, \ell)$ and $p' = ((i'_a)_{a\in\Gamma'}, (d'_a)_{a\in\Gamma'}, \ell')$ be two arithmetic progressions. Then $p \otimes p'$ is an arithmetic progression which can be computed in time $(|p| + |p'|)^{O(1)}$.*

The next proposition can be shown using Lemma 15 and 18.

Proposition 19. *Let X be a nonterminal of $\mathbb{T}$. The periodic occurrences of $\mathbb{P}$ at the cut of X form an arithmetic progression which can be computed in time $(|\mathbb{T}| + |\mathbb{P}|)^{O(1)}$.*

We now get the following theorem easily.

Theorem 20. *Given an independence alphabet (Σ, I), and two SLPs $\mathbb{P}$ and $\mathbb{T}$ over Σ such that $\mathsf{alph}(\mathbb{P}) = \mathsf{alph}(\mathbb{T})$, we can decide in polynomial time whether $[\mathsf{val}(\mathbb{P})]_I$ is a factor of $[\mathsf{val}(\mathbb{T})]_I$.*

Proof. Let X be a nonterminal of $\mathbb{T}$. Using [11] we compute for each pair $(a,b) \in D$ the arithmetic progression of occurrences of $\pi_{a,b}(\mathsf{val}(\mathbb{P}))$ at the cut of $X^{\{a,b\}}$. By applying Prop. 17 to the first and to the last elements of each of these arithmetic progressions, we compute in polynomial time the single occurrences at the cut of X. The periodic occurrences can be computed in polynomial time using Prop. 19. The result follows, since $[\mathsf{val}(\mathbb{P})]_I$ is a factor of $[\mathsf{val}(\mathbb{T})]_I$ iff there is a nonterminal X of $\mathbb{T}$ for which there is a single occurrence of $\mathbb{P}$ at the cut of X or a periodic occurrence of $\mathbb{P}$ at the cut of X. $\square$

In [9], a slight generalization of Theorem 20 is shown.

7 Compressed Conjugacy

In order to prove Thm. 3 we need some further concepts from [20]. If for a trace x we have $\mathsf{NF}_R(x) = uyu^{-1}$ in $\mathbb{M}(\Sigma^{\pm1}, I)$ for traces y, u such that $\min(y) \cap \min(y^{-1}) = \emptyset$, then we call y the *core* of x, $\mathsf{core}(x)$ for short; it is uniquely defined [20]. Note that a trace t is a double a-cone if and only if $t \in \mathsf{IRR}(R)$ and $\mathsf{core}(t) = a$. The following result, which follows by combining results from [12] and [21], allows us to transfer the conjugacy problem in $\mathbb{G}(\Sigma, I)$ to a problem on traces:

Theorem 21 ([12,21]). *Let $u, v \in \mathbb{M}(\Sigma^{\pm 1}, I)$. Then u is conjugated to v in $\mathbb{G}(\Sigma, I)$ if and only if: (i) $|\mathsf{core}(u)|_a = |\mathsf{core}(v)|_a$ for all $a \in \Sigma^{\pm 1}$ and (ii) $\mathsf{core}(u)$ is a factor of $\mathsf{core}(v)^{2|\Sigma|}$.*

In order to apply Thm. 21 to SLP-compressed traces, we need a polynomial time algorithm for computing an SLP that represents $\mathsf{core}([\mathsf{val}(\mathbb{A})]_I)$ for a given SLP $\mathbb{A}$. The following lemma is crucial:

Lemma 22. *Let $x \in \mathsf{IRR}(R)$ and $d = x \sqcap x^{-1}$. Then $\mathsf{NF}_R(d^{-1}xd) = \mathsf{core}(x)$.*

Example 23. We take the independence alphabet from Example 5 and consider the trace $x = [c^{-1}d^{-1}a^{-1}ba^{-1}cabdc^{-1}d^{-1}a^{-1}b^{-1}dca]_I \in \mathbb{M}(\Sigma^{\pm 1}, I)$, whose dependence graph looks as follows:

$$\begin{tikzcd}
c^{-1} \arrow[rr] \arrow[dr] & & d^{-1} \arrow[r] & c \arrow[r] \arrow[dr] & d \arrow[r] & c^{-1} \arrow[r] \arrow[dr] & d^{-1} \arrow[r] \arrow[r, bend left, dashed, no head] & d \arrow[r] & c \\
a^{-1} \arrow[r] & b \arrow[r] \arrow[urr] & a^{-1} \arrow[r] \arrow[r, bend right, dashed, no head] & a \arrow[r] & b \arrow[r] \arrow[ur] & a^{-1} \arrow[r] & b^{-1} \arrow[rr] \arrow[urr] & & a
\end{tikzcd}$$

We have $\mathsf{NF}_R(x) = [c^{-1}d^{-1}a^{-1}bcbdc^{-1}a^{-1}b^{-1}ca]_I$:

$$\begin{tikzcd}
c^{-1} \arrow[rr] \arrow[dr] & & d^{-1} \arrow[r] & c \arrow[r] \arrow[dr] & d \arrow[r] & c^{-1} \arrow[dr] & & c \\
a^{-1} \arrow[r] & b \arrow[urr] & & & b \arrow[r] \arrow[ur] & a^{-1} \arrow[r] & b^{-1} \arrow[r] \arrow[ur] & a
\end{tikzcd}$$

Hence, the core of x is $\mathsf{core}(x) = [d^{-1}cbdc^{-1}a^{-1}]_I$ (the middle part in the above diagram). Note that we have $\mathsf{NF}_R(x) \sqcap \mathsf{NF}_R(x^{-1}) = c^{-1}a^{-1}b$. This trace occurs as a prefix of $\mathsf{NF}_R(x)$ and its inverse occurs as a suffix of $\mathsf{NF}_R(x)$. By cyclically cancelling $c^{-1}a^{-1}b$ in $\mathsf{NF}_R(x)$, we obtain $d^{-1}cbdc^{-1}a^{-1} = \mathsf{core}(x)$.

Thm. 7 and 8 and Lemma 22 imply:

Corollary 24. *Fix an independence alphabet $(\Sigma^{\pm 1}, I)$. Then, for a given SLP $\mathbb{A}$ over the alphabet $\Sigma^{\pm 1}$ one can compute in polynomial time an SLP $\mathbb{B}$ with $[\mathsf{val}(\mathbb{B})]_I = \mathsf{core}([\mathsf{val}(\mathbb{A})]_I)$.*

We can now infer Thm. 3: Let $\mathbb{A}$ and $\mathbb{B}$ be two given SLPs over $\Sigma^{\pm 1}$. We want to check, whether $\mathsf{val}(\mathbb{A})$ and $\mathsf{val}(\mathbb{B})$ represent conjugated elements of the graph group $\mathbb{G}(\Sigma, I)$. Using Cor. 24, we can compute in polynomial time SLPs $\mathbb{C}$ and $\mathbb{D}$ with $[\mathsf{val}(\mathbb{C})]_I = \mathsf{core}([\mathsf{val}(\mathbb{A})]_I)$ and $[\mathsf{val}(\mathbb{D})]_I = \mathsf{core}([\mathsf{val}(\mathbb{B})]_I)$. By Thm. 21, it suffices to check, whether (i) $|\mathsf{core}([\mathsf{val}(\mathbb{C})]_I)|_a = |\mathsf{core}([\mathsf{val}(\mathbb{D})]_I)|_a$ for all $a \in \Sigma^{\pm 1}$ and (ii) whether $\mathsf{core}([\mathsf{val}(\mathbb{C})]_I)$ is a factor of $\mathsf{core}([\mathsf{val}(\mathbb{D})]_I)^{2|\Sigma|}$. Condition (i) can be easily checked in polynomial time, since the number of occurrences of a symbol in a compressed string can be computed in polynomial time. Moreover, condition (ii) can be checked in polynomial time by Thm. 20, since (by condition (i)) we can assume that $\mathsf{alph}(\mathsf{val}(\mathbb{C})) = \mathsf{alph}(\mathsf{val}(\mathbb{D}))$. $\square$

8 Open Problems

Though we have shown that some cases of the simultaneous compressed conjugacy problem for graph groups (see Sec. 3) can be decided in polynomial time, it remains unclear whether this holds also for the general case. It is also unclear to the authors, whether the general compressed pattern matching problem for traces, where we drop restriction $\mathsf{alph}(\mathbb{T}) = \mathsf{alph}(\mathbb{P})$, can be decided in polynomial time. Finally, it is not clear, whether Thm. 1–3 also hold if the independence alphabet is part of the input.

References

1. Charney, R.: An introduction to right-angled Artin groups. Geometriae Dedicata 125, 141–158 (2007)
2. Cori, R., Métivier, Y., Zielonka, W.: Asynchronous mappings and asynchronous cellular automata. Information and Computation 106(2), 159–202 (1993)
3. Crisp, J., Godelle, E., Wiest, B.: The conjugacy problem in right-angled Artin groups and their subgroups. Journal of Topology 2(3) (2009)
4. Day, M.B.: Peak reduction and finite presentations for automorphism groups of right-angled Artin groups. Geometry & Topology 13(2), 817–855 (2009)
5. Diekert, V.: Combinatorics on Traces. LNCS, vol. 454. Springer, Heidelberg (1990)
6. Gasieniec, L., Karpinski, M., Plandowski, W., Rytter, W.: Efficient algorithms for Lempel-Ziv encoding. In: Karlsson, R., Lingas, A. (eds.) SWAT 1996. LNCS, vol. 1097, pp. 392–403. Springer, Heidelberg (1996)
7. Genest, B., Muscholl, A.: Pattern matching and membership for hierarchical message sequence charts. Theory of Computing Systems 42(4), 536–567 (2008)
8. Hagenah, C.: Gleichungen mit regulären Randbedingungen über freien Gruppen. PhD thesis, University of Stuttgart, Institut für Informatik (2000)
9. Haubold, N., Lohrey, M., Mathissen, C.: Compressed conjugacy and the word problem for outer automorphism groups of graph groups (2010), `http://arxiv.org/abs/1003.1233`
10. Laurence, M.R.: A generating set for the automorphism group of a graph group. Journal of the London Mathematical Society. Second Series 52(2), 318–334 (1995)
11. Lifshits, Y.: Processing compressed texts: A tractability border. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 228–240. Springer, Heidelberg (2007)
12. Liu, H.-N., Wrathall, C., Zeger, K.: Efficient solution to some problems in free partially commutative monoids. Information and Computation 89(2), 180–198 (1990)
13. Lohrey, M., Schleimer, S.: Efficient computation in groups via compression. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) CSR 2007. LNCS, vol. 4649, pp. 249–258. Springer, Heidelberg (2007)
14. Lyndon, R.C., Schupp, P.E.: Combinatorial Group Theory. Springer, Heidelberg (1977)
15. Macdonald, J.: Compressed words and automorphisms in fully residually free groups. International Journal of Algebra and Computation (2009) (to appear)
16. Myasnikov, A., Shpilrain, V., Ushakov, A.: Group-based Cryptography. Birkhäuser, Basel (2008)
17. Plandowski, W.: Testing equivalence of morphisms on context-free languages. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 460–470. Springer, Heidelberg (1994)

18. Schleimer, S.: Polynomial-time word problems. Commentarii Mathematici Helvetici 83(4), 741–765 (2008)
19. Servatius, H.: Automorphisms of graph groups. Journal of Algebra 126(1), 34–60 (1989)
20. Wrathall, C.: The word problem for free partially commutative groups. Journal of Symbolic Computation 6(1), 99–104 (1988)
21. Wrathall, C.: Free partially commutative groups. In: Combinatorics, Computing and Complexity, pp. 195–216. Kluwer Academic Press, Dordrecht (1989)

Using Light to Implement Parallel Boolean Algebra

Tom Head[*]

Mathematical Sciences
Binghamton University
Binghamton, New York 13902-6000 USA
tom@math.binghamton.edu

Abstract. We design and implement highly parallel algorithms that use light as the tool of computation. An ordinary xerox machine and a box of transparencies constitutes our computer. We find the maximum in a list of n-bit numbers of arbitrary length using at most n xerox copying steps. We decide, for any graph having n vertices and m edges, whether a 3-coloring exists in at most $2n + 4m$ copying steps. For large instances of problems such as the 3-color problem, this solution method may require the production of transparencies that display challengingly high densities of information. Our ultimate purpose here is to give hand tested 'ultra-parallel' algorithmic procedures that may provide useful suggestions for future optical technologies.

Keywords: Unconventional computing, photo-computing, light-based computing, optical computing, xerography.

1 Introduction

We show how to design and implement by hand highly parallel algorithms that use light as the central tool for computing. An ordinary xerox machine accompanied with a box of plastic transparencies constitutes the hardware of our computer. Each computation involves a sequence of xerox operations in each of which a new transparency is formed from a previous transparency that may be overlaid with other transparencies or with masking rectangles. We have carried out by hand on a standard xerox machine all the operations we discuss and any reader can do the same. *We hasten to recognize that there is a practical limit to the size of problem instances that can be treated with the technology we have used thus far. This limitation is discussed in Section 6.* Nevertheless we hope that our work may contribute to some future light based technology.

We represent binary information using the contrast between opaque and transparent squares on a plastic transparency. Each opaque square will be represented

[*] I thank Hiroki Sayama and Lucas Sabalka for suggestions which allowed me to convert my originally awkward version of Section 4 to its current greatly improved form. I thank each of the three anonymous referees for suggesting corrections and improvements that have been made.

as ■ and each transparent square will be represented as a blank square space which we make visible by writing ␣. In Section 2 our first elementary illustrative algorithm will deal with 6-bit binary numbers and in this case the bit 1 is represented as ␣ and the bit 0 as ■. Thus in this setting the number 101001 is represented as ␣■␣■■␣. Our computations will be carried out by overlaying fixed length columns of symbols ■ and ␣ and will use the fact that any stack of such symbols is transparent if and only if every symbol in the stack is transparent. We define a complementation operation, $'$, by: ■$'$ = ␣ and ␣$'$ = ■. Standard xerox machines make negative copies as well as positive copies. Consequently the operation $'$ comes already implemented on xerox machines.

In a logical context with TRUE encoded as ␣ and FALSE as ■, overlaying symbols computes Boolean AND. Moreover complementation, $'$, gives the Boolean NOT. We regard Boolean Algebra as at the core of mathematical and computational thought. Consequently we regard all of our efforts in computing with xerography as (thinly disguised) Boolean Algebra. It is parallel (or vector) Boolean Algebra because many operations are carried out simultaneously as we work with columns of Boolean values.

In previous publications [3,4,5,1] we have defined and illustrated our techniques using xerography to treat three standard algorithmic problems: the satisfiability of sets of disjunctive Boolean clauses, the determination and study of the independent subsets of an undirected graph, and the determination and study of the subcovers of a set covering. The complexity of our algorithmic procedures can be measured by the number of xerox operations they require. The number of steps required in our algorithms grows only linearly or at most quadratically in the length of the presentation required for each instance. Since the standard computation of addition is accomplished using symmetric difference, Boolean AND, and left shift, the usual addition algorithm is immediately implementable with xerography - but with no reduction of complexity. The only possible advantage in adding xerographicly is that an arbitrary number of such additions can be performed simultaneously with no additional steps required.

We show in Section 2, as a simple 'warm-up' exercise, how to *determine the largest number in a list of n-bit binary numbers using at most n xerox operations.* This implements with light the classical concept of content addressable memory. Our major effort here is the presentation in Sections 3, 4, 5 of a method of deciding the 3-colorability [2] of an arbitrary undirected graph. We illustrate in complete detail the solution of a specific instance for the 3-coloring of a small graph. *Deciding the 3-colorability of a graph having n vertices and m edges requires at most $2n + 4m$ xerox operations.* The key tool for our solution is the creation in Section 4 of a potentially general purpose procedure that uses only three xerox operations in the construction of a pointer to those rows of two congruent rectangular displays which are identical.

Other interesting schemes for using light as a computational tool have been introduced in [10] and [9]. Light has also been used in [8]. See [7] and consult www.ece.arizona.edu/∼ocppl/ for optical computation. Our approach to computing with light arose as a transformation of our approach to biomolecular computing using DNA molecules in aqueous solution [6].

2 Finding the Largest Number in a List

This Section must be read with constant reference to Figure 1 which displays in its leftmost six columns a list of 36 non-negative 6-bit integers. In this table the bit 0 is represented as an opaque square ■ and the bit 1 as the transparent square _ . Thus, for example, the first row displays the number 010011. As the simplest illustration of our xerographic computing we solve the problem of finding the maximum number in the given list of 6-bit integers.

We understand that the table of 36 integers is given on a single transparency having dimensions appropriate for use on our xerox machine. Here are the steps of the computation:

1. We mask all but column 1 and ask whether any light passes through column 1. In the present case we answer 'Yes'. This tells us that the *first* bit in the maximum number is a 1. Since the answer is Yes, we make a xerox copy 1^x of column 1 after masking all columns other than column 1.
2. Place column 1^x over column 2 of the table of numbers. We indicate the column created by laying 1^x over 2 as $1^x/2$ When all other columns are masked, does light show through $1^x/2$? Yes. This tells us that the *second* bit of the maximum number is 1. Make a xerox copy 2^x of $1^x/2$.
3. Place column 2^x over column 3, producing $2^x/3$. Light through $2^x/3$? No. This tells us that no number in the table that has 1 as each of its first two bits has a 1 as its third bit. Consequently the third bit in the maximum number is 0. No xerox copy of $2^x/3$ is made.
4. Place column 2^x over column 4, producing $2^x/4$. Light through $2^x/4$? Yes. The fourth bit of the maximum is therefore 1. Make a xerox copy 4^x of $2^x/4$.
5. Place 4^x over 5, producing $4^x/5$. Light through $4^x/5$? No. The fifth bit is 0. No xerox is made.
6. Place 4^x over 6, producing $4^x/6$. Light through $4^x/6$? Yes. The sixth bit is 1. *We have accumulated the bits of the maximum entry, which is* 110101. If a permanent record of the final result of the computation is desired, make a xerox copy 6^x of $4^x/6$. When 6^x is placed adjacent to the original table it constitutes a 'pointer' to the occurrences of the maximum values in the table.

If in the first step of the procedure above, light had not passed through column 1 then we would have concluded that the first bit of the maximum is 0 and we would not have made a xerox copy of column 1. Instead we would have tested for light through column 2 and proceeded with column 2 playing the role previously played by column 1.

If one wishes to find the *minimum* number in a list, De Morgan's Law tells us that this can be done with two extra xerox operations: Make a negative xerox copy of the original list. Find the maximum of this second list. Form a negative

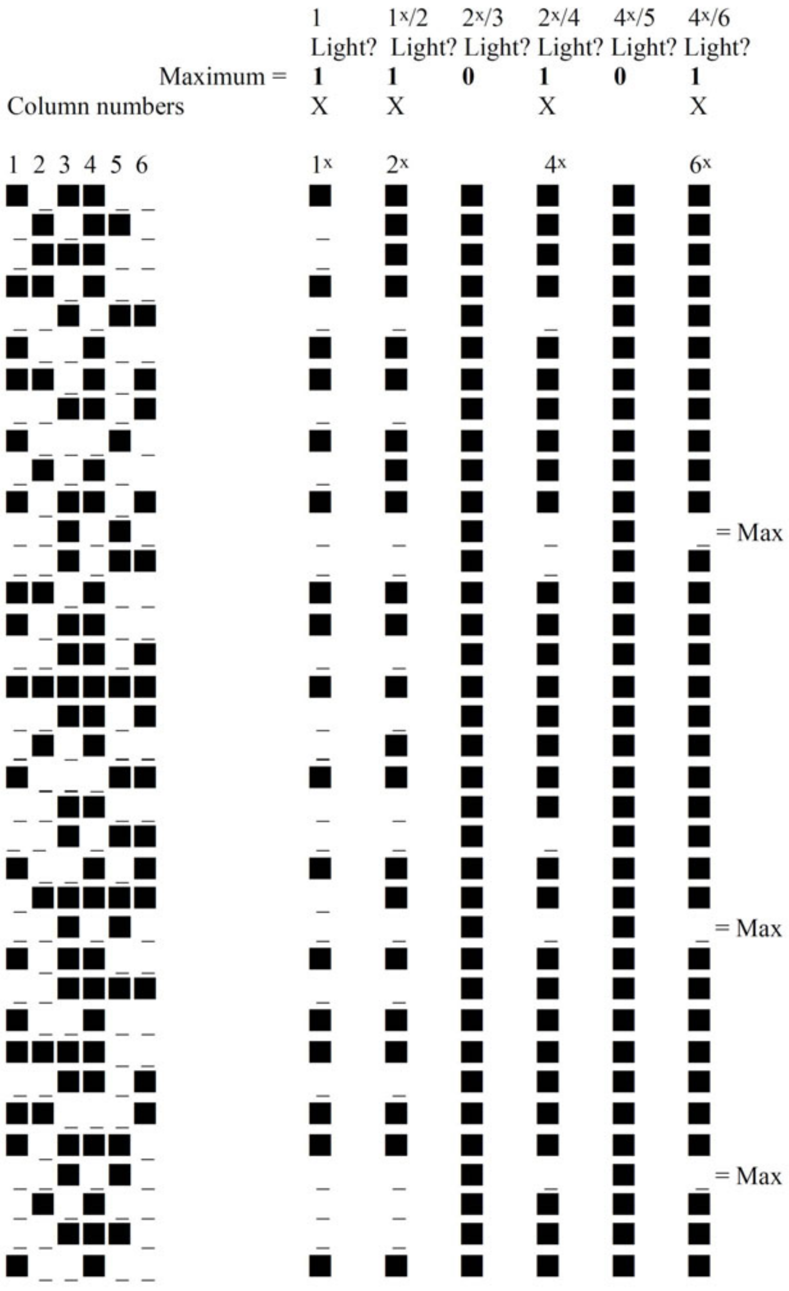


Fig. 1. (As Explained in Section 2) Finding the Maximum in a list of non-negative integers and the locations in the list where it occurs

xerox copy of the maximum of the second to obtain the minimum of the original list. It seems amusing that we can find the maximum and the minimum of the list without even reading the list.

In any future instrumental implementation the decision as to whether a column does or does not admit light might made by replacing the human eye with a photon detector. The appropriately masked transparency, exposing only the column of interest, could be tested by laying it over any glowing panel of light. A xerox machine will provide an adequate source of light: Leave the cover off and punch the control to produce one sweep of light that will activate/not-activate the photon detector appropriately. If this is done then each light test can be counted as equivalent to making a xerox copy.If light steps are counted as xerox operations then finding the maximum would have complexity $2n$ rather than n.

We operate only with columns and **never with rows**! In the problems it is our purpose to treat we expect to deal with rectangular tables with relatively few columns but with vastly many rows. Commonly we deal with tables having n columns and 2^n, 3^n, or more rows. Our purpose is to provide solutions that require only $O(n)$ or $O(n^2)$ steps as compared to the $O(2^n)$ steps in known algorithms. Consequently any inspection of a substantial percentage of the rows would push our solutions into $O(2^n)$ steps defeating the purpose of our procedures.

In this Section we have assumed that a rectangular array is given to us initially. In the problem we are about to treat we must begin by constructing the initial array.

3 Deciding the 3-Colorability of an Undirected Graph

The ideal published report of our computations would include plastic pages that are the actual transparencies used in the computation process. Since this is not practical, we have simulated in Section 2 on paper the appearance of such transparencies. When thinking with paper simulations there is one point that must be kept in mind: On paper white squares can obscure underlying black squares but (happily) this is not possible with transparencies. We make one more simplification in our representations: Once one has a clear mental vision of the actual transparencies being used, it takes less space to 'encode' the transparencies without the cumbersome representations ■ and _ . For the remainder of this article we will allow ourselves to indicate the presence of an opaque rectangle ■ by writing the bit **1** and the presence of a transparent rectangle _ by writing the bit **0**. Thus the display: ■ _ _ _ ■ _ ■ _ _ ■ will be now communicated by writing: **1000101001**. The reader is requested to 'see' in his/her mind an opaque square when reading a **1** and a transparent square when reading a **0**.

Recall that the classical k-coloring problem for a given graph is a decision problem with a yes/no answer [2]. The question is: Given a finite graph and a finite set of k colors, can the vertices of the graph be colored in such a way that the two endpoints of each edge in the graph have distinct colors? A more intricate problem is to specify either one (or all) explicit coloring(s) that meet(s) this condition. We will illustrate xerographic procedures for deciding 3-colorability

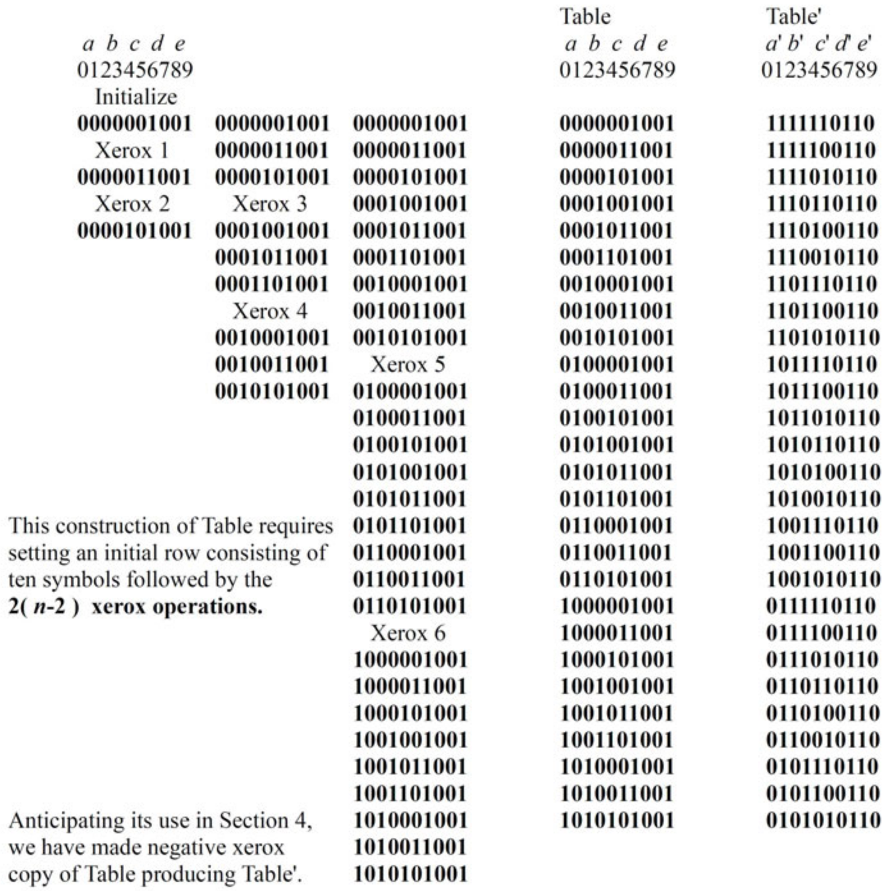


			Table	Table'
a b c d e			*a b c d e*	*a' b' c' d' e'*
0123456789			0123456789	0123456789
Initialize				
0000001001	0000001001	0000001001	0000001001	1111110110
Xerox 1	0000011001	0000011001	0000011001	1111100110
0000011001	0000101001	0000101001	0000101001	1111010110
Xerox 2	Xerox 3	0001001001	0001001001	1110110110
0000101001	0001001001	0001011001	0001011001	1110100110
	0001011001	0001101001	0001101001	1110010110
	0001101001	0010001001	0010001001	1101110110
	Xerox 4	0010011001	0010011001	1101100110
	0010001001	0010101001	0010101001	1101010110
	0010011001	Xerox 5	0100001001	1011110110
	0010101001	0100001001	0100011001	1011100110
		0100011001	0100101001	1011010110
		0100101001	0101001001	1010110110
		0101001001	0101011001	1010100110
		0101011001	0101101001	1010010110
This construction of Table requires		0101101001	0110001001	1001110110
setting an initial row consisting of		0110001001	0110011001	1001100110
ten symbols followed by the		0110011001	0110101001	1001010110
2(*n*-2) xerox operations.		0110101001	1000001001	0111110110
		Xerox 6	1000011001	0111100110
		1000001001	1000101001	0111010110
		1000011001	1001001001	0110110110
		1000101001	1001011001	0110100110
		1001001001	1001101001	0110010110
		1001011001	1010001001	0101110110
		1001101001	1010011001	0101100110
Anticipating its use in Section 4,		1010001001	1010101001	0101010110
we have made negative xerox		1010011001		
copy of Table producing Table'.		1010101001		

Fig. 2. (As Explained in Section 3) The construction of the table of all potential 3-colorings of a graph having vertices $\{a, b, c, d, e\}$. The colors are red (encoded as 00), yellow (encoded as 01) and blue (encoded as 10).

of a simple graph G having 5 vertices $\{a, b, c, d, e\}$ and 6 edges: $\{a, b\}$, $\{a, c\}$, $\{a, e\}$, $\{b, c\}$, $\{c, d\}$ and $\{d, e\}$.

This Section is devoted to the construction of an appropriate list of potential colorings. It must be read with constant reference to Figure 2. We shorten the required list we construct by choosing one edge, in this case $\{d, e\}$, and assigning its endpoints distinct colors. In Section 5 the remaining edges are attended to and the solution is completed. The reader who has understood our treatment of this instance of the 3-color problem will be able to make the very small adjustments needed to solve k-coloring problems.

Let $K =$ red, yellow, blue be our set of colors. We choose a binary encoding for K: red $=$ **00**, yellow $=$ **01**, blue $=$ **10**. We observe first that there is at least one edge, for example: $\{d, e\}$. We construct the table, given in Figure 2, of all those

$3^{(5-2)} = 27$ colorings having d blue and e yellow. One such coloring is: a red, b red, c red, d blue, e yellow which we encode as the binary word **0000001001** where we will understand that the first two bits encode the color for a, the second two bits the color for b, and so on. We construct a table of all such possibilities using only $2(5 - 2) = 6$ xerox operations following an initialization of the first word to **0000001001**.

Let a rectangle of width w and height h be prescribed on a transparency appropriate for reading by the xerox machine to be used. In this w by h rectangle we will construct a table that is 10 bits wide and $3^{(5-2)} = 27$ bits high. As the initialization step we reserve a top row of width $10(1/27)h$ and height $(1/27)h$. We then initialize this top row to contain the ten bits **0000001001** expressed on the actual transparency, of course, with the bits represented as ■ and _ . This gives the first row of the table of colorings in Figure 2. We obtain the second row by masking the bit in position 5 of the first row and making a xerox copy of the result. We obtain the third row by masking the bit in position 4 of the first row and making a xerox copy of the result. We have now created the leftmost column of Fig.2. The two xeroxes are placed under the initial row as displayed in the second column of Fig. 2. We now treat this three row block in a manner parallel to the way we treated row 1. Thus we mask bit 3 of this three row block and xerox the result. We then mask bit 2 of the three row block and xerox the result. We have now created the second column of Fig. 2. The two xeroxes are placed under the initial three row block as displayed in the third column of Fig. 2. We now treat the resulting nine row block as we did the three row block. We make two xeroxes of the nine row block, one with bit 1 masked and one with bit 0 masked. This gives column 3 of Fig. 2. The fourth column of Fig. 2 is formed by placing the two new xeroxes under the the first nine row block giving the desired 27 row Table. This table can now be used to decide 3-colorability of any graph having five nodes that contains at least one edge.

We have also provided, as the fifth column of Fig. 2, Table′ which was obtained from Table by taking one *negative* xerox copy of Table. Table and Table′ are equally fundamental tools for use in Section 5. To complete the solution of the 3-color problem we need the ‘subroutine’ provided in the next Section.

Note that in making constructions such as the Table of Fig. 2, it is possible to avoid creating a very long display that is of comparatively tiny width. This can be done by laying newly constructed segments to the side, rather than below the previously constructed portion of the display. We have previously explained this point and the slight complications it creates in the context of our discussions of Boolean satisfiability (Head, 2007, 2009).

4 Pointing to Pairs of Identical Words in Parallel Tables

For a graph that has $\{a, b\}$ as an edge, no acceptable 3-coloring of the graph can have a and b both red, both yellow, or both blue. Thus we need a procedure that works only with the columns of Table and marks as unacceptable all those rows of Table that have as initial four bits either **0000**, **0101**, or **1010**. We give a very general procedure that requires only three xerox operations to mark all such rows. We present this procedure and the confirmation of its validity by listing all 16 of the 4-bit words and verifying that for every 4-bit word the equality/inequality of its first bit pair with its second bit pair is correctly determined. We discuss this procedure by referring to Figure 3. The leftmost column pair in Fig. 3 gives all 16 pairs of two bit words in columns headed as a and b. Suppose this 16 line table lies on a transparency. Here is the procedure for constructing a 16 bit column which, when laid beside the table, has 0 adjacent to every equal pair and 1 adjacent to every unequal pair. Thus the column we construct 'points to' each of the equal pairs with a 0. Here are the steps of this construction:

1. Make a negative xerox copy of the original four column table headed by a and b. This gives the second four column table in Fig. 3 headed by a' and b'.
2. Lay column a over column b to produce the two column table a/b'. Make a negative xerox copy of a/b' producing the column headed $(a/b')'$.
3. Lay column b over column a' to produce the two column table b/a'. Make a negative xerox copy of b/a' producing the column headed $(b/a')'$.
4. Lay the column headed $(a/b')'$ over the column headed $(b/a')'$ to produce the column headed $(a/b')'/(b/a')'$. We have given separate headings, L and R, to the left and right bits of this column.
5. Make a positive xerox copy of column L. Lay this copy of L over column R to produce the column headed L/R.

Inspect the 16 rows of the column L/R to confirm that it is the desired 'pointer' to the equal pairs.

Note that the construction of the equality pointer has required, once the columns for a' and b' are available, only three xerox operations. In Section 5 we will need an inequality pointer. Consequently we have included the column headed $(L/R)'$. This column is obtained by making a negative xerox copy of L/R. *Thus the construction of the inequality pointer requires only four xerox operations.*

Addendum. The procedure meticulously illustrated in this figure is carried out five times in the completion of the solution of our instance of the 3-color problem given in Section 5 and illustrated in Figure 4. The illustration here justifies our presenting only the results of the five applications of this procedure in Section 5 as shown in Figure 4. However, the validity of this procedure for pairs of n-bit words, for every positive integer n is easily justified: Let u and v be n-bit words having bits expressed as $\blacksquare$ and $\_$. Let u' and v' be the complements of u and v. If u and v are identical, then so are u' and v'. Thus both u/v' and v/u' are entirely opaque. Consequently both $(u/v')'$ and $(v/u')'$ are entirely transparent and so is $(u/v')'/(v/u')'$. Thus when all n bits are 'stacked' into a single stack the result is transparent. On the other hand if, for some k, the k-th bits of u and v differ then the k-th bits of u' and v' also differ. Then the k-th bit of one of u/v' and v/u' is transparent and consequently the k-th bit of one of $(u/v')'$ and $(v/u')'$ is opaque. Thus when all n bits are 'stacked' into a single stack the result is opaque.

Given	Given'		X		X	X		X
a *b*	*a*' *b*'	*a*/ *b*'	(*a*/ *b*')'	*b*/ *a*'	(*b*/ *a*')'	(*a*/ *b*')' / (*b*/ *a*')'	*L*/*R*	(*L*/ *R*)'
						LR	=	≠
00 00	11 11	11	00	11	00	00	0	1
00 01	11 10	10	01	11	00	01	1	0
00 10	11 01	01	10	11	00	10	1	0
00 11	11 00	00	11	11	00	11	1	0
01 00	10 11	11	00	10	01	01	1	0
01 01	10 10	11	00	11	00	00	0	1
01 10	10 01	01	10	10	01	11	1	0
01 11	10 00	01	10	11	00	10	1	0
10 00	01 11	11	00	01	10	10	1	0
10 01	10 10	10	01	11	00	01	1	0
10 10	01 01	11	00	11	00	00	0	1
10 11	01 00	10	01	11	00	01	1	0
11 00	00 11	11	00	00	11	11	1	0
11 01	00 10	11	00	01	10	10	1	0
11 10	00 01	11	00	10	01	01	1	0
11 11	00 00	11	00	11	00	00	0	1

Fig. 3. (As Explained in Section 4) The construction of a pointer (=) to the rows of paired tables in which the entries are identical. The validity of the procedure is confirmed here for all 16 possible pairs of 2-bit words a & b.

5 Completing the 3-Coloring Decision

Can the graph G having vertex set $\{a, b, c, d, e\}$ and edges $\{a, b\}$, $\{a, c\}$, $\{a, e\}$, $\{b, c\}$, $\{c, d\}$ and $\{d, e\}$ be 3-colored? We discuss the completion of this decision by referring to Figure 4. The leftmost column of Fig. 4 lists again the Table of 27 potential 3-colorings (with d blue and e yellow). For ease of making visual confirmations we have listed this same table with spacing that isolates the five bit-pairs in each row. The four-xerox procedure for detecting inequality of pairs is now applied once for each of the edges $\{a, b\}$, $\{a, c\}$, $\{a, e\}$, $\{b, c\}$ and $\{c, d\}$. The edge $\{d, e\}$ need not be treated since we set d and e with distinct colors in the construction of Table. Applying the procedure of Section 4 to $\{a, b\}$ gives the column headed $a \neq b$. The next four columns headed $a \neq c$, $a \neq e$, $b \neq c$ and $c \neq d$ are produced in the same manner. The compound overlay $a \neq b/\ a \neq c/\ a \neq e/\ b \neq c/\ c \neq d$ gives the column P that points to the acceptable 3-colorings by exhibiting a 0 in the row corresponding to each acceptable coloring.

Recall that the classical 3-coloring problem [2] asks for a yes/no answer to the question: Is there at least one 3-coloring. In the context of our xerographic computing procedures this question is equivalent to the question: Does light pass through column P (of Fig. 4)? In this instance we answer 'Yes' to both forms of the question. We are reasonably satisfied with the procedure given here for making this yes/no decision. The more subtle request for a specific acceptable 3-coloring becomes the more subtle question of individualizing a location on the pointer P through which light passes and reading the corresponding color from Table. We are less satisfied with our procedure for providing a specific coloring.

Table	For easy viewing	Treating each edge as in Section 4:						
		4X	4X	4X	4X	4X		////
a b c d e	*a b c d e*	*a*≠*b*	*a*≠*c*	*a*≠*e*	*b*≠*c*	*c*≠*d*		*P*
0000001001	00 00 00 10 01	1	1	0	1	0		1
0000011001	00 00 01 10 01	1	0	0	0	0		1
0000101001	00 00 10 10 01	1	0	0	1	1		1
0001001001	00 01 00 10 01	0	1	0	0	0		1
0001011001	00 01 01 10 01	0	0	0	1	0		1
0001101001	00 01 10 10 01	0	0	0	0	1		1
0010001001	00 10 00 10 01	0	1	0	0	0		1
0010011001	00 10 01 10 01	0	0	0	0	0	*rbyby*	0
0010101001	00 10 10 10 01	0	0	0	1	1		1
0100001001	01 00 00 10 01	0	0	1	1	0		1
0100011001	01 00 01 10 01	0	1	1	0	0		1
0100101001	01 00 10 10 01	0	0	1	0	1		1
0101001001	01 01 00 10 01	1	0	1	0	0		1
0101011001	01 01 01 10 01	1	1	1	1	0		1
0101101001	01 01 10 10 01	1	0	1	0	1		1
0110001001	01 10 00 10 01	0	0	1	0	0		1
0110011001	01 10 01 10 01	0	1	1	0	0		1
0110101001	01 10 10 10 01	0	0	1	1	1		1
1000001001	10 00 00 10 01	0	0	0	1	0		1
1000011001	10 00 01 10 01	0	0	0	0	0	*bryby*	0
1000101001	10 00 10 10 01	0	1	0	0	1		1
1001001001	10 01 00 10 01	0	0	0	0	0	*byrby*	0
1001011001	10 01 01 10 01	0	0	0	1	0		1
1001101001	10 01 10 10 01	0	1	0	0	1		1
1010001001	10 10 00 10 01	1	0	0	0	0		1
1010011001	10 10 01 10 01	1	0	0	0	0		1
1010101001	10 10 10 10 01	1	1	0	1	1		1

Fig. 4. (As Explained in Section 5) The construction of the pointer to those rows of the table in Figure 2 that give acceptable colorings. The edges of the graph being considered here are $\{a, b\}$, $\{a, c\}$, $\{a, e\}$, $\{b, c\}$, $\{c, d\}$, and $\{d, e\}$.

For tiny problem instances such as the one treated here, listing a coloring when one exists is not challenging. In the problem instance we have treated here three specific colorings are easily read. However as the size of problem instances grows the transparent squares become so tiny that, although light through P can be detected, it may be very difficult to individualize the location of a 'dot' of light. We have made some progress on this problem previously in Section 7 of [1]. But here we have drifted into the topic of Section 6.

Here is an exact accounting of the xerox operations our procedures require in solving any 3-color decision problem for a graph having n vertices and m edges. We assume $n > 2$ and $m > 1$.

Creating the initial row of Table:	1 xerox - of 0000000000 with bits 7 & 10 masked.
Construction of Table:	$2(n-2)$ xerox operations.
Creating Table$'$:	1 xerox operation.
Applying Equality tests:	$4(m-1)$ xerox operations.
Keeping a record of the solution P:	1 xerox operation.
Total:	$2n+4m-5$ xerox operations.

Why should counting the number of xerox operations made be accepted as a measure of complexity? Because the number of transparency maskings and overlayings associated with each xerox operation is bounded. Other operations, initialization and light testing, are performed only once.

6 Can the Present Suggestions Contribute to a Future Practical Technology Using Light?

In this article most of the space has been devoted to the treatment of a tiny case of an NP-complete algorithmic problem. Previous articles on 'transparent computing' have been devoted entirely to the treatment of NP-complete problems. In all these treatments there has been an analog of 'buying time with space'. We have 'bought time with density of information representation'. We have kept the actual physical space (the area of a transparency) constant, but the density of information in this restricted area grows without bound as the size of the description of instances grows. A human could solve only toy-scale NP-complete problems using a xerox machine as illustrated. (A SAT with ten variables could be treated, but probably not one with 11.) On the bright side, once one has the Table for a ten variable SAT one can solve almost immediately any SAT using 10 or fewer variables (only one xerox needed for each clause). The operations used (sliding transparencies left & right, testing for light through a column, & copying) can surely be carried out roboticly, but even a robot imitating hand operations would probably quickly hit a density limit. I do not know how the bound on workable density may grow as new light technologies (free of xerography) are developed. My hope is merely that my ideas for transparent computing suggest something of value to future designs for computing with light. Alternatively, the procedures used here may suggest procedures that can be implemented in other technologies not based on light.

The most pleasing feature for me of the present article is the elegance of the three step determination of equality of two multi-bit columns given in Section 4. Likewise, the most pleasing feature of [5] is the constructing, from an n-column table T having arbitrarily many rows, using only $O(n^2)$ steps, the two tables: (1) 'LShift' which is T with all its _ shifted to the left of all its ■ and (2) 'Cardinals' which, for each k $(1 < k < n)$, has a column that is a pointer to those rows of T in which _ occurs exactly k times. Such general purpose tools for handling columns of data may prove valuable for processing data stored as n-bit words. Moreover, the work in Section 2 of this article gives a hint for quickly locating all words in such a data base that have any specific values in any specified columns. A practical development from transparent computing might arise from the application of these tools and others to access, query and process in parallel data stored as fixed-length binary words.

References

1. Franco, G., Head, T.: Computing transparently: the subcovers of a set covering (2009) (manuscript)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability - A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., San Francisco (1979)
3. Head, T.: Photocomputing: explorations with transparency & opacity. Parallel Processing Letters 17, 339–347 (2007)
4. Head, T.: Parallel computing by xeroxing onto transparencies. In: Condon, A., Harel, D., Kok, J.N., Salomaa, A., Winfree, E. (eds.) Algorithmic Bioprocesses, pp. 631–637. Springer, Heidelberg (2009)
5. Head, T.: Computing transparently: the independent sets in a graph. Natural Computing (to appear 2010)
6. Head, T., Gal, S.: Aqueous computing: writing on molecules dissolved in water. In: Chen, J., Jonoska, N., Rozenberg, G. (eds.) Nanotechnology: Science and Computation, pp. 321–331. Springer, Heidelberg (2006)
7. Louri, A.: Optical content-addressable parallel processor: architecture, algorithms, and design concepts. Applied Optics 31, 3241–3258 (1992)
8. Naor, M., Shamir, A.: Visual Cryptography. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 1–12. Springer, Heidelberg (1995)
9. Oltean, M.: Solving the Hamiltonian path problem with a light based computer. Natural Computing 6, 57–70 (2008)
10. Schulte, D.: Rainbow sort: sorting at the speed of light. Natural Computing 5, 67–82 (2005)

Periodicity in Tilings

Emmanuel Jeandel and Pascal Vanier

Laboratoire d'Informatique Fondamentale (LIF)
Aix-Marseille Université, CNRS,
39 avenue Fréderic Joliot Curie, 13013 Marseille, France
emmanuel.jeandel@lif.univ-mrs.fr, pascal.vanier@lif.univ-mrs.fr

Abstract. Tilings and tiling systems are an abstract concept that arise both as a computational model and as a dynamical system. In this paper, we prove an analog of the theorems of Fagin [9] and Selman and Jones [14] by characterizing sets of periods of tiling systems by complexity classes.

Keywords: Computational and structural complexity, tilings, dynamical systems.

Introduction

The model of tilings was introduced by Wang [22] in the 60s to study decision problems on some classes of logical formulas. Roughly speaking, we are given some local constraints (a *tiling system*), and we consider colorings of the plane that respect these constraints (a *tiling*).

Tilings are both a simple and powerful model: the definitions are quite easy to grasp, however most decision problems on tilings are computationally intractable, starting from the most important one: decide whether a given tiling system tiles the plane [3]. This is due in part to a straightforward encoding of Turing machines in tilings (see e.g. [5,21]) and to the existence of *aperiodic* tiling systems, i.e. that produce many tilings, but none of them being periodic. In fact, Harel [12] and van Emde Boas [21] give strong evidence that tilings are more suitable than Turing machines and satisfiability problems to express hardness in both complexity and recursivity theory.

In a similar manner, we will show in this paper how to do *descriptive complexity* [8] with tilings. More precisely, we will prove that sets of *periods* of tilings correspond exactly to non-deterministic exponential time, hence to spectra of first order formulas [14]. As a consequence, the problems whether non-deterministic exponential time, first order spectra, or sets of periods are closed under complementation are equivalent.

The result is in itself not surprising: in fact, the proof of Jones and Selman [14] on spectra of first order formulas uses a multi-dimensional generalization of finite automata which may be interpreted as a kind of tiling system. Furthermore, the encoding of Turing machines in tilings, in particular in periodic tilings, is well known. This paper makes use of the fact that n steps of a Turing machine can be encoded into a tiling of period exactly n. Usual proofs [1,11] have a quadratic blowup, which is unsuitable for our purpose.

1 Tilings, Periodicity and Computations

For any dimension $d \geq 1$, a tiling of $\mathbb{Z}^d$ with a finite set of tiles T is a mapping $c : \mathbb{Z}^d \to T$. A $\mathbb{Z}^d$-tiling system is the pair (T, F), where F is a finite set of forbidden patterns $F \subset T^N$, where N is a finite subset of $\mathbb{Z}^d$ called the neighborhood. A tiling c is said to be *valid* if and only if none of the patterns of F ever appear in c. Since the number of forbidden patterns is finite, we could specify the rules by *allowed* patterns as well. We give an example of such a tiling system with the tiles of figure 1a and the forbidden patterns of figure 1b. The allowed tilings are shown in figure 1c.

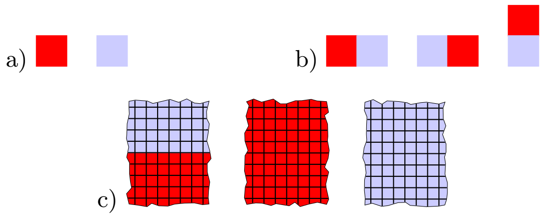


Fig. 1. The set of tiles (a) and the forbidden patterns (b) can only form tilings of the forms shown in (c)

The *cartesian product* of two tiling systems τ_1 and τ_2 is the tiling system obtained by superimposing the tiles of τ_1 to the tiles of τ_2 with the rules of τ_1 on the layer of τ_1 and the rules of τ_2 on the layer of τ_2. We will sometimes restrict the resulting tiling system by removing some superimposition of tiles.

A tiling c of dimension $d = 2$ is said to be *horizontally periodic* if and only if there exists a *period* $p \in \mathbb{N}^*$ such that for all $x, y \in \mathbb{Z}$, $c(x, y) = c(x + p, y)$. A tiling c of $\mathbb{Z}^d$ is *periodic* if it has the same period on all its dimensions:

$$c(x_1, x_2, \ldots, x_d) = c(x_1 + p, x_2, \ldots, x_d) = \cdots = c(x_1, x_2, \ldots, x_d + p)$$

The smallest such p is called the *(horizontal) eigenperiod* of c.

A tiling system is *aperiodic* if and only if it tiles the plane but there is no valid periodic[1] tiling. Such tiling systems have been shown to exist [3] and are at the core of the undecidability of the *domino problem* (decides if a given tiling system tiles the plane). J. Kari [15,17] gave such a tiling system with an interesting property: determinism. A tiling system is *NE-deterministic* (for *North-East*) if given two tiles respectively at the southern and western neighbour of a given cell, there is at most one tile that can be put in this cell so that the finite pattern is valid. The mechanism is shown in figure 2a.

[1] On none of the dimensions.

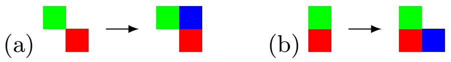


Fig. 2. North-East (NE) determinism (a) and East (E) determinism (b) in a tiling system

It is easy to change some details in order to have an other form of determinism: two tiles vertically adjacent will force their lower right neighboring tile as shown in figure 2b. This type of determinism will be called East-determinism. With such a tiling system, if a column of the plane is given, the half plane on its right is then determined.

As said earlier, tilings and recursivity are intimately linked. In fact, it is quite easy to encode Turing machines in tilings. Such encodings can be found e.g. in [16,7]. Given a Turing machine M, we can build a tiling system τ_M in figure 3. The tiling system is given by *Wang tiles*, i.e., we can only glue two tiles together if they coincide on their common edge. This tiling system τ_M has the following property: there is an accepting path for the word u in time (less than) t using space (less than) w if and only if we can tile a rectangle of size $(w+2)\times t$ with white borders, the first row containing the input. Note that this method works for both deterministic and non-deterministic machines.

2 Recognizing Languages with Tilings

Let τ be a $\mathbb{Z}^d$-tiling system. Then we define

$$\mathcal{L}_\tau = \{n \mid \text{there exists a tiling by } \tau \text{ of eigenperiod } n\}$$

If τ is a $\mathbb{Z}^2$-tiling system, we define

$$\mathcal{L}^h_\tau = \{n \mid \text{there exists a tiling by } \tau \text{ of horizontal eigenperiod } n\}$$

Definition 1. *A set $L \subseteq \mathbb{N}^\star$ is in $\mathcal{T}$ if $L = \mathcal{L}_\tau$ for some $\mathbb{Z}^d$-tiling system τ and some d. A set $L \subseteq \mathbb{N}^\star$ is in $\mathcal{H}$ if $L = \mathcal{L}^h_\tau$ for some $\mathbb{Z}^2$-tiling system τ.*

$\mathcal{T}$ and $\mathcal{H}$ are the classes of languages recognized by tiling and recognized horizontally by tiling respectively.

We say that a language $\mathcal{L}$ is recognized by a tiling system τ if and only if $\mathcal{L} = \mathcal{L}^h_\tau$ or $\mathcal{L} = \mathcal{L}_\tau$, depending on the context. It is easy to see that $\mathcal{T}$ and $\mathcal{H}$ are closed under union. It is not clear whether they are closed under intersection: if τ and τ' are tiling systems, a natural way to do intersection is to consider the *cartesian product* of τ and τ'. However, if for example $\mathcal{L}_\tau = \{2\}$ and $\mathcal{L}_{\tau'} = \{3\}$, then there exists in τ a tiling of eigenperiod 2, hence of period 6, and the same is true for τ', so that in this example $\tau \times \tau'$ will contain a tiling of eigenperiod 6 whereas $\mathcal{L}_\tau \cap \mathcal{L}_{\tau'} = \emptyset$.

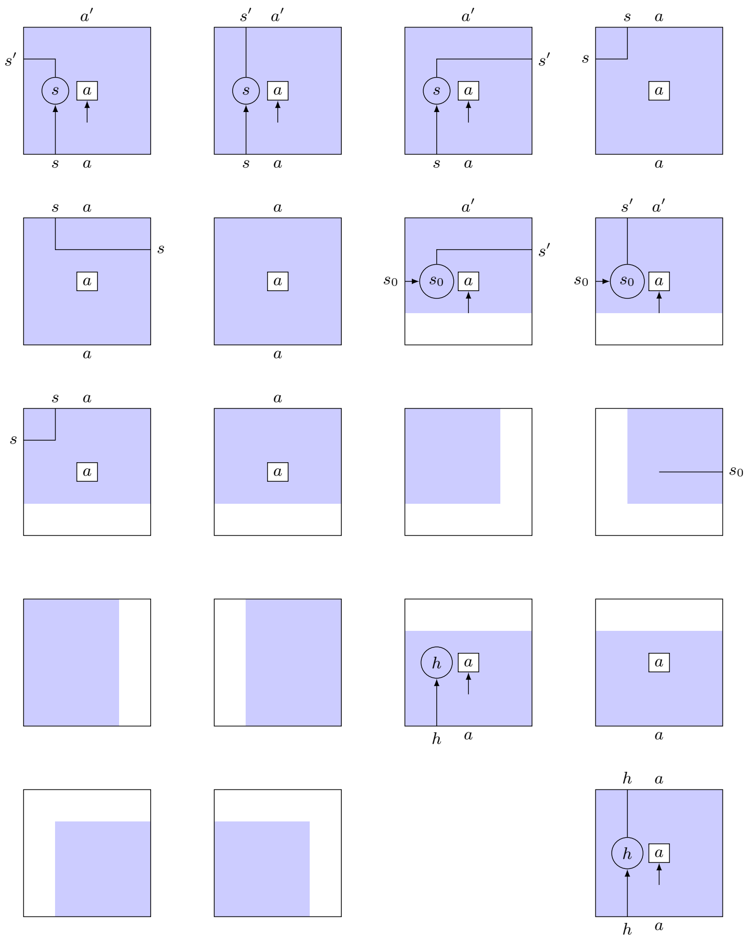


Fig. 3. A tiling system, given by Wang tiles, simulating a Turing machine. The meaning of the labels are the following:

- label s_0 represents the initial state of the Turing machine.
- The top-left tile corresponds to the case where the Turing machine, given the state s and the letter a on the tape, writes a', moves the head to the left and to change from state s to s'. The two other tiles are similar.
- h represents a halting state. Note that the only states that can appear in the last step of a computation (before a border appears) are halting states.

In this paper, we prove the following:

Theorem 1. *$\mathcal{H}$ is closed under union, intersection and complementation.*

Theorem 2. *$\mathcal{T}$ is closed under intersection. $\mathcal{T}$ is closed under complementation if and only if* ***NE*** *=* ***coNE****.*

Here **NE** is the class of languages recognized by a (one-tape) non-deterministic Turing machine in time 2^{cn} for some $c > 0$. Note that for theorem 2 we need to work in any dimension d. That is, if $\mathcal{L}_1, \mathcal{L}_2$ are the sets of periods of the $\mathbb{Z}^d$-tiling systems τ_1, τ_2, then there exists a $\mathbb{Z}^{d'}$-tiling system τ' that corresponds to the set of period $\mathcal{L}_1 \cap \mathcal{L}_2$. However d' can be larger than d.

To prove these theorems, we will actually give a characterization of our classes $\mathcal{H}$ and $\mathcal{T}$ in terms of structural complexity. This will be the purpose of the next two sections.

3 $\mathcal{H}$ and NSPACE(2^n)

To formulate our theorem, we consider sets of periods, i.e. subsets of $\mathbb{N}^\star$, as unary or binary languages. If $L \subset \mathbb{N}^\star$ then we define $un(L) = \{1^{n-1} | n \in L\}$. We define $bin(L)$ to be the set of binary representations (missing the leading one) of numbers of L. As an example, if $L = \{1, 4, 9\}$, then $un(L) = \{\epsilon, 111, 11111111\}$ and $bin(L) = \{\epsilon, 00, 001\}$. Note that any language over the letter 1 (resp. the letters $\{0, 1\}$) is the unary (resp. binary) representation of some subset of $\mathbb{N}^\star$.

We now proceed to the statement of the theorem:

Theorem 3. *Let $\mathcal{L}$ be a language, the following statements are equivalent:*

i) $\mathcal{L} \in \mathcal{H}$
ii) $un(\mathcal{L}) \in \mathbf{NSPACE}(n)$
iii) $bin(\mathcal{L}) \in \mathbf{NSPACE}(2^n)$

Recall that **NSPACE**(n) is the set of languages recognized by a (one-tape) non-deterministic Turing machine in space $\mathcal{O}(n)$.

The $(ii) \Leftrightarrow (iii)$ is folklore from computational complexity theory. The following two lemmas will prove the equivalence $(ii) \Leftrightarrow (i)$ hence the result.

Lemma 1. *For any tiling system τ, $un(\mathcal{L}^h_\tau) \in \mathbf{NSPACE}(n)$.*

Proof. Let $\tau = (T, F)$ be a tiling system. We will construct a non-deterministic Turing machine accepting 1^n if and only if $n + 1$ is a horizontal eigenperiod of τ. The machine has to work in space $\mathcal{O}(n)$, the input being given in unary.

Let $r > 0$ be an integer such that all patterns in the neighborhood N are smaller than a $r \times r$ square. Then a configuration c is correctly tiled if and only if all $r \times r$ blocks of c are correctly tiled. Furthermore, we can prove that if a horizontally periodic tiling of period n exists, then we can find such a tiling which is also vertically periodic of period at most $|T|^{rn}$. Now we give the algorithm, starting from n as an input:

– Initialize an array P of size n so that $P[i] = 1$ for all i.
– First choose non-deterministically $p \leq |T|^{rn}$
– Choose r bi-infinite rows $(c_i)_{0 \leq i \leq r-1}$ of period n (that is, choose $r \times n$ tiles).
– For each $r + 1 \leq i \leq p$, choose a bi-infinite row c_i of period n (that is, choose n tiles), and verify that all $r \times r$ blocks in the rows $c_i \dots c_{i-r+1}$ are correctly tiled. At each time, keep only the last r rows in memory (we never forget the r first rows though).

– (Verification of the eigenperiod) If at any of the previous steps, the row c_i is not periodic of period $k < n$, then $P[k] := 0$
– For $i \leq r$, verify that all $r \times r$ blocks in the rows $c_{p-i} \dots c_p c_0 \dots c_{i-r-1}$ are correctly tiled
– If there is some k such that $P[k] = 1$, reject. Otherwise accept.

This algorithm needs to keep only $2r$ rows at each time in memory, hence is in space $\mathcal{O}(n)$. □

Lemma 2. *For any unary language $L \in$ **NSPACE**(n), then $\{n \in \mathbb{N}^\star \mid 1^{n-1} \in L\} \in \mathcal{H}$.*

Proof. Let $L \in$ **NSPACE**(n), there exists then a non-deterministic Turing machine M accepting L in linear space. Using traditional tricks from complexity theory, we can suppose that on input 1^n the Turing machine uses exactly $n + 1$ cells of the tape (i.e. the input, with one additional cell on the right) and works in time exactly c^n for some constant c.

We will build a tiling system τ so that $1^n \in L$ if and only if $n+4$ is a period of the tiling τ. The modification to obtain $n + 1$ rather that $n + 4$, and thus prove the lemma, is left to the reader (basically "fatten" the gray tiles presented below so that they absorb 3 adjacent tiles), and serve no interest other than technical.

The proof may basically be split into two parts: First produce a tileset so that every tiling of horizontal period n looks like a grid of rectangles of size n by c^n delimited by gray cells (see fig. 6b). Then encode the Turing machine M inside these rectangles. The main difficulty is in the first part, the second part being relatively straightforward. Note however, that as M is nondeterministic, the computation in different rectangles might be different. To not break the periodicity, we will have to *synchronize* all the machines.

The tiling system will be made of several components (or *layers*), each of them having a specific goal. The components and their rules are as follows:

– The first component A is composed of an aperiodic E-deterministic tiling system, whose tiles will be called "whites". We take the one from section 1. We add a "gray" tile. The rules forbid any pattern containing a white tile above or below a gray tile. Hence a column containing a gray tile can only have gray tiles. We also forbid for technical reasons two gray tiles to appear next to each other horizontally.

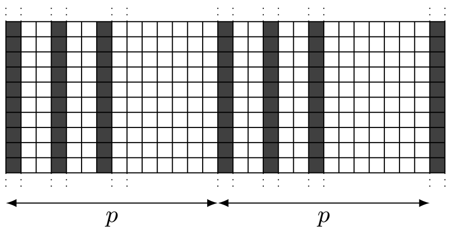


Fig. 4. A periodic tiling with the tiling system A

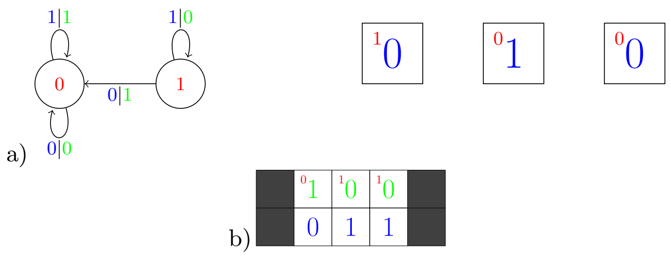


Fig. 5. a) The transducer doing the addition of one bit and the corresponding tiles. A valid tiling with these tiles is given in c).

With this construction, a periodic tiling of period p must have gray columns, as the white tiles form an aperiodic tileset.

For the moment nothing forbids more than one gray column to appear inside a period. Figure 4 shows a possible form of a periodic tiling at this stage.

– The second component $D = P \times \{R, B\}$ will produce gray rows so that the (horizontally periodic) tiling will consist of $n \times c^n$ white rectangles delimited by these gray columns and rows.

  The idea is as follows: suppose each word between two gray columns is a word over the alphabet $\{0, \dots c-1\}$, that is, represents a number k between 0 and $c^n - 1$. Then it is easy with a tiling system to ensure that the number on the *next* line is $k+1$ (with the convention $(c^n - 1) + 1 = 0$). See figure 5 for a transducer in the case $c = 2$ and its realization as a tiling system.

  With the $\{R, B\}$ subcomponent, we mark the lines corresponding to the number 0, so that one line out of c^n is marked. We proceed as follows. First, horizontal bi-infinite lines are uniform: that is a tile is R if and only if its left neighbour is R. Next, a gray tile is R if and only if it is at the west of a 0-tile and at the north-west of a $c-1$-tile. That is, lines that are colored in R represent the time when the number is reinitialized from $c^n - 1$ to 0.

  Figure 6a shows some typical tiling at this stage: The period of a tiling is not necessarily the same as the distance between the rectangles, it may be larger. Indeed, the white tiles in two consecutive rectangles may be different.

– Component T is only a copy of A (without the same rules) which allows us to synchronize the first white columns after the gray columns: synchronising these columns ensures that the aperiodic components between two gray columns are always the same, since the aperiodic tiles are E-deterministic. The rules are simple, two horizontal neighbors have the same value on this component and a tile having a gray tile on its left has the same value in A as in T.

  At this stage, we have regular rectangles on all the plane, whose width correspond to the period of the tiling, as shown in figure 6b.

– The last component M is the component allowing us to encode Turing machines in each rectangle. We use the encoding τ_M we described previously in section 1. We force the computation to appear inside the white tiles: the

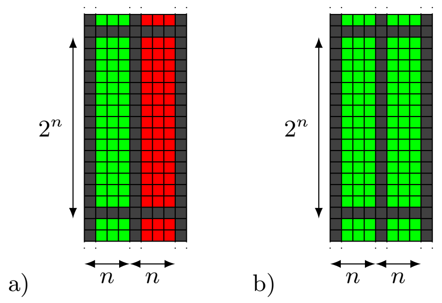


Fig. 6. a) The form after adding the component D, the aperiodic components between two gray columns can be different. b) After adding also the component T, the aperiodic components are exactly the same.

white bottom borders must appear only in the row R, and the row below will have top borders. And the two tiles between the row R and the gray tiles are corner tiles. Finally, the input of the Turing machine (hence the row above the row R) consists of only "1" symbols, with a final blank symbol.

The Turing machines considered here being non-deterministic, there could be different valid transitions on two horizontally adjacent rectangles, that is why we synchronize the transitions on each row. The method for the synchronization of the transition is almost the same as the method for the synchronisation of the aperiodic components, and thus not provided here.

Now we prove that $1^n \in L$ if and only if $n + 4$ is a period of the tiling system τ. First suppose that n is a period and consider a tiling of period n.

- Due to component A, a gray column must appear. The period is a succession of either gray and white columns.
- Due to the component D, the gray columns are spaced by a period of p, $p < n$.
- Due to component T the tiling we obtain is (horizontally) p-periodic when restricted to the components D, T, A.
- For the component M to be correctly tiled, the input 1^{p-4} ($4 = 1$ (gray) + 1 (left border) + 1 (right border) + 1 (blank marker)) must be accepted by the Turing machine, hence $1^{p-4} \in L$
- Finally, due to the synchronization of the non-deterministic transition, the M component is also p-periodic. As a consequence, our tiling is p-periodic, hence $n = p - 4$. Therefore $1^{n+4} \in L$

Conversely, suppose $1^n \in L$. Consider the coloring of period $n + 4$ obtained as follows (only a period is described):

- The component A consists of $n + 3$ correctly tiled columns of our aperiodic E-deterministic tiling systems, with an additional gray column. As the E-deterministic tiling system tiles the plane, such a tiling is possible

- The component M corresponds to a successful computation path of the Turing machine on the input 1^n, that exists by hypothesis. As the computation lasts exactly than c^n steps, the computation fits exactly inside the $n \times c^n$ rectangle.
- We then add all other layers according to the rules to obtain a valid configuration, thus obtaining a valid tiling of period exactly $n+4$. □

Corollary 1. *The languages recognized horizontally by tiling are closed under intersection and complementation.*

Proof. Immerman-Szelepczenyi's theorem [13,2] states that non-deterministic space complexity classes are closed under complementation. The result is then a consequence of theorem 3. □

This theorem could be generalized to tilings of dimension d by considering tilings having a period n on $d-1$ dimensions, as explained in [4].

4 $\mathcal{T}$ and NE

We now proceed to total periods rather than horizontal periods. We will prove:

Theorem 4. *Let $\mathcal{L} \subset \mathbb{N}^\star$ be a language, the following statements are equivalent:*

i) There exists a $\mathbb{Z}^d$-tiling system τ for some d so that $\mathcal{L} = \mathcal{L}_\tau$
*ii) $un(\mathcal{L}) \in$ **NP***
*iii) $bin(\mathcal{L}) \in$ **NE***

In these cases, $\mathcal{L}$ is also the spectrum of a first order formula, see [14].

We will obtain as a corollary theorem 2. Note the slight difference in formulation between theorem 4 and theorem 3. While we can encode a Turing machine working in time n in a tiling of size n^2, we cannot check the validity of the tiling in less than $\mathcal{O}(n^2)$ time steps. More generally, it is unclear whether we can encode a Turing machine working in time n^c in a tiling of size less than n^{2c}. To overcome this gap, we need to work in any dimension d: A language $L \in \mathbf{NTIME}(n^d)$ will be encoded into a tiling in dimension $2d$ and a tiling in dimension d will be encoded into a language L in $\mathbf{NTIME}(n^d)$.

The gap here is not surprising: while space complexity classes are usually model independent, this is not the case for time complexity, where the exact definition of the computational model matters. An exact characterization of periodic tilings for $d = 2$ is in fact possible, but messy: it would involve Turing machines working in space $O(n)$ with $O(n)$ reversals, see e.g [6].

Proof. The statements $(i) \Rightarrow (ii) \Leftrightarrow (iii)$ were already explained. So we only have to prove $(ii) \Rightarrow (i)$. We will see how a language $L \in \mathbf{NTIME}(n^d)$ will be encoded as periods in dimension $2d$. The proof is similar to the previous one, and we provide only a sketch due to the lack of space. There are two steps:

- Build a tileset τ so that every tiling of period n looks like a lattice of hypercubes of size n delimited by gray cells.
- Encode the computation of the Turing Machine inside the cubes.

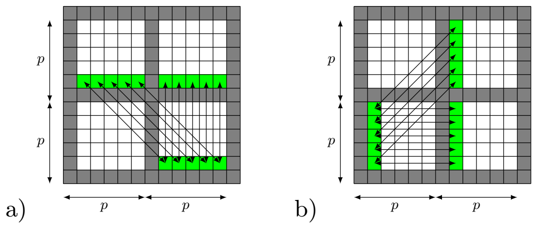


Fig. 7. Transmission of the first row in a) and of the first column in b)

First step. We will work in dimension 2, as it is relatively straightforward to adapt the technique to higher dimensions.

The idea is to take an aperiodic NE-deterministic tiling system as white tiles. Then we add in this component three gray tiles: a cross tile, a horizontal tile and a vertical tile: gray horizontal (resp. vertical) lines consist of horizontal (resp. vertical) tiles, and they can intersect only on cross tiles.

Let A be this component. Now consider a periodic tiling. This tiling cannot contain only white tiles. Hence, it contains a horizontal tile, a vertical tile or a cross tile. The problem is that the presence of a horizontal(resp. vertical) tile does not imply the presence of a vertical (resp. horizontal) tile. The idea is to use in the next component an unary counter in *both* directions, so that it creates horizontal lines between vertical columns, and conversely.

The next components are also simple. The key point is that we use a NE-deterministic tiling system (rather than an E-deterministic) to ensure that all white squares contain the same tiles: in a NE-deterministic tiling, a square is entirely determined by its first row and its first column. Hence it is sufficient to synchronize the first row and the first column of all squares to synchronize the aperiodic components. A way to do this is given in figure 7.

Second step We now have to explain how to encode the computation of a Turing Machine working in time n^d into a cube of size n in dimension $2d$. The idea is to *fold* the space-time diagram of the Turing machine so that if fits into the cube. Each cell of the space-time diagram has coordinates (t, s) with $t \leq n^d, s \leq n^d$. We now have to transform each cell (t, s) into a cell of the hypercube of size n in dimension $2d$ so that two consecutive (in time or space) cells of the space-time diagram correspond to two adjacent cells of the cube, so that we can verify *locally* that the cube indeed encodes the computation of the Turing machine. This is exactly what a reflected n-ary Gray code [10,18] does.

Such a folding has already been described by Borchert [4] and can also be deduced from [14]. Basically, the cell at position $(t_0, \ldots, t_{d-1}) \in [0, n-1]^d$ will represent the integer $t = \sum a_i n^i$ where $a_i = t_i$ if $\sum_{j>i} t_j$ is even, and $a_i = n-1-t_i$ otherwise (formula (51) in [18]). The noticeable fact is that the direction in which to look for the next positition is given by the parity of the sum of the *stronger* bits. Hence, we will encode these parity layers in the tiling, an example for a three dimensional folding can be seen on figure 8.

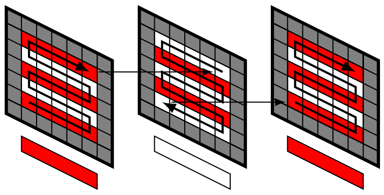


Fig. 8. Folding of a three dimensional cube, the red on the parity layer stands for +1 and the white for -1. The direction where to look for the next cell is given by the parity layer.

However recall that Turing machines are non-deterministic, so we have to synchronize the transitions between the different hypercubes, as was done in the previous theorem. Similar techniques can be used. □

Concluding Remarks

The results presented in this paper establish a link between computational complexity and tilings, in particular between the complexity classes $\mathbf{NSPACE}(2^n)$, $\mathbf{NE}$ and the sets of periods of the tilings. The result is very different from the sets of periods we can obtain for 1-dimensional tilings (subshifts of finite type, see [20])

For 1-dimensional tilings, the number c_n of tilings of period n is enclosed in the *zeta* function: $\zeta(z) = \exp\left(\sum \frac{c_n}{n} z^n\right)$ The zeta function is well understood in dimension 1 [20]. A zeta function for tilings of the plane has been described by Lind [19]. To be thorough, one needs to count not only tilings of period n as we did in this article, but tilings of period Γ, where Γ is any lattice. We think a complete characterization of sets of periods for general lattices Γ of $\mathbb{Z}^2$ is out of reach. If we accept to deal only with square periods (that is with the lattices $n\mathbb{Z} \times n\mathbb{Z}$), as we did here, preliminary work suggests we can characterize the number of periodic tilings via the class $\#\mathbf{E}$.

References

1. Allauzen, C., Durand, B.: Tiling Problems. In: The classical decision problem, Perspectives in Mathematical Logic, ch. Appendix A. Springer, Heidelberg (2001)
2. Balcazar, J., Diaz, J., Gabarro, J.: Structural Complexity II. Springer, Heidelberg (1988)
3. Berger, R.: The Undecidability of the Domino Problem. Memoirs of the American Mathematical Society, vol. 66. The American Mathematical Society, Providence (1966)
4. Borchert, B.: Formal Language characterizations of P, NP, and PSPACE. Journal of Automata, Languages and Combinatorics 13(3/4), 161–183 (2008)
5. Buchi, J.R.: Turing-Machines and the Entscheidungsproblem. Math. Annalen 148, 201–213 (1962)

6. Carayol, A., Meyer, A.: Context-Sensitive Languages, Rational Graphs and Determinism. Logical Methods in Computer Science 2(2), 1–24 (2006)
7. Chaitin, G.: The Halting Probability via Wang Tiles. Fundamenta Informaticae 86(4), 429–433 (2008)
8. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory. Springer Monographs in Mathematics. Springer, Berlin (1995)
9. Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. In: Karp, R. (ed.) SIAM-AMS Proceedings of Complexity of Computation, vol. 7, pp. 43–73 (1974)
10. Flores, I.: Reflected Number Systems. IRE Transactions on Electronic Computers EC-5(2), 79–82 (1956)
11. Gurevich, Y., Koryakov, I.: Remarks on Berger's paper on the domino problem. Siberian Math. Journal (1972)
12. Harel, D.: Recurring Dominoes: Making the Highly Undecidable Highly Understandable. Annals of Discrete Mathematics 24, 51–72 (1985)
13. Immerman, N.: Nondeterministic space is closed under complementation. SIAM Journal on Computing 17(5), 935–938 (1988)
14. Jones, N.D., Selman, A.L.: Turing machines and the spectra of first-order formulas with equality. In: STOC 1972: Proceedings of the Fourth Annual ACM Symposium on Theory of Computing, pp. 157–167 (1972)
15. Kari, J.: The Nilpotency Problem of One-Dimensional Cellular Automata. SIAM Journal on Computing 21(3), 571–586 (1992)
16. Kari, J.: Reversibility and surjectivity problems of cellular automata. J. Comput. Syst. Sci. 48(1), 149–182 (1994)
17. Kari, J., Papasoglu, P.: Deterministic Aperiodic Tile Sets. Geometric And Functional Analysis 9, 353–369 (1999)
18. Knuth, D.E.: Generating all Tuples and Permutations. In: The Art of Computer Programming, vol. 4 fasc. 2. Addison-Wesley. Reading (2005)
19. Lind, D.: A zeta function for $\mathbb{Z}^d$-actions. In: Proceedings of Warwick Symposium on $\mathbb{Z}^d$ actions. LMS Lecture Notes Series, vol. 228, pp. 433–450. Cambridge Univ. Press, Cambridge (1996)
20. Lind, D.A., Marcus, B.: An Introduction to Symbolic Dynamics and Coding. Cambridge University Press, New York (1995)
21. van Emde Boas, P.: The convenience of Tilings. In: Complexity, Logic, and Recursion Theory. Lecture Notes in Pure and Applied Mathematics, vol. 187. CRC, Boca Raton (1997)
22. Wang, H.: Proving theorems by Pattern Recognition II. Bell Systems Technical Journal 40, 1–41 (1961)

Complexity in Union-Free Regular Languages

Galina Jirásková[1,⋆] and Tomáš Masopust[2,⋆⋆]

[1] Mathematical Institute, Slovak Academy of Sciences
Grešákova 6, 040 01 Košice, Slovak Republic
jiraskov@saske.sk

[2] Mathematical Institute, Czech Academy of Sciences
Žižkova 22, 616 62 Brno, Czech Republic
masopust@ipm.cz

Abstract. We continue the investigation of union-free regular languages that are described by regular expressions without the union operation. We also define deterministic union-free languages as languages recognized by one-cycle-free-path deterministic finite automata, and show that they are properly included in the class of union-free languages. We prove that (deterministic) union-freeness of languages does not accelerate regular operations, except for the reversal in the nondeterministic case.

1 Introduction

Regular languages are the simplest languages in the Chomsky hierarchy. They have been intensively investigated due to their practical applications in various areas of computer science, and for their importance in the theory as well.

In recent years, several special subclasses have been deeply examined: finite languages that can be described by expressions without the star operation [6,7,32], suffix- and prefix-free languages that are used in codes [12], star-free and locally testable languages, ideal, closed, and convex languages, etc.

Here we continue this research and study union-free regular languages that can be represented by regular expressions without the union operation. Nagy in [26] described one-cycle-free-path nondeterministic finite automata, in which from each state, there is exactly one cycle-free path to the final state. He showed that such automata accept exactly the class of union-free languages. We first complement his results with some closure properties. Then, in Section 3, we investigate the nondeterministic state complexity of operations in the class of union-free languages. Quite surprisingly, we show that all known upper bounds can be reached by union-free languages, except for the reversal, where the tight bound is n instead of $n + 1$. In Section 4, we define deterministic union-free languages as languages accepted by deterministic one-cycle-free-path automata, and show that they are properly included in the class of union-free languages. We study the state complexity of quite a number of regular operations, and prove that deterministic union-freeness of languages does not accelerate any of them.

⋆ Research supported by the VEGA Grant no. 2/0111/09.

⋆⋆ Research supported by the CAS Institutional Research Plan no. AV0Z10190503.

To conclude this section, we mention three more related works. Crvenković, Dolinka, and Ésik [9] investigated algebraic properties of union-free languages. Nagy [25] and Afonin and Golomazov [2] studied union-free decompositions of regular languages.

2 Preliminaries

We assume familiarity with basic concepts of finite automata and regular languages. For all unexplained notions, we refer the reader to [29,31,32].

If Σ is a finite alphabet, then Σ^* denotes the set of all strings over the alphabet Σ including the empty string ε. A *language* over an alphabet Σ is any subset of Σ^*. We denote the size of a finite set A by $|A|$ and its power-set by 2^A.

A *nondeterministic finite automaton* (nfa) is a quintuple $M = (Q, \Sigma, \delta, S, F)$, where Q is a finite non-empty set of states, Σ is an input alphabet, S is the set of initial states, F is the set of accepting states, and δ is the transition function that maps $Q \times (\Sigma \cup \{\varepsilon\})$ into 2^Q. The transition function is extended to the domain $2^Q \times \Sigma^*$ in a natural way. The *language accepted by* the nfa M is the set of all strings accepted by M. The automaton M is *deterministic* (dfa) if it has a single initial state, no ε-transitions, and $|\delta(q, a)| = 1$ for all states q in Q and symbols a in Σ. In this case, we usually write $\delta : Q \times \Sigma \to Q$.

A language is *regular* if there exists an nfa (or a dfa) accepting the language. The *state complexity* of a regular language L, denoted by $\mathrm{sc}(L)$, is the smallest number of states in any dfa accepting the language L. The *nondeterministic state complexity* of a regular language L, $\mathrm{nsc}(L)$, is defined as the smallest number of states in any ε-free nfa that accepts L and has a single initial state.

A path from state p to state q in an nfa/dfa M is a sequence $p_0 a_1 p_1 a_2 \cdots a_n p_n$, where $p_0 = p$, $p_n = q$, and $p_i \in \delta(p_{i-1}, a_i)$ for $i = 1, 2, \ldots, n$. The path is called *accepting cycle-free* if p_n is an accepting state, and $p_i \neq p_j$ whenever $i \neq j$. An nfa/dfa is a *one-cycle-free-path* (1cfp) nfa/dfa if there is a unique accepting cycle-free path from each of its states.

A *regular expression* over an alphabet Σ is defined inductively as follows: $\emptyset$, ε, and a, for a in Σ, are regular expressions. If r and t are regular expressions, then also $(s \cup t)$, $(s \cdot t)$, and $(s)^*$ are regular expressions. A regular expression is *union-free* if no symbol $\cup$ occurs in it. A regular language is *union-free* if there exists a union-free regular expression describing the language.

Let K and L be languages over Σ. We denote by $K \cap L$, $K \cup L$, $K - L$, $K \oplus L$ the intersection, union, difference, and symmetric difference of K and L, respectively. To denote complement, Kleene star, and reversal of L, we use L^c, L^*, and L^R. The left and right quotient of a language L with respect to a string w is the set $w \backslash L = \{x \mid wx \in L\}$ and $L/w = \{x \mid xw \in L\}$, respectively. The cyclic shift of a language L is defined as $L^{shift} = \{uv \mid vu \in L\}$. The *shuffle* of two languages is $K \sqcup\!\sqcup L = \{u_1 v_1 u_2 v_2 \cdots u_m v_m \mid m \geqslant 1,\, u_i, v_i \in \Sigma^*,\, u_1 \cdots u_m \in K,\, v_1 \cdots v_m \in L\}$. For the definition of positional addition, $K + L$, we refer to [18]: informally, strings are considered as numbers encoded in a $|\Sigma|$-adic system, and automata read their inputs from the least significant digit.

3 Union-Free Regular Languages

A regular language is union-free if it can be described by a union-free regular expression. Nagy in [26] proved that the class of union-free regular languages coincides with the class of languages recognized by one-cycle-free-path nfa's. He also showed that union-free languages are closed under concatenation, Kleene-star, and substitution by a union-free language. Using an observation that the shortest string of a union-free language is unique, he proved not closeness under union, complementation, intersection, and substitution by a regular language. Our first result complements the closure properties.

Theorem 1 (Closure Properties). *The class of union-free regular languages is closed under reversal, but not closed under cyclic shift, shuffle, symmetric difference, difference, left and right quotients, and positional addition.*

Proof. We prove the closeness under reversal by induction on the structure of a regular expression. If r is $\emptyset$, or ε, or a, then the reversal is described by the same expression. If $r = st$, or $r = s^*$, then the reversal is $L(t)^R L(s)^R$ or $(L(s)^R)^*$, respectively, which are union-free due to closeness under concatenation and star.

To prove the non-closure properties, we give union-free languages such that the shortest string in the resulting language is always of length two, and we show that there are at least two such strings in all cases: $\{ab\}^{shift} = \{a\} \sqcup\!\sqcup \{b\} = \{ab\} \oplus \{ba\} = \{ab, ba\}$; $a(b \cup c)^* - a^* = \{ab, ac, \dots\}$; $g \backslash (ge \cup gf)^* b = \{eb, fb, \dots\}$ and $a(eb \cup fb)^*/b = \{ae, af, \dots\}$; $88^* + 33^* = \{11, 19, \dots\}$. As the shortest strings are not unique, the resulting languages are not union-free. $\square$

The subset construction assures that every nfa of n states can be simulated by a dfa of at most 2^n states. The worst case binary examples are known for a long time [20,22,24]. In addition, Domaratzki et al. [10] have shown that there are at least 2^{n-2} distinct binary languages recognized by nfa's of n states that require 2^n deterministic states. However, none of the above mentioned automata is one-cycle-free-path nfa. The following theorem shows that the bound 2^n is tight in the class of union-free regular languages as well.

Theorem 2 (NFA to DFA Conversion). *For every positive integer n, there exists a binary one-cycle-free-path nfa of n states whose equivalent minimal dfa has 2^n states.*

Proof. Consider the binary 1cfp nfa with states $0, 1, \dots, n-1$, of which 0 is the initial state, and $n-1$ is the sole accepting state. By a, each state i goes to $\{i+1\}$, except for state $n-1$ which goes to the empty set. By b, each state i goes to $\{0, i\}$. Let us show that the corresponding subset automaton has 2^n reachable and pairwise inequivalent states. Each singleton $\{i\}$ is reached from the initial state $\{0\}$ by a^i, and the empty set is reached by a^n. Each set $\{i_1, i_2, \dots, i_k\}$, where $0 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n-1$, of size k $(2 \leqslant k \leqslant n)$ is reached from the set $\{i_2 - i_1, i_3 - i_1, \dots, i_k - i_1\}$ of size $k-1$ by the string ba^{i_1}. This proves the reachability of all subsets. For inequivalence, notice that the string a^{n-1-i} is accepted by the nfa only from state i. Two different subsets must differ in a state i, and so the string a^{n-1-i} distinguishes the two subsets. $\square$

We next study the nondeterministic state complexity of regular operations in the class of union-free languages. Quite surprisingly, all upper bounds can be reached by union-free languages, except for the reversal where the upper bound is n instead of $n+1$. To prove the results we use a fooling set lower-bound technique [3,4,5,11,14].

A set of pairs of strings $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ is called a *fooling set* for a language L if (1) for all i, the string $x_i y_i$ is in the language L, and (2) if $i \neq j$, then at least one of the strings $x_i y_j$ and $x_j y_i$ is not in the language L.

It is well-known that the size of a fooling set for a regular language provides a lower bound on the number of states in any nfa for this language. The argument is simple. We can fix accepting computations of any nfa on strings $x_i y_i$. The states on these computations reached after reading x_i must be pairwise distinct, otherwise the nfa would accept both $x_i y_j$ and $x_j y_i$ for two distinct pairs.

The next lemma shows that sometimes one more state is necessary. The lemma can be used to simplify some proofs from the literature, for example, the results on union, reversal, and cyclic shift of nfa languages.

Lemma 1. *Let L be a regular language. Let $\mathcal{A}$ and $\mathcal{B}$ be sets of pairs of strings and let u and v be two strings such that $\mathcal{A} \cup \mathcal{B}$, $\mathcal{A} \cup \{(\varepsilon, u)\}$, and $\mathcal{B} \cup \{(\varepsilon, v)\}$ are fooling sets for L. Then every nfa for L has at least $|\mathcal{A}| + |\mathcal{B}| + 1$ states.*

Proof. Consider an nfa for L, and let $\mathcal{A} = \{(x_i, y_i) \mid i = 1, 2, \ldots, m\}$ and $\mathcal{B} = \{(x_{m+j}, y_{m+j}) \mid j = 1, 2, \ldots, n\}$. Since the strings $x_k y_k$ are in L, we can fix an accepting computation of the nfa on each string $x_k y_k$. Let p_k be the state on this computation that is reached after reading x_k. Since $\mathcal{A} \cup \mathcal{B}$ is a fooling set for L, the states $p_1, p_2, \ldots, p_{m+n}$ must be pairwise distinct. Since $\mathcal{A} \cup \{(\varepsilon, u)\}$ is a fooling set, the initial state must be distinct from all states $p_1, p_2, \ldots, p_m$. Since $\mathcal{B} \cup \{(\varepsilon, v)\}$ is a fooling set, the initial state must also be distinct from all states $p_{m+1}, p_{m+2}, \ldots, p_{m+n}$. Thus the nfa has at least $m + n + 1$ states. □

Theorem 3 (Nondeterministic State Complexity). *Let K and L be union-free regular languages over an alphabet Σ accepted by an m-state and an n-state one-cycle-free-path nfa, respectively. Then*
1. $\text{nsc}(K \cup L) \leqslant m + n + 1$, *and the bound is tight if* $|\Sigma| \geqslant 2$;
2. $\text{nsc}(K \cap L) \leqslant mn$, *and the bound is tight if* $|\Sigma| \geqslant 2$;
3. $\text{nsc}(KL) \leqslant m + n$, *and the bound is tight if* $|\Sigma| \geqslant 2$;
4. $\text{nsc}(K \sqcup\!\sqcup L) \leqslant mn$, *and the bound is tight if* $|\Sigma| \geqslant 2$;
5. $\text{nsc}(K + L) \leqslant 2mn + 2m + 2n + 1$, *and the bound is tight if* $|\Sigma| \geqslant 6$;
6. $\text{nsc}(L^2) \leqslant 2n$, *and the bound is tight if* $|\Sigma| \geqslant 2$;
7. $\text{nsc}(L^c) \leqslant 2^n$, *and the bound is tight if* $|\Sigma| \geqslant 3$;
8. $\text{nsc}(L^R) \leqslant n$, *and the bound is tight if* $|\Sigma| \geqslant 1$;
9. $\text{nsc}(L^*) \leqslant n + 1$, *and the bound is tight if* $|\Sigma| \geqslant 1$;
10. $\text{nsc}(L^{shift}) \leqslant 2n^2 + 1$, *and the bound is tight if* $|\Sigma| \geqslant 2$;

Proof. 1. To get an nfa for the union, we add a new initial state that goes by the empty string to the initial states of the given automata. For tightness, consider binary union-free languages $(a^m)^*$ and $(b^n)^*$ [13], and the following sets of pairs of strings: $\mathcal{A} = \{(a^i, a^{m-i}) \mid i = 1, 2, \ldots, m-1\} \cup \{(a^m, a^m)\}$ and $\mathcal{B} = \{(b^j, b^{n-j}) \mid j = 1, 2, \ldots, n-1\} \cup \{(b^n, b^n)\}$. Let $L = (a^m)^* \cup (b^n)^*$, and let us show that the set $\mathcal{A} \cup \mathcal{B}$ is a fooling set for the language L. The concatenation of the first and the second part of each pair results in a string in $\{a^m, a^{2m}, b^n, b^{2n}\}$, and so is in the language L. Next, the concatenation of the first part of a pair and the second part of another pair results in a string in $\{a^r, a^{m+r}, b^s, b^{n+s}, a^r b^s, b^s a^r, a^m b^n, b^n a^m \mid 0 < r < m, 0 < s < n\}$, and so is not in L. Finally, both sets $\mathcal{A} \cup \{(\varepsilon, b^n)\}$ and $\mathcal{B} \cup \{(\varepsilon, a^m)\}$ are fooling sets for L as well. By Lemma 1, every nfa for L has at least $m + n + 1$ states.

2. Standard cross-product construction provides the upper bound mn on the intersection. To prove that the bound is tight consider binary 1cfp nfa's that count the number of a's modulo m and the number of b's modulo n, respectively. Then the set $\{(a^i b^j, a^{m-i} b^{n-j}) \mid 0 \leqslant i \leqslant m-1,\ 0 \leqslant j \leqslant n-1\}$ is a fooling set of size mn for the intersection of the languages accepted by the two automata.

3. To get an nfa for the concatenation from two given nfa's, we only need to add an ε-transition from all final states in the first automaton to the initial state in the second automaton. For tightness, consider languages $(a^m)^*$ and $(b^n)^*$. The set $\{(a^i, a^{m-i} b^n) \mid i = 0, 1, \ldots, m-1\} \cup \{(a^m b^j, b^{n-j}) \mid j = 1, 2, \ldots, n\}$ is a fooling set of size $m + n$ for the concatenation of the two languages.

4. The state set of an nfa for the shuffle is the product of the state sets of given nfa's, and its transition function δ is defined using transitions functions δ_A and δ_B of the given automata by $\delta((p, q), a) = \{(\delta_A(p, a), q), (p, \delta_B(q, a))\}$ [8]. This gives the upper bound mn. The bound is reached by the shuffle of the languages $(a^m)^*$ and $(b^n)^*$ since the set $\{(a^i b^j, a^{m-i} b^{n-j}) \mid 0 \leqslant i \leqslant m-1,\ 0 \leqslant j \leqslant n-1\}$ is a fooling set of size mn for the shuffle.

5. The nfa for positional addition in [18] consists of $2mn + 2m + 2n + 1$ states, and it is shown here that the bound is reached by the positional addition of union-free languages $((1^*5)^m)^*$ and $((2^*5)^n)^*$ over the alphabet $\{0, 1, 2, 3, 4, 5\}$.

6. Since L^2 is the concatenation of the language L with itself, the upper bound $2n$ follows from part 3. To prove tightness consider the 1cfp nfa shown in Fig. 1. Construct an nfa with the state set $Q = \{p_0, p_1, \ldots, p_{n-1}\} \cup \{q_0, q_1, \ldots, q_{n-1}\}$ for the language L^2 from two copies of the nfa for L by adding an ε-transition from the final state in the first copy to the initial state in the second copy. The initial state of the resulting nfa is p_0, the only final state is q_{n-1}. For each state s in Q, define strings x_s and y_s as follows (notice that for each state s, the initial state p_0 goes to s by x_s, and each s goes to the accepting state q_{n-1} by y_s):

$$x_s = \begin{cases} a^i & \text{if } s = p_i, \\ a^{2n-2} b^{n-1-i} & \text{if } s = q_i, \end{cases} \qquad y_s = \begin{cases} a^{2n-2-i} & \text{if } s = p_i \text{ and } i \neq n-1, \\ b^{n-1} a^{2n-2} & \text{if } s = p_{n-1}, \\ a^{n-1-i} & \text{if } s = q_i. \end{cases}$$

Then the set $\{(x_s, y_s) \mid s \in Q\}$ is a fooling set for the language L^2 of size $2n$.

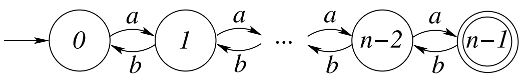


Fig. 1. The one-cycle-free nfa reaching the bound $2n$ on square

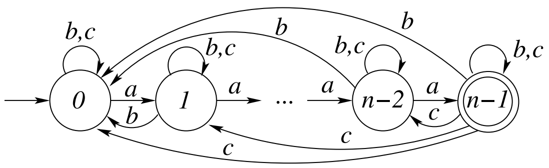


Fig. 2. The one-cycle-free nfa reaching the bound 2^n on complement

7. After applying subset construction to a given nfa and interchanging the accepting and rejecting states, we get an nfa (even a dfa) of at most 2^n states for the complement of the language recognized by the given nfa. The bound has been proved to be tight for a growing alphabet in [28], for a four-letter alphabet in [5], and for a binary alphabet in [16]. However, the binary witness nfa's in [16] are not 1cfp. We prove the tightness of the bound also in the class of 1cfp automata. To this aim consider the ternary language L recognized by the 1cfp nfa in Fig. 2; denote the state set $\{0, 1, \dots, n-1\}$ by Q. By c, state $n-1$ goes to $\{0, 1, \dots, n-1\}$, and each other state i goes to $\{i\}$. Transitions by a and b are the same as in the automaton in the proof of Theorem 2. Therefore, in the corresponding subset automaton, each subset S of the state set Q is reached from the initial state $\{0\}$ by a string x_S in $\{a, b\}^*$. We are now going to define strings y_S so that the set $\{(x_S, y_S) \mid S \subseteq Q\}$ would be a fooling set for L^c.

Let S be a subset of Q. If $S = \{0, 1, \dots, n-2\}$, let $y_S = c$, otherwise let $y_S = y_1 y_2 \cdots y_n$, where for each i in Q, $y_{n-i} = a$ if $i \in S$, and $y_{n-i} = ca$ if $i \notin S$. Then the set $\{(x_S, y_S) \mid S \subseteq Q\}$ is a fooling set for the language L^c of size 2^n.

8. To get an n-state nfa for the reversal of a language accepted by an n-state 1cfp nfa, we reverse all transitions, make the initial state final, and (the only) final state the initial. The unary union-free language a^{n-1} reaches the bound.

9. The standard construction of an nfa for the Kleene star that adds a new initial (and accepting) state connected through an ε-transition to the initial state of the given nfa as well as ε-transitions from each final state to the initial state, provides the upper bound $n + 1$. For tightness, consider the union-free language $a^{n-1}(a^n)^*$. The set $\{(\varepsilon, \varepsilon)\} \cup \{(a^i, a^{n-1-i}) \mid i = 1, 2, \dots, n-2\} \cup \{(a^{n-1}, a^n), (a^n, a^{n-1})\}$ is a fooling set of size $n+1$ for the star of this language.

10. The nfa for cyclic shift in consists of $2n^2 + 1$ states, and the one-cycle-free-free nfa in Fig. 1 reaches the bound [17]. To prove the result, a fooling set of size $2n^2$ is described in [17], and then Lemma 1 is used to show that one more state is necessary. □

4 Deterministic Union-Free Regular Languages

We now turn our attention to deterministic union-free languages, that is, to languages that are recognized by one-cycle-free-path deterministic finite automata. We first show that deterministic union-free languages are properly included in the class of union-free languages. Then we study the state complexity of regular operations in the class of deterministic union-free languages.

Theorem 4 (1cfp DFAs vs. 1cfp NFAs). *The class of deterministic union-free languages is a proper subclass of the class of union-free regular languages.*

Proof. Let $k \geqslant 3$. We show that there exists a unary union-free regular language such that every dfa for this language has at least k final states, and so the language is not deterministic union-free. Set $n = k(k-1)/2$.

Define a unary $2n$-state dfa with states $0, 1, \ldots, 2n-1$, of which 0 is the initial state. The set of final states is $\{0, n, n+(k-1), n+(k-1)+(k-2), \ldots, 2n-1\}$. Each state i goes by a to state $i+1$, except for state $2n-1$ that goes to itself. Let L be a language recognized by this dfa. Since

$$n + (k-1) + (k-2) + \cdots + (k-(k-2)) = 2n-1,$$

there are $k-2$ final states greater than n, and so the dfa has k final states. Moreover, state $2n-2$ is not final. We now show that the automaton is minimal. Let i and j be two states with $i < j$. Then there exists an integer m such that by the string a^m, state j goes to state $2n-1$, while state i goes to state $2n-2$. Since state $2n-1$ is final and state $2n-2$ is not, the states i and j are inequivalent and the dfa is minimal. It turns out that every dfa for the language L must have at least k final states, and so the language L is not deterministic union-free.

To prove that the language L is union-free, we describe a 1cfp nfa for L. The only initial and final state of the nfa is state 0. Next, construct $k+n$ cycles that are pairwise disjoint, except for state 0. The length of the cycles is consequently $n, n+(k-1), n+(k-1)+(k-2), \ldots, 2n-1$, and then $2n, 2n+1, \ldots, 3n-1$. The automaton is 1cfp nfa, accepts all strings in L of length less that $2n$, as well as all strings of length at least $2n$, but no other strings since going through more than one cycle results in a string of length at least $2n$. □

The next theorem shows that deterministic union-freeness of languages does not accelerate basic regular operations. This contrasts with the results in previously studied subclasses of regular languages such as finite, unary, prefix-, suffix-, factor-, subword-free (or closed, or convex) etc. In the case of intersection and square, the known witness languages are deterministic union-free [33,27]. Slightly changed Maslov's automata [21] provide lower bounds for star and concatenation, while a modification of the hardest dfa in [17] gives a lower bound for cyclic shift. In the case of reversal, the paper [30] claims that there is a binary n-state dfa language whose reversal requires 2^n deterministic states. Although the witness automaton is one-cycle-free-path dfa, the result cannot be used because the proof is not correct. If $n = 8$, then the resulting dfa has only 252 states instead of 256, as the reader can verify using a software, for example, in [1].

Theorem 5 (State Complexity). *Let K and L be deterministic union-free regular languages over an alphabet Σ accepted by an m-state and an n-state one-cycle-free-path dfa, respectively. Then*

1. $\mathrm{sc}(K \cup L) \leqslant mn$, *and the bound is tight if* $|\Sigma| \geqslant 2$;
2. $\mathrm{sc}(K \cap L) \leqslant mn$, *and the bound is tight if* $|\Sigma| \geqslant 2$;
3. $\mathrm{sc}(K - L) \leqslant mn$, *and the bound is tight if* $|\Sigma| \geqslant 2$;

4. $\mathrm{sc}(K \oplus L) \leqslant mn$*, and the bound is tight if* $|\Sigma| \geqslant 2$*;*
5. $\mathrm{sc}(KL) \leqslant m2^n - 2^{n-1}$ *(*$m \geqslant 2, n \geqslant 3$*), and the bound is tight if* $|\Sigma| \geqslant 2$*;*
6. $\mathrm{sc}(L^2) \leqslant n2^n - 2^{n-1}$*, and the bound is tight if* $|\Sigma| \geqslant 2$*;*
7. $\mathrm{sc}(L^c) \leqslant n$*, and the bound is tight if* $|\Sigma| \geqslant 1$*;*
8. $\mathrm{sc}(L^*) \leqslant 2^{n-1} + 2^{n-2}$ *(*$n \geqslant 2$*), and the bound is tight if* $|\Sigma| \geqslant 2$*;*
9. $\mathrm{sc}(L^R) \leqslant 2^n$ *(*$n \geqslant 2$*), and the bound is tight if* $|\Sigma| \geqslant 3$*;*
10. $\mathrm{sc}(L^{shift}) \leqslant 2^{n^2+n\log n}$*. The bound* $2^{n^2+n\log n-5n}$ *can be reached if* $|\Sigma| \geqslant 4$*.*

Proof. 1.-4. The cross-product construction gives the upper bound mn. For all four operations, the bound is reached by deterministic union-free binary languages $((b^*a)^m)^*$ and $((a^*b)^n)^*$: the strings a^ib^j with $0 \leqslant i \leqslant m-1$ and $0 \leqslant j \leqslant n-1$ are pairwise inequivalent in the right-invariant congruence defined by the intersection (union, difference, symmetric difference, respectively).

5. The upper bound is $m2^n - 2^{n-1}$ [21,33]. Notice that neither the ternary witness automata in [33] nor binary witness automata in [15] are 1cfp dfa's. However, Maslov [21] claimed the result for two binary languages accepted by automata, the first of which is 1cfp dfa, while the second one can be made to be 1cfp dfa by changing its accepting state from $n-1$ to $n-2$. Since no proof is provided in [21], we recall the two automata and show that they reach the upper bound. Consider languages accepted by the 1cfp dfa's shown in Fig. 3. Construct an nfa for the concatenation of the two languages from these dfa's by adding an ε-transition from state q_{m-1} to state 0. The initial state of the nfa is state q_0, the sole accepting state is $n-2$. We first prove by induction on the size of subsets that each set $\{q_i\} \cup S$, where $0 \leqslant i \leqslant m-2$ and S is a subset of $\{0, 1, \ldots, n-1\}$, as well as each set $\{q_{m-1}\} \cup T$, where T is a subset of $\{0, 1, \ldots, n-1\}$ containing state 0, is reachable. Each singleton $\{q_i\}$ with $i \leqslant m-2$ is reached from the initial state $\{q_0\}$ by a^i. Assume the reachability of all appropriate sets of size k. Let $S = \{q_i, j_1, j_2, \ldots, j_k\}$ be a subset of size $k+1$. First, let $i = m-1$, and so $j_1 = 0$. Since the symbol a is a permutation symbol in the second dfa, we can use $j \ominus r$ to denote the state that goes to state j by a^r. Consider the set $S' = \{q_{m-2}, j_2 \ominus 1, \ldots, j_k \ominus 1\}$ of size k. The set S' is reachable by the induction hypothesis, and since S' goes to S by a, the set S is reachable as well. Now let $i \leqslant m-2$ and $j_1 = 0$. Then the set S is reached from the set $\{q_{m-1}, 0, j_2 \ominus (i+1), \ldots, j_k \ominus (i+1)\}$ by a^{i+1}. Finally, if $i \leqslant m-2$ and $j_1 > 0$, then S is reached from the set $\{q_i, 0, j_2 - j_1, j_3 - j_1, \ldots, j_k - j_1\}$ by b^{j_1}.

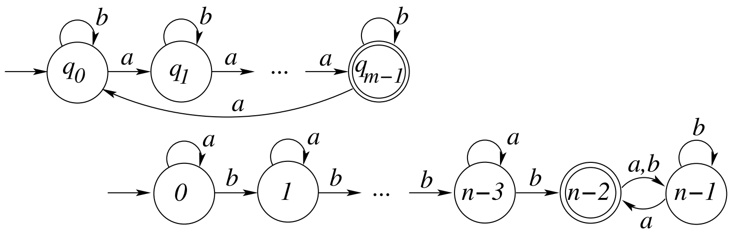


Fig. 3. The one-cycle-free-path dfa's reaching the bound $m2^n - 2^{n-1}$ on concatenation

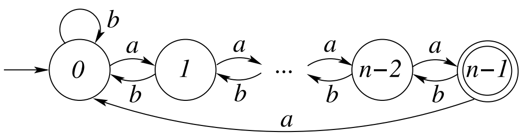


Fig. 4. The one-cycle-free-path dfa reaching the bound $2^{n-1} + 2^{n-2}$ on star

This concludes the proof of reachability. Now let $\{q_i\} \cup S$ and $\{q_j\} \cup T$ be two different reachable sets. If $i < j$, then the string $ba^{m-j-1}b^{n-2}$ distinguishes the two subsets. If $i = j$, then S and T differ in a state j, and moreover, $j > 0$ if $i = m - 1$. Then either the string b^{n-j-2} (if $j \leqslant n - 3$), or the empty string (if $j = n - 2$), or the string a (if $j = n - 1$) distinguishes the two subsets.

6. The upper bound follows from the upper bound on concatenation, and, as shown in [27], is reached by the binary language recognized by the 1cfp dfa with states $0, 1, \ldots, n-1$, of which 0 is the initial state, and $n-1$ is the sole accepting state; by a, each state i goes to state $i+1 \bmod n$, and by b, each state i goes to itself except for state 1 that goes to state 0 by b.

7. To get a dfa for the complement we only need to exchange the accepting and rejecting states. The bound is reached by the language $(a^n)^*$.

8. The upper bound is $2^{n-1} + 2^{n-2}$ [33]. The witness language in [33] is not deterministic union-free. However, Maslov [21] provides deterministic union-free witness example shown in Fig. 4. Since there is no proof in [21], we give it here. Construct an nfa for the star of the language accepted by the 1cfp dfa in Fig. 4 by adding a new initial and accepting state q_0 that goes to state 1 by a and to state 0 by b, and by adding the transition by a from state $n-2$ to state 0. The initial state $\{q_0\}$ and all singletons $\{i\}$ are reachable. Assume that all subsets of size $k-1$ containing state 0, or containing neither 0 nor $n-1$ are reachable. Let $S = \{i_1, i_2, \ldots, i_k\}$ be a subset of size k with $0 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n-1$ (and if $i_1 > 0$ then $i_k < n-1$). First, let $i_1 = 0$. Then the set S is reached from the set $\{i_2 + (n-1) - i_k - 1, i_3 + (n-1) - i_k - 1, \ldots, i_{k-1} + (n-1) - i_k - 1, n-2\}$ of size $k-1$, containing neither 0 nor $n-1$, by the string ab^{n-1-i_k}. Now let $i_1 > 0$. Then $i_k < n-1$, and the set S is reached from the set $\{0, i_2 - i_1, i_3 - i_1, \ldots, i_k - i_1\}$, which contains state 0, by a. To prove inequivalence notice that the initial (and accepting) state $\{q_0\}$ cannot be equivalent to any state not containing state $n-1$. However, the string a^n is accepted by the nfa from state $n-1$ but not from state q_0. Two different subsets of the state set of the given dfa differ in a state i, and the string a^{n-1-i} distinguishes the two subsets.

9. The reversal of a dfa language is accepted by the nfa obtained from the given dfa by reversing all transitions, making all accepting states initial, and the initial state accepting. The subset construction gives a dfa of at most 2^n states. As pointed out by Mirkin [23], the Lupanov's ternary worst-case example for nfa-to-dfa conversion in [20] is, in fact, a reversed dfa. Leiss [19] presented a ternary and a binary dfa's that reach the the upper bound. Since none of these automata is 1cfp dfa, let us consider the 1cfp dfa shown in Fig. 5. Construct the reversed nfa. Notice that in this nfa each state i goes to state $(i+1) \bmod n$

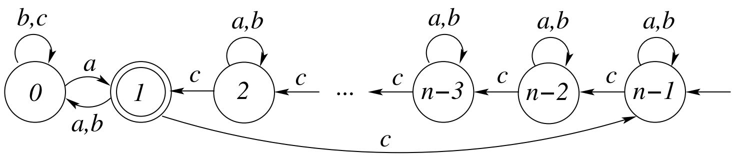


Fig. 5. The one-cycle-free-path dfa reaching the bound 2^n on reversal

by ca. It turns out that in the subset automaton, each subset not containing state 0 is reached from a subset containing state 0 by a string in $(ca)^*$. Let us show by induction on the size of subsets that each subset of the state set $\{0, 1, \ldots, n-1\}$ containing state 0 is reachable in the subset automaton. The singleton $\{0\}$ is reached from the initial state $\{1\}$ of the subset automaton by a. The subset $\{0, i_1, i_2, \ldots, i_k\}$, where $1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant n-1$, of size $k+1$ is reached from the set $\{0, i_2 - i_1 + 1, i_3 - i_1 + 1, \ldots, i_k - i_1 + 1\}$ of size k by the string $bc^{i_1 - 1}$. Finally, the empty set is reached from state $\{1\}$ by b. For inequivalence, notice that the string c^{n-1-i} is accepted by the nfa only from state i for $i = 1, 2, \ldots, n-1$, and the string ac^{n-2} only from state 0.

10. The upper bound is from [21,17]. The work [17] proves the lower bound $2^{n^2 + n\log n - 5n}$ for the language recognized by the dfa over the alphabet $\{a, b, c, d\}$ with states $0, 1, \ldots, n-1$, of which 0 is the initial state and $n-1$ is the sole accepting state, and transitions are defined as follows: By a, states 0 and $n-1$ go to itself and there is a circle $(1, 2, \ldots, n-2)$; by b, state 0 goes to itself and there is a circle $(1, 2, \ldots, n-1)$; by c, all states go to itself except for state 0 that goes to 1 and state 1 that goes to 0; by d, all states go to state 0 except for state $n-1$ that goes to state 1. This automaton is not one-cycle-free-path dfa. Therefore, let us change transitions on symbol b so that in a new dfa by b, all states go to itself except for state $n-2$ that goes to $n-1$ and state $n-1$ that goes to $n-2$. The resulting automaton is a 1cfp dfa, and moreover, the transitions by old symbol b are now implemented by the string ba. It turns out that the proof in [17] works for the new 1cfp dfa if we replace all occurrences of b in the proof by the string ba. □

5 Conclusions

We investigated union-free regular languages that can be described by regular expressions without the union operation. Using known results of Nagy [26] on characterization of automata accepting those languages, we proved some closure properties, and studied the nondeterministic state complexity of regular operations. We showed that all known upper bounds can be reached by union-free languages, except for the reversal, where the tight bound is n instead of $n+1$. We also defined deterministic union-free languages as languages recognized by deterministic one-cycle-free-path automata, and proved that they are properly included in the class of union-free languages. We examined the state complexity of quite a number of regular operations, and showed that deterministic union-freeness of languages accelerates none of them. This contrasts with the results on complexity of operations in previously studied subclasses of regular languages.

Some questions remain open. We conjecture that for the difference of two union-free languages, nfa's need $m2^n$ states, and we do not now the result on the shuffle of deterministic union-free languages. A description of deterministic union-free regular languages in terms of regular expressions or grammars, as well as the case of unary union-free languages, is of interest too.

References

1. JFLAP, `http://www.jflap.org/`
2. Afonin, S., Golomazov, D.: Minimal union-free decompositions of regular languages. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 83–92. Springer, Heidelberg (2009)
3. Aho, A.V., Ullman, J.D., Yannakakis, M.: On notions of information transfer in VLSI circuits. In: Proc. 15th Annual ACM Symposium on the Theory of Computing, pp. 133–139 (1983)
4. Birget, J.-C.: Intersection and union of regular languages and state complexity. Inform. Process. Lett. 43, 185–190 (1992)
5. Birget, J.-C.: Partial orders on words, minimal elements of regular languages, and state complexity. Theoret. Comput. Sci. 119, 267–291 (1993), `http://clam.rutgers.edu/~birget/poWordsERR.ps`
6. Bordihn, H., Holzer, M., Kutrib, M.: Determination of finite automata accepting subregular languages. Theoret. Comput. Sci. 410, 3209–3222 (2009)
7. Câmpeanu, C., Culik II, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 60–70. Springer, Heidelberg (2001)
8. Câmpeanu, C., Salomaa, K., Yu, S.: Tight lower bound for the state complexity of shuffle of regular languages. J. Autom. Lang. Comb. 7, 303–310 (2002)
9. Crvenković, S., Dolinka, I., Ésik, Z.: On equations for union-free regular languages. Inform. and Comput. 164, 152–172 (2001)
10. Domaratzki, M., Kisman, D., Shallit, J.: On the number of distinct languages accepted by finite automata with n states. J. Autom. Lang. Comb. 7, 469–486 (2002)
11. Glaister, I., Shallit, J.: A lower bound technique for the size of nondeterministic finite automata. Inform. Process. Lett. 59, 75–77 (1996)
12. Han, Y.-S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. Theoret. Comput. Sci. 410, 2537–2548 (2009)
13. Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. Internat. J. Found. Comput. Sci. 14, 1087–1102 (2003)
14. Hromkovič, J.: Communication complexity and parallel computing. Springer, Heidelberg (1997)
15. Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation. Internat. J. Found. Comput. Sci. 16, 511–529 (2005)
16. Jirásková, G.: State complexity of some operations on binary regular languages. Theoret. Comput. Sci. 330, 287–298 (2005)
17. Jirásková, G., Okhotin, A.: State complexity of cyclic shift. Theor. Inform. Appl. 42, 335–360 (2008)

18. Jirásková, G., Okhotin, A.: Nondeterministic state complexity of positional addition. In: Dassow, J., Pighizzini, G., Truthe, B. (eds.) 11th International Workshop on Descriptional Complexity of Formal Systems, pp. 199–210. Otto-von-Guericke-Universität, Magdeburg (2009)
19. Leiss, L.: Succint representation of regular languages by boolean automata. Theoret. Comput. Sci. 13, 323–330 (1981)
20. Lupanov, O.B.: Über den Vergleich zweier Typen endlicher Quellen. Probl. Kybernetik 6, 328–335 (1966); Translation from Probl. Kibernetiki 9, 321–326 (1963)
21. Maslov, A.N.: Estimates of the number of states of finite automata. Sov. Math., Dokl. 11, 1373–1375 (1970); Translation from Dokl. Akad. Nauk SSSR 194, 1266–1268 (1970)
22. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Proc. 12th IEEE Symp. on Switching and Automata Theory, pp. 188–191 (1971)
23. Mirkin, B.G.: On dual automata. Kibernetika 1, 7–10 (1966)
24. Moore, F.R.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Trans. Comput. 20, 1211–1219 (1971)
25. Nagy, B.: A normal form for regular expressions. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) CDMTCS-252 report, supplemental material for DLT 2004, pp. 53–62. University of Auckland, New Zealand (2004)
26. Nagy, B.: Union-free regular languages and 1-cycle-free-path automata. Publ. Math. Debrecen 68, 183–197 (2006)
27. Rampersad, N.: The state complexity of L^2 and L^k. Inform. Process. Lett. 98, 231–234 (2006)
28. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Proc. 10th Annual ACM Symposium on the Theory of Computing, pp. 275–286 (1978)
29. Salomaa, A.: Formal languages. Academic Press, New York (1973)
30. Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages. Theoret. Comput. Sci. 320, 315–329 (2004)
31. Sipser, M.: Introduction to the theory of computation. PWS Publishing Company, Boston (1997)
32. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, ch. 2, vol. I, pp. 41–110. Springer, Heidelberg (1997)
33. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoret. Comput. Sci. 125, 315–328 (1994)

Schema for Parallel Insertion and Deletion*

Lila Kari and Shinnosuke Seki

Department of Computer Science, University of Western Ontario,
London, Ontario, N6A 5B7, Canada
{lila,sseki}@csd.uwo.ca

Abstract. We propose a general framework for parallel insertion/deletion operations based on p-schemata. A p-schema is a set of tuples of words. When being used for parallel insertion of a language into a word, an element of a p-schema specifies how to split the given word into factors between which the insertion of the language will take place. Parallel deletion based on a p-schema is defined as an "inverse" operation of parallel insertion based on the p-schema. Several well-known language operations are particular cases of p-schema-based insertions or deletions: catenation, Kleene star, reverse catenation, sequential insertion, parallel insertion, insertion next to a given letter, contextual insertion, right and left quotient, sequential deletion, parallel deletion. Additional operations that can be defined using p-schemata include contextual parallel insertion, as well as parallel insertion (deletion) of exactly n words, at most n words, an arbitrary number of words. We also consider the decidability and undecidability of existence of solutions of language equations involving p-schema-based parallel insertion/deletion.

1 Introduction

Since Adleman's success [1] in solving the Directed Hamiltonian Path Problem purely by biological means, which threw new light on fundamental research on operations in formal language theory, various bio-operations have been intensively investigated. Examples include hairpin inversion [2], circular insertion/deletion [3], excisions of loop, hairpin, and double-loop [4], and contextual insertion/deletion [5], to name a few.

The fact that one can experimentally implement in the laboratory some variants of insertions and deletions into/from DNA sequences [6], and use these as the sole primitives for DNA computation, gives practical significance to the research on insertion and deletion. Contextual insertion and deletion are also of theoretical interest because they have been proved to be Turing-universal [5]. In this paper, we will parallelize contextual insertion and deletion. For words x and

* We thank anonymous referees for their valuable comments on the earlier version of this paper. In particular, we are indebted to one of them for the comparison between our framework and I-shuffle. This research was supported by The Natural Sciences and Engineering Research Council of Canada Discovery Grant R2824A01 and Canada Research Chair Award to L.K.

y, the (x, y)-contextual insertion of a language L into a word w [5] results in the language

$$\bigcup_{w_1, w_2 \text{ with } w = w_1 x y w_2} w_1 x L y w_2.$$

In other words, one considers all the possibilities of cutting w into two segments, such that the first segment ends with x and the second segment begins with y, and for each such possibility L is inserted between these segments. This operation suggests that for any positive integer n, an n-tuple $(w_1, w_2, \ldots, w_n)$ of words may be used to control the parallel insertion of $n-1$ instances of L into $w = w_1 w_2 \cdots w_n$ to generate the language $w_1 L w_2 L \cdots L w_{n-1} L w_n$. A set of such tuples is called a *parallel operation schema* or *p-schema* for short, and we call the parallel insertion thus determined *parallel insertion based on the p-schema.* A p-schema can be used to control not only parallel insertion but parallel deletion as well. Parallel deletion of L from a word w based on a given n-tuple $(u_1, u_2, \ldots, u_n)$ deletes $n-1$ non-overlapping elements of L from w so as to leave this n-tuple, and concatenates them to generate the word $u = u_1 u_2 \cdots u_n$. As we shall see in Section 3, various well-known sequential as well as parallel operations (catenation, Kleene star, reverse catenation, sequential insertion, parallel insertion, insertion next to a given letter, contextual insertion, right and left quotient, sequential deletion, parallel deletion) are special instances of parallel operations based on p-schemata. Additional operations that can be defined using p-schemata are contextual parallel insertion, as well as parallel insertion (deletion) of *exactly n words, at most n words, an arbitrary number of words.*

Besides being proper generalizations of existing language operations, parallel operations based on p-schemata lead to some interesting results when studied in the context of language equations. Equations of the form $X_1 \diamond X_2 = X_3$ have been intensely studied in the literature, where $\diamond$ is a binary operation on languages, and some of X_1, X_2, X_3 are fixed languages, while the others are unknowns (see, e.g., [5,7,8,9,10,11,12,13,14]). In this paper, we focus on such language equations with $\diamond$ being p-schema-based insertion or deletion. Since these two operations are parameterized by p-schemata, we can also consider the problem of deciding whether $L_1 \diamond_X L_2 = L_3$ has a solution, i.e., whether there exists a p-schema F such that parallelly inserting L_2 into (deleting from) L_1 based on F results in L_3.

In general, procedures do not exist for solving such equations when they involve a context-free language. Therefore, we focus on solving equations of the form (1) $X \longleftarrow_F R_2 = R_3$, (2) $R_1 \longleftarrow_X R_2 = R_3$, (3) $R_1 \longleftarrow_F X = R_3$, and their p-schema-based deletion variants, where all of R_1, R_2, R_3, F are regular[1]. Among these equations, the equations of the first or second form can be solved using the technique of [14]. The application of this technique presumes the property that the union of all the solutions to the given equation is the unique maximal solution. As we shall see, the third-type equations do not have this property,

[1] By catenating words in a tuple of words via a special symbol #, we can naturally associate a set of tuples of words with a language, and as such we can establish a Chomsky-hierarchy for the sets of tuples of words.

that is, they may have multiple maximal solutions. Algorithms to solve these equations are one of the main contributions of this paper. Our algorithms work not only as a procedure to decide the existence of solutions, but as a procedure to enumerate all maximal solutions (Theorems 6 and 7). Moreover, combining these algorithms with the algorithms to solve the equations of the first or second form (outlined in Section 5) enables us to solve two-variables equations of the form $X \leftarrow_F Y = R_3$ (Theorem 9), $R_1 \leftarrow_X Y = R_3$ (Theorem 10), and $R_1 \rightarrowtail_X Y = R_3$ (Theorem 11). The proposed algorithms can be modified to also solve inequality (set inclusion) variants of the above-mentioned equations with maximality condition on variables.

2 Preliminaries

By Σ we denote a finite alphabet, and the set of words over Σ is denoted by Σ^* which includes the *empty word* λ. For a given word w, its length is denoted by $|w|$, and its reversal is denoted by w^R. For an integer $n \geq 0$, Σ^n, $\Sigma^{\leq n}$, and $\Sigma^{\geq n}$ denote the sets of all words of length *exactly* n, *at most* n, and *at least* n, respectively. A word u is called a *factor* (*prefix*, *suffix*) of a word w if $w = xuy$ (resp. $w = uy$, $w = xu$) for some words x, y. Let us denote the set of all prefixes (suffixes) of w by $\mathrm{Pref}(w)$ (resp. $\mathrm{Suf}(w)$). For a language $L \subseteq \Sigma^*$, $L^c = \Sigma^* \setminus L$.

Regular languages are specified by (non-deterministic) finite automata (NFA) $A = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, $s \in Q$ is the start state, $F \subseteq Q$ is a set of final states, and δ is a map from $Q \times \Sigma$ to 2^Q. For notational convenience, we employ the notation NFA also to denote a language accepted by an NFA (we use this slight abuse of notation for other kinds of acceptors). The family of languages accepted by NFAs is denoted by REG. An NFA is said to be *deterministic* if δ is a function. The deterministic property of a machine is stated explicitly by using the capital letter D. A language is said to be *effectively regular* if there exists an algorithm to construct an NFA which accepts this language.

A characterization of languages can be given in terms of *syntactic semigroups*. For a language $L \subseteq \Sigma^*$, there exists a maximal congruence $\equiv_L$ which saturates L (i.e., L is a union of equivalence classes). This is called the *syntactic congruence* of L, which is formally defined as follows: for $u, v \in \Sigma^*$,

$$u \equiv_L v \iff \text{for any } x, y \in \Sigma^*,\ xuy \in L \text{ if and only if } xvy \in L.$$

For a word $w \in \Sigma^*$, a set $[w]_{\equiv_L} = \{u \in \Sigma^* \mid w \equiv_L u\}$ is called an *equivalence class* with w as its representative. The number of equivalence classes is called the *index* of $\equiv_L$.

Theorem 1 ([15]). *Let $L \subseteq \Sigma^*$ be a language. The index of $\equiv_L$ is finite if and only if L is regular.*

For technical reasons, we define a function called *saturator with respect to a language* L_1. Let σ_{L_1} be a function from a word w into the equivalence class $[w]_{\equiv_{L_1}}$. The saturator w.r.t. L_1 is its extension defined as $\sigma_{L_1}(L) = \bigcup_{w \in L} [w]_{\equiv_{L_1}}$.

We can choose an arbitrary word in $[w]_{\equiv_L}$ as a representative of this class. By taking a representative from every class, we can construct a subset of Σ^* called a *complete system of representatives* of $\Sigma^*/\equiv_L$. In particular, for a regular language R, there exists a complete system of representatives which is computable. Let $A = (Q, \Sigma, \delta, s, F)$ be the (unique) minimal-DFA for R. Then $u \equiv_R v$ if and only if $\delta(q, u) = \delta(q, v)$ for any $q \in Q$. Hence, the index of $\equiv_L$ is at most $|Q|^{|Q|}$.

Theorem 2. *Let R be a regular language and $A = (Q, \Sigma, \delta, s, F)$ be the min-DFA for R. Each equivalence class in $\Sigma^*/\equiv_R$ is regular, and contains a word of length at most $|Q|^{|Q|}$.*

Corollary 1. *For a regular language R, there exists a computable complete system of representatives of $\Sigma^*/\equiv_L$.*

3 Parallel Insertion and Deletion Schema

Imagine that we will insert a language L into a word u in parallel. Let $\prod_{i=1}^{n} \Sigma^*$ be the Cartesian product of Σ^* with itself n times; that is to say, the set of all n-tuples of words. Let $\mathfrak{F} = \bigcup_{n\geq 1} \underbrace{\Sigma^* \times \Sigma^* \times \cdots \times \Sigma^*}_{n \text{ times}}$. A subset F of $\mathfrak{F}$ can be used to control the parallel insertion of a language L in a sense that if $(u_1, u_2, \ldots, u_n) \in F$, then the word $u = u_1 u_2 \cdots u_n$ is split in the manner dictated by the n-tuple in F, and L is inserted between u_i and u_{i+1} for all $1 \leq i < n$ to generate the language $u_1 L u_2 L \cdots u_{n-1} L u_n$. The set can be also used to control a parallel deletion. For this intended end-usage, we call a subset of $\mathfrak{F}$ a *parallel schema*, or shortly *p-schema*, *over* Σ.

As abstracted above, a p-schema F enables us to define the *(parallel) insertion* $\hookleftarrow_F$ as: for a word $u \in \Sigma^*$ and a language $L \subseteq \Sigma^*$,

$$u \hookleftarrow_F L = \bigcup_{n\geq 1, u = u_1\cdots u_n, (u_1,\ldots,u_n)\in F} u_1 L u_2 L \cdots u_{n-1} L u_n.$$

Note that an n-tuple in F parallel-inserts $n-1$ words from L into u. Similarly, we define the *(parallel) deletion* $\rightarrowtail_G$ *based on a p-schema* G as: for a word $w \in \Sigma^*$ and a language $L \subseteq \Sigma^*$,

$$\begin{aligned} w \rightarrowtail_G L = \{u_1 \cdots u_n \mid n \geq 1, x_1, \ldots, x_{n-1} \in L, \\ (u_1, \ldots, u_n) \in G, w = u_1 x_1 u_2 x_2 \cdots u_{n-1} x_{n-1} u_n\}, \end{aligned}$$

These operations are extended to languages in a conventional manner: for a language L_1, $L_1 \hookleftarrow_F L = \bigcup_{u\in L} u \hookleftarrow_F L$ and $L_1 \rightarrowtail_G L = \bigcup_{w\in L} w \rightarrowtail_G L$.

Many of the well-known operations are particular cases of p-schema-based operations. We list instances of p-schema-based insertion:

catenation	$F_{\text{cat}} = \Sigma^* \times \lambda$,
reverse catenation	$F_{\text{rcat}} = \lambda \times \Sigma^*$,
(sequential) insertion	$F_{\text{sins}} = \Sigma^* \times \Sigma^*$,
parallel insertion	$F_{\text{pins}} = \bigcup_{n\geq 0}(\lambda \times \prod_{i=1}^{n} \Sigma \times \lambda)$.

Deletions based on F_{cat}, F_{rcat}, F_{sins}, and F_{pins} correspond to right and left quotient, (sequential) deletion, and parallel deletion, respectively.

Parallel insertion (deletion) of *exactly n words*, *at most n words*, or *arbitrary number of words* are important instances of insertion (deletion) based on:

$$F_{\mathrm{pins}(n)} = \prod_{i=1}^{n+1} \Sigma^*, \qquad F_{\mathrm{pins}(\le n)} = \bigcup_{i=0}^{n} F_{\mathrm{pins}(i)}, \qquad F_* = \bigcup_{i=0}^{\infty} F_{\mathrm{pins}(i)},$$

respectively. Using for instance F_*, one can implement Kleene-star, the most well-studied unary operation in formal language theory, as $L^* = \lambda \leftarrow_{F_*} L$.

The p-schemata introduced so far are "syntactic" in a sense, while many of semantic (letter-sensitive) operations are known. For a letter $b \in \Sigma$, *parallel insertion next to b* [13] is the insertion based on $F_{\mathrm{pinsb}} = \{(u_1, u_2, \ldots, u_n) \mid n \ge 1, u_1, \ldots, u_n \in (\Sigma \backslash \{b\})^* b\}$. For a context $C \subseteq \Sigma^* \times \Sigma^*$, *$C$-contextual (sequential) insertion* [5] is the insertion based on $F_{\mathrm{scins}}(C) = \bigcup_{(x,y) \in C} \Sigma^* x \times y \Sigma^*$. This operation is naturally parallelized as *C-contextual parallel insertion* with the p-schema $F_{\mathrm{pcins}}(C) = \{(u_1, \ldots, u_n) \mid n \ge 1, \forall 1 \le i < n, \big(\mathrm{Suf}(u_i) \times \mathrm{Pref}(u_{i+1})\big) \cap C \ne \emptyset\}$.

It may be worth noting that the descriptional powers of our framework and of I-shuffle proposed by Domaratzki, Rozenberg, and Salomaa [16] (a generalization of semantic shuffle proposed by Domaratzki [11]) are incomparable. Indeed, only I-shuffle can specify contexts not only on the left operand but also on the right operand, while p-schema-based operations can insert/delete multiple copies of right operand. Thus, insertion/deletion based on a p-schema which contains 2-tuples and/or 1-tuples is a special instance of I-shuffle.

4 Hierarchy of p-Schemata and Closure Properties

In this section, we investigate closure properties of abstract families of acceptors augmented with reversal-bounded counters under the p-schema-based operations. Such an acceptor was proposed by Ibarra [17] as the *counter machine*. For $k \ge 0$, let NCM(k) be the class of NFAs augmented with k reversal-bounded counters, and NCM be the union of such classes over all k's. By augmenting an NCM with an unrestricted pushdown stack, we obtain a *non-deterministic pushdown counter machine* (NPCM). For $k \ge 0$, let NPCM(k) be an NPCM with k reversal-bounded counters. DCM(k), DPCM(k), DCM, and DPCM are the deterministic analogs of NCM(k), NPCM(k), NCM, and NPCM. A desirable property specific to these deterministic classes is proved by Ibarra [17] as follows:

Theorem 3. *For $L_1 \in$ DCM and $L_2 \in$ DPCM, it is decidable whether $L_1 = L_2$.*

It is natural to encode a tuple $(u_1, u_2, \ldots, u_n)$ as a word $u_1 \# u_2 \# \cdots \# u_n$ using a special symbol #. Denoting this (one-to-one) encoding by ψ, we can encode a p-schema F as $\psi(F) = \{\psi(f) \mid f \in F\}$. Furthermore, we say that a p-schema F is *in a language class $\mathcal{L}$* if $\psi(F) \in \mathcal{L}$. For instance, F is *regular* if $\psi(F)$ is a regular language over $\Sigma \cup \{\#\}$.

First of all, we prove that NCM is closed under insertion/deletion based on an NCM p-schema with an analysis on how many reversal-bounded counters are required.

Proposition 1. *Let $L_1 \in \mathrm{NCM}(k_1)$, $L_2 \in \mathrm{NCM}(k_2)$, and F be a p-schema in $\mathrm{NCM}(k_\psi)$. Then both $L_1 \hookleftarrow_F L_2$ and $L_1 \rightarrowtail_F L_2$ are in $\mathrm{NCM}(k_1 + k_2 + k_\psi)$.*

Proof. We show only a construction of an NCM M for $L_1 \hookleftarrow_F L_2$, and omit the construction of an NCM for $L_1 \rightarrowtail_F L_2$.

Let M_1, M_2, M_ψ be NCMs with k_1, k_2, k_ψ counters for $L_1, L_2, \psi(F)$, respectively. M expects its input to be of the form $u_1x_1u_2x_2 \cdots x_{n-1}x_n$ for some integer $n \geq 1$, $u_1u_2 \cdots u_n \in L_1$, $x_1, x_2, \ldots, x_{n-1} \in L_2$, and $u_1\#u_2\# \cdots \#u_n \in \psi(F)$. M simulates M_1 and M_ψ simultaneously. Guessing non-deterministically that the prefix $u_1x_1 \cdots x_{i-1}u_i$ has been read, M pauses the simulation of both M_1 and M_ψ and instead activates the simulation of M_2 on x_i after having M_ψ make a #-transition. When M_2 is in one of its accepting states, M non-deterministically resumes the simulation of M_1 and M_ψ on the suffix $u_{i+1}x_{i+1} \cdots x_{n-1}u_n$ of the input. The simulation of M_2 is initialized every time it is invoked. □

In the previous proof, the resulting NCM does not require more counters than required for the operands, and thus, the case of $k_1 = k_2 = k_\psi = 0$, i.e., where all languages are regular, is a corollary.

Corollary 2. *For regular languages R_1, R_2 and a regular p-schema F, both $R_1 \hookleftarrow_F R_2$ and $R_1 \rightarrowtail_F R_2$ are effectively regular.*

We can prove an analogous result of Proposition 1 for NPCM. By enlarging some of the respective language classes which L_1, L_2, and F belong to, up to NPCM, we can ask whether or not $L_1 \hookleftarrow_F L_2$ or $L_1 \rightarrowtail_F L_2$ are in NPCM. In the following we only address some non-closure properties of DPCM with implications to language equation solvability in the next section.

Let us define the *balanced language* L_b over $\Sigma = \{a, \$\}$ as follows:

$$L_b = \{a^{i_1}\$a^{i_2}\$ \cdots \$a^{i_k}\$a^{i_{k+1}}\$ \cdots \$a^{i_n} \mid n \geq 2,\ i_1, \ldots, i_n \geq 0 \text{ and}$$
$$\exists 1 \leq k < n \text{ such that } i_1 + i_2 + \cdots + i_k = i_{k+1} + \cdots + i_n\}.$$

In other words, a word in L_b has a central marker \$ so that the number of a's to the left of this marker is equal to the number of a's to its right. For $L_1 = \{a^n\$a^n \mid n \geq 1\}$, we obtain $L_1 \hookleftarrow_{F_*} \$ = L_b$. Recall the definition of F_*; in this case it scatters an arbitrary number of \$'s into any word in L_1. We can generate L_b also by deletion. Let $L_1 = \bigcup_{n \geq 0}(a^n \sqcup\!\sqcup \$^*)\$\#(a^n \sqcup\!\sqcup \$^*)$ and $F = \{a, \$\}^* \times \{a, \$\}^*$, where $\sqcup\!\sqcup$ denotes shuffle operation. Then $L_b = L_1 \rightarrowtail_F \#$. These L_1's are DCM(1). L_b is clearly in NCM(1) because the non-determinism makes it possible for the reversal-bounded counter to guess when it should transit into its decrementing mode. In contrast, L_b is proved not to be DPCM (see, e.g., [18]). Consequently we have the following non-closure property.

Proposition 2. *There exist $L_1 \in \mathrm{DCM}(1)$, a regular p-schema F, and a singleton language L_2 such that $L_1 \hookleftarrow_F L_2 \notin \mathrm{DPCM}$.*

Proposition 3. *There exist $L_1 \in \mathrm{DCM}(1)$, a regular p-schema F, and a singleton language L_2 such that $L_1 \rightarrowtail_F L_2 \notin \mathrm{DPCM}$.*

By swapping the roles of L_1 and F in the above example, we can also obtain the following non-closure property.

Proposition 4. *There exist a regular language R_1, a singleton language L_2, and a $\mathrm{DCM}(1)$ p-schema F such that $R_1 \rightarrowtail_F L_2 \notin \mathrm{DPCM}$.*

5 Language Equations with *p*-Schemata-Based Operations

In this section, we consider language equations involving p-schema-based operations. The simplest equations to be studied are one-variable equations of the form $X \leftarrowtail_F L_2 = L_3$, $L_1 \leftarrowtail_X L_2 = L_3$, $L_1 \leftarrowtail_F X = L_3$, and their deletion variants. Such equations with special instances of p-schema-based operations (catenation, insertion, etc.) as well as incomparable operations (shuffle, etc.) have been intensively studied for the last decades [8,9,11,12,13,14,19]. These papers mainly dealt with language equations with the property that the union of all their solutions (if any) is also their solution (maximum solution). For instance, if $XL = R$ and $YL = R$, then $(X \cup Y)L = R$. For such equations, we can employ a technique established in [14]; assuming a given equation has a solution, firstly construct the candidate of its maximum solution, and then substitute it into the equation to check whether it is actually a solution. Since $X \leftarrowtail_F L_2 = L_3$, $L_1 \leftarrowtail_X L_2 = L_3$, and their deletion variants have this property, this technique can solve these equations. We will now see how to construct the candidate for each.

In [9], Cui, Kari, and Seki defined the left-l-inverse relation between operations as: the operation $\bullet$ is *left-l-inverse* of the operation $\circ$ if for any words $u, w \in \Sigma^*$ and any language $L \subseteq \Sigma^*$, $w \in u \circ L \iff u \in w \bullet L$. This is a symmetric relation. By definition, insertion and deletion based on the same p-schema are left-l-inverse to each other. There they proved that for operations $\circ, \bullet$ which are left-l-inverse to each other, if $X \circ L_2 = L_3$ has a solution, then $(L_3^c \bullet L_2)^c$ is its maximum solution.

Theorem 4. *For regular languages R_2, R_3 and a regular p-schema F, the existence of a solution to both $X \leftarrowtail_F R_2 = R_3$ and $X \rightarrowtail_F R_2 = R_3$ is decidable.*

Proof. Both $(R_3^c \rightarrowtail_F R_2)^c \leftarrowtail_F R_2$ and $(R_3^c \leftarrowtail_F R_2)^c \rightarrowtail_F R_2$ are regular according to Corollary 2 and the fact that REG is closed under complement. Now it suffices to employ Theorem 3 for testing the equality. □

For $L_1 \leftarrowtail_X L_2 = L_3$, the candidate is $F_{\max} = \{f \in \mathfrak{F} \mid L_1 \leftarrowtail_f L_2 \subseteq L_3\}$. For $L_1 \rightarrowtail_X L_2 = L_3$, $F_{\max}$ should be rather $\{f \in \mathfrak{F} \mid L_1 \rightarrowtail_f L_2 \subseteq L_3\}$. When L_1, L_2, L_3 are all regular, we can construct an NFA for $\psi(\mathfrak{F} \setminus F_{\max})$, which is equal to $(\Sigma \cup \#)^* \setminus \psi(F_{\max})$. A similar problem was studied in [12], and our construction originates from theirs. As such, the proof of next result is omitted.

Theorem 5. *For regular languages R_1, R_2, R_3, the existence of a solution to both $R_1 \leftarrowtail_X R_2 = R_3$ and $R_1 \rightarrowtail_X R_2 = R_3$ is decidable.*

5.1 Solving $L_1 \leftarrowtail_F X = L_3$

In contrast, the equations $L_1 \leftarrowtail_F X = L_3$ and $L_1 \rightarrowtail_F X = L_3$ may not have a maximum solution. For example, let $L_1 = L_3 = \{a^{2n} \mid n \geq 1\}$, and $F = F_{\text{pins}(2)} \cup F_{\text{pins}(0)}$. Both $L_{\text{even}} = \{a^{2m} \mid m \geq 0\}$ and $L_{\text{odd}} = \{a^{2m+1} \mid m \geq 0\}$ are (maximal) solutions to $L_1 \leftarrowtail_F X = L_3$. On the other hand, $L_1 \leftarrowtail_F (L_{\text{even}} \cup L_{\text{odd}})$ can generate a^3, which is not in L_3. For deletion, let $F = \{(\lambda, aba), (\lambda, \lambda, \lambda), (aba, \lambda)\}$, and $L_1 = \{ababa\}$. Then $L_1 \rightarrowtail_F \{ab\} = L_1 \rightarrowtail_F \{ba\} = \{aba\}$, but $L_1 \rightarrowtail_F \{ab, ba\} = \{aba, a\}$. These examplify that we cannot apply the previously-mentioned approach to solving language equations with the second operand being unknown.

We propose an alternative approach based on an idea from Conway (Chapter 6 of [8]) to solve $f(\Sigma \cup \{X_1, X_2, \ldots\}) \subseteq R$, where f is a regular function over Σ and variables $X_1, X_2, \ldots$, and R is a regular language. The idea shall be briefly explained in terms of p-schema-based operations in order to step into more general cases than the case when all the involved languages are regular.

Lemma 1. *Let L, L_1 be languages. Then $(L_1 \leftarrowtail_F (L_2 \cup w)) \cap L \neq \emptyset$ if and only if $(L_1 \leftarrowtail_F (L_2 \cup [w]_{\equiv_L})) \cap L \neq \emptyset$ for any word w and language L_2.*

By replacing L in this lemma with L_3^c, we can see that if $L_1 \leftarrowtail_F (L_2 \cup w) \subseteq L_3$, then $L_1 \leftarrowtail_F (L_2 \cup [w]_{\equiv_{L_3}}) \subseteq L_3$. Thus, it makes sense to introduce the notion of a syntactic solution. For a language L, we say that a solution to a one-variable language equation is *syntactic with respect to* L if it is a union of equivalence classes in $\Sigma^* / \equiv_L$.

Proposition 5. *For languages L_1, L_3, the equation $L_1 \leftarrowtail X = L_3$ has a solution if and only if it has a syntactic solution with respect to L_3.*

Thus, in order to determine whether $L_1 \leftarrowtail_F X = L_3$ has a solution, it suffices to test whether it has a syntactic solution. On condition that this test can be executed, this problem becomes decidable. If L_3 is regular, then the number of candidates of syntactic solution is finite (Theorem 1), and they are regular (Theorem 2). Let ß $= \{\sigma_{R_3}(L) \mid L \subseteq \Sigma^*\}$, the set of all candidates of syntactic solution. A pseudocode to solve $L_1 \leftarrowtail_F X = R_3$ is given below:

Algorithm to solve $L_1 \leftarrowtail_F X = R_3$

```
1. Order the elements of ß in some way (let us denote the $i$-th element of
   ß by ß[$i$]).
2. for each $1 \leq i \leq |$ß$|$, test whether $L_1 \leftarrowtail_F$ ß[$i$] is equal to $R_3$.
```

With the further condition that L_1 and F are chosen so that any language obtained by substituting a candidate into $L_1 \leftarrowtail_F X$ is comparable with R_3 for equality, this algorithm becomes executable. One such condition of significance is that both L_1 and F are regular. In this case, the algorithm, Theorem 3, and Corollary 2 lead us to the next theorem, which is stronger than decidability. It should be noted that maximal solutions are syntactic.

Theorem 6. *For regular languages R_1, R_3 and a regular p-schema F, the set of all syntactic solutions to $R_1 \hookleftarrow_F X = R_3$ is computable.*

The regularity of R_3 is necessary for the algorithm to work, whereas such condition is not imposed on L_1. If a condition on L_1, F under which $L_1 \hookleftarrow_F ß[i] \in$ DPCM for any $1 \leq i \leq |ß|$ were found, we could solve $L_1 \hookleftarrow_F X = R_3$ under it using Theorem 3. This is an unsettled question, but as suggested in Proposition 2, weakening the condition on L_1 slightly can make $L_1 \hookleftarrow_F X$ non-DPCM. It is probably more promising to broaden the class of F.

5.2 Solving $L_1 \rightarrowtail_F X = L_3$

Let us continue the investigation on the existence of right operand by changing the operation to p-schema-based deletion.

Lemma 2. *Let L_1 be a language. Then $L_1 \rightarrowtail_F (\{w\} \cup L_2) = L_1 \rightarrowtail_F ([w]_{\equiv_{L_1}} \cup L_2)$ for any word w, language L_2, and a p-schema F.*

Proof. Let $u \in L_1 \rightarrowtail_F ([w]_{\equiv_{L_1}} \cup L_2)$; that is, there exist $v \in L_1$, $n \geq 0$, $(u_1, u_2, \ldots, u_{n+1}) \in F$, and $x_1, \ldots, x_n \in [w]_{\equiv_{L_1}} \cup L_2$ such that $u = u_1 u_2 \cdots u_{n+1}$ and $v = u_1 x_1 u_2 x_2 \cdots u_n x_n u_{n+1}$. Now on v if $x_i \in [w]_{\equiv_{L_1}}$, then we replace x_i with w, and this process converts v into a word v'. Note that this replacement process guarantees that $v' \in L_1$ because the replaced factors are equal to w with respect to the syntactic congruence of L_1. Moreover, $u \in v' \rightarrowtail_F (\{w\} \cup L_2)$. Thus, $L_1 \rightarrowtail_F (\{w\} \cup L_2) \supseteq L_1 \rightarrowtail_F ([w]_{\equiv_{L_1}} \cup L_2)$. □

This lemma provides us with two approaches to determine whether a given equation with p-schema-based deletion has a solution. The first approach is based on syntactic solutions. Given a language L_2, Lemma 2 implies that $L_1 \rightarrowtail_F L_2 = L_1 \rightarrowtail_F \sigma_{L_1}(L_2)$. Therefore, as in the case of insertion, the existence of a solution to $L_1 \rightarrowtail_F X = L_3$ is reduced to that of its syntactic solutions, but with respect to L_1 (not L_3). Moreover, maximal solutions are syntactic.

Proposition 6. *For languages L_1, L_3 and a p-schema F, the equation $L_1 \rightarrowtail_F X = L_3$ has a solution if and only if it has a syntactic solution with respect to L_1. Furthermore, its maximal solution (if any) is syntactic.*

With a straightforward modification, the algorithm presented in Sect. 5.1 can be used to output all syntactic solutions to $R_1 \rightarrowtail_F X = L_3$ with F being a regular p-schema. Thus, we have the following result, analogous to Theorem 6.

Theorem 7. *For a regular language R_1, $L_3 \in$ DPCM, and a regular p-schema F, the set of all syntactic solutions to $R_1 \rightarrowtail_F X = L_3$ is computable.*

Note that even if L_3 is DPCM, the equation above is solvable due to Corollary 2 and Theorem 3.

The existence of the second approach provided by Lemma 2 is due to the essential difference between Lemma 2 and its analog for insertion (Lemma 1).

A word obtained by deleting some words in L_2 from a word in L_1 can be also obtained by deleting their representatives in a complete system of representatives with respect to L_1 from the word in L_2 based on the same schema; this is not true for insertion. Since its choice is arbitrary, we fix $\mathfrak{R}(L_1)$ to be the set of smallest words according to the lexicographical order in each equivalence class. We say that a solution to $L_1 \rightarrowtail_F X = L_3$ is *representative* if it is a subset of $\mathfrak{R}(L_1)$.

Proposition 7. *For languages L_1, L_3 and a p-schema F, the equation $L_1 \rightarrowtail_F X = L_3$ has a solution if and only if it has a representative solution.*

If L_1 is regular, then $\mathfrak{R}(L_1)$ is a finite computable set due to Theorem 1 and Corollary 1, and hence, our argument based on representative solution amounts to the second approach.

Theorem 8. *For a regular language R_1, $L_3 \in$ DPCM, and a regular p-schema F, the set of all representative solutions of $R_1 \rightarrowtail_F X = L_3$ is computable.*

With Theorem 1, Lemma 2 also leads us to a corollary about the number of distinct languages obtained by p-schema-based deletion from a regular language. Namely, given a regular language R_1 and a p-schema F, there exist at most a finite number of languages which can be represented in the form $R_1 \rightarrowtail_F L_2$ for some language L_2. This result is known for sequential deletion [13].

5.3 Solving Two-Variables Language Equations and Inequalities

There is one thing which deserves explicit emphasis: the set of all candidates of syntactic solutions is solely determined by only one of L_3, L_1, and does not depend on the other or F at all. This property paves the way to solving two-variables language equations of the form $X \leftarrowtail_F Y = L_3$, $L_1 \leftarrowtail_X Y = L_3$, and $L_1 \rightarrowtail_X Y = L_3$. The first equation with $F = F_{\text{cat}}$ (catenation) has been investigated under the name of *decomposition of regular languages* and proved to be decidable [19,20].

Let us assume that (L_1, L_2) is a solution of $X \leftarrowtail_F Y = L_3$. Then $\sigma_{L_3}(L_2)$ is a solution of $L_1 \leftarrowtail_F Y = L_3$, and hence, $(L_1, \sigma_{L_3}(L_2))$ is also a solution of $X \leftarrowtail_F Y = L_3$. This means that if the equation has a solution (pair of languages), then it also has a solution whose second element is a sum of equivalence classes in $\Sigma^*/\equiv_{L_3}$. Therefore, solving $X \leftarrowtail_F ß[i] = L_3$ for all $1 \leq i \leq |ß|$ using Theorem 4 amounts to solving the two-variables equation. For a regular language R_3 and a regular p-schema F, the above method works effectively to solve $X \leftarrowtail_F Y = R_3$.

Theorem 9. *It is decidable whether the equation $X \leftarrowtail_F Y = R_3$ has a solution or not if both R_3 and F are regular.*

Undertaking the same “two-staged” strategy but using Theorem 5 instead, we can solve the equations of second and third forms.

Theorem 10. *For regular languages R_1, R_3, it is decidable whether the equation $R_1 \leftarrowtail_X Y = R_3$ has a solution or not.*

Theorem 11. *For a regular language R_1 and a* DPCM L_3*, it is decidable whether the equation $R_1 \rightarrowtail_X Y = L_3$ has a solution.*

Unlike p-schema-based insertion, this strategy does not work to solve the equation of the form $X \rightarrowtail_F Y = L_3$. This is because in this case it is not L_3 but L_1 that determines the syntactic solutions of $L_1 \rightarrowtail_F Y = L_3$.

The usage of the proposed algorithm is not exclusive to solving language *equations*. By replacing the equality test in Step 2 with the following inclusion test "for each $1 \leq i \leq |\text{ß}|$, test whether $L_1 \leftarrowtail_F \text{ß}[i]$ is a subset of R_3", the proposed algorithm can answer the problem of finding maximal solutions to the language inequality $L_1 \leftarrowtail_F X \subseteq R_3$, and with the two-staged strategy, this further enables us to solve $X \leftarrowtail_F Y \subseteq R_3$ and $L_1 \leftarrowtail_X Y \subseteq R_3$. Now it should be trivial how to approach $R_1 \rightarrowtail_F X \subseteq L_3$ and $R_1 \rightarrowtail_X Y \subseteq L_3$.

5.4 Undecidability

We conclude this section and this paper by complementing the decidability results obtained so far with some undecidability results for one-variable equations. Usually, the existence of solutions to a language equation of this type is decidable if all known languages are regular, and undecidable if at least one of the known languages is context-free. The results of this section bring down, for several cases, the limit for undecidability of existence of solutions of such language equations from the class of context-free languages to NCM(1). The equation $L_1 \leftarrowtail_F X = L_3$ is solvable in the case of L_1, F, L_3 being regular, i.e., NCM(0). Actually, we shall prove that once one of them becomes NCM(1), then this problem immediately turns into undecidable.

Proposition 8. *For languages L_1, L_3 and a p-schema F, if one of L_1, L_3, F is in* NCM(1) *and the others are regular, it is undecidable whether $L_1 \leftarrowtail_F X = L_3$ has a solution or not.*

Proof. We employ the reduction of universe problem (whether a given NCM(1) is Σ^*) into these problems. The universe problem is known to be undecidable for the class NCM(1) [17]. Because of space limitations, we can consider here only the case when F is an NCM(1) p-schema.

Let $\natural, \$$ be special symbols not included in Σ. Based on a given $L \in$ NCM(1), we define a p-schema $F = \lambda \times \$L$, which is in NCM(1), too. Then for regular languages $\$\Sigma^*$ and $\natural\$\Sigma^*$, we claim that $\$\Sigma^* \leftarrowtail_F X = \natural\$\Sigma^*$ has a solution $\Longleftrightarrow$ $L = \Sigma^*$. Indeed, the left-hand side of the above equation is $X\$L$ so that its only one possible solution is $X = \natural$. Thus, the existence of the solution leads us immediately to that L is universe. $\square$

For the equation $L_1 \rightarrowtail_F X = L_3$, the similar undecidability result holds.

Proposition 9. *For languages L_1, L_3 and a p-schema F, if one of L_1, L_3, F is in* NCM(1) *and the others are regular, it is undecidable whether $L_1 \rightarrowtail_F X = L_3$ has a solution or not.*

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science 266(5187), 1021–1024 (1994)
2. Ehrenfeucht, A., Harju, T., Petre, I., Rozenberg, G.: Patterns of micronuclear genes in ciliates. In: Jonoska, N., Seeman, N.C. (eds.) DNA 2001. LNCS, vol. 2340, pp. 279–289. Springer, Heidelberg (2002)
3. Landweber, L.F., Kari, L.: The evolution of cellular computing: Nature's solution to a computational problem. In: Kari, L., Rubin, H., Wood, D. (eds.) Proc. DNA-Based Computers IV, pp. 3–13 (1999)
4. Freund, R., Martín-Vide, C., Mitrana, V.: On some operations on strings suggested by gene assembly in ciliates. New Generation Computing 20, 279–293 (2002)
5. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. Information and Computation 131, 47–61 (1996)
6. Dieffenbach, C.W., Dveksler, G.S. (eds.): PCR Primer: A Laboratory Manual. Cold Spring Harbor Laboratory Press (2003)
7. Anselmo, M., Restivo, A.: On languages factorizing the free monoid. International Journal of Algebra and Computations 6, 413–427 (1996)
8. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall, London (1971)
9. Cui, B., Kari, L., Seki, S.: Block insertion and deletion on trajectories (2009) (in preparation)
10. Daley, M., Ibarra, O., Kari, L.: Closure and decidability properties of some language classes with respect to ciliate bio-operations. Theoretical Computer Science 306, 19–38 (2003)
11. Domaratzki, M.: Semantic shuffle on and deletion along trajectories. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) DLT 2004. LNCS, vol. 3340, pp. 163–174. Springer, Heidelberg (2004)
12. Domaratzki, M., Salomaa, K.: Decidability of trajectory-based equations. Theoretical Computer Science 345, 304–330 (2005)
13. Kari, L.: On Insertion and Deletion in Formal Languages. PhD thesis, University of Turku, Department of Mathematics, SF-20500 Turku, Finland (1991)
14. Kari, L.: On language equations with invertible operations. Theoretical Computer Science 132, 129–150 (1994)
15. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development 3, 114–125 (1959)
16. Domaratzki, M., Rozenberg, G., Salomaa, K.: Interpreted trajectories. Fundamenta Informaticae 73, 81–97 (2006)
17. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. Journal of the ACM 25, 116–133 (1978)
18. Chiniforooshan, E., Daley, M., Ibarra, O.H., Kari, L., Seki, S.: Reversal-bounded counter machines and multihead automata: Revisited (in preparation 2010)
19. Kari, L., Thierrin, G.: Maximal and minimal solutions to language equations. Journal of Computer and System Sciences 53, 487–496 (1996)
20. Salomaa, A., Yu, S.: On the decomposition of finite languages. In: Rozenberg, G., Thomas, W. (eds.) Developments in Language Theory, pp. 22–31 (1999)

On Schützenberger Products of Semirings

Ondřej Klíma and Libor Polák[⋆]

Department of Mathematics and Statistics, Masaryk University
Kotlářská 2, 611 37 Brno, Czech Republic

Abstract. The Schützenberger product of (ordered) monoids is an essential tool when studying the polynomial operators on Boolean and positive varieties of languages and concatenation hierarchies. Here we consider rather disjunctive varieties of languages and therefore the recognition of languages is by finite idempotent semirings. We define a product of finite idempotent semirings and we show similar results to those concerning Schützenberger products of monoids and ordered monoids.

Keywords: Polynomial operators on classes of languages, idempotent semirings, Schützenberger product.

1 Introduction

In the algebraic theory of regular languages a significant role is played by languages of the form

$$L_0 a_1 L_1 a_2 \dots a_n L_n \ , \qquad (*)$$

where a_i's are letters and L_i's are from a fixed class of languages. Such a product of languages is essential ingredient when considering concatenation hierarchies of regular languages. In Pin's survey paper [5] one can find a detailed overview and complete references.

In this area a method of studying properties of a given class of languages via properties of syntactic monoids of these languages is used. For example, here we can, for a given sequence of regular languages $K_0, \dots, K_m$, consider the sequence of (syntactic) monoids recognizing corresponding languages and construct the so-called Schützenberger product of monoids which recognizes the languages $K_0 a_1 K_1 a_2 \dots a_m K_m$, where a_i's are letters. Moreover, the constructed monoid is not unnecessarily huge, because every language which is recognized by this monoid can be written as a Boolean combination of languages of the form $(*)$, where the languages L_i's are recognized by the same monoids as the given languages K_j's – see Theorems 1.4, 1.5 of Pin's book [3] where the results are attributed to Schützenberger, Straubing, Reutenauer and Pin. An alternative proof can be found in Simon [10]. The main application of this result for second level of the Straubing-Thérien concatenation hierarchy can be found in [7].

[⋆] Both authors were supported by the Ministry of Education of the Czech Republic under the project MSM 0021622409 and by the Grant no. 201/09/1313 of the Grant Agency of the Czech Republic.

A significant progress was made by Pin in [6] where he introduced the Schützenberger products of ordered monoids. Note that the latter products of ordered monoids is a generalization of the previous Schützenberger products of (unordered) monoids.

Later on, the second author initiated a study of the so-called disjunctive varieties of regular languages in [8]. These classes are not necessarily closed under taking finite intersections. In that theory languages are recognized by idempotent semirings instead of (ordered) monoids. The authors studied certain subhierarchies of concatenation hierarchies in [1]; we refer to that paper for a short overview on differences among Boolean, positive and disjunctive varieties of languages. In connection with the languages of the form $(*)$, it is natural to consider classes of languages which are finite unions of such languages. For such considerations an analogue of Schützenberger product of (ordered) monoids for idempotent semirings would be very useful. A suitable definition and basic properties are the subject of this paper.

Our definition of Schützenberger product of semirings was inspired by such products of (ordered) monoids. Nevertheless our construction is not a simple generalization as it would seem to be. First of all, our definition is based on certain closure operators which makes this concept a bit tricky. And surprisingly the definition of our product of semirings is not a generalization of Schützenberger products of (ordered) monoids in the sense that Schützenberger products of (ordered) monoids can be obtained from our product for specially chosen semirings. When we compare Schützenberger products of monoids and such products of ordered monoids then the situation is quite simple: each monoid can be equipped by ordering, namely $=$, in such a way that the resulting ordered monoid recognized the same languages as the original monoid. This is not true for idempotent semirings; it is not possible to construct an idempotent semiring recognizing the same languages as a given finite (ordered) monoid. This fact is connected with the observation that the (ordered) syntactic monoid is a substructure of the syntactic semiring of a given language but we could not exactly identify it.

In future investigations one should relate our product with the triangular product of Rhodes and Steinberg [9] used in their q-theory.

Our paper is structured as follows. After this introductory section we place preliminaries in Section 2. We define our product in the next section and we also formulate the main results there. Section 4 is devoted to proofs – it is possible to skip it during the first look into our contribution. In the last section we iniciate applications of our main results to disjunctive varieties of languages.

2 Preliminaries

An *idempotent semiring* is a structure $(S, \cdot, +, 1)$ where

- $(S, \cdot)$ is a monoid with the neutral element 1,
- $(S, +)$ is a semilattice,
- $(\forall\, a, b, c \in S \ ) \ (\ a(b + c) = ab + ac, \ \ (a + b)c = ac + bc \ ).$

Let $A^{\sqcup}$ denote the set of all finite non-empty subsets of A^*. For $U, V \in A^{\sqcup}$, we define $U \cdot V = \{ uv \mid u \in U,\, v \in V \}$. Then $(A^{\sqcup}, \cdot, \cup, \{\lambda\})$ is a free idempotent semiring over A.

On each idempotent semiring $(S, \cdot, +, 1)$ we define an order $\leq$ by

$$a \leq b \ \text{ if and only if } \ a + b = b\,.$$

It is a compatible order, i.e. it satisfies

$$(\ \forall\, a, b, c \in S\ )\ a \leq b \ \text{ implies } \ ac \leq bc \ \text{ and } \ ca \leq cb\,.$$

An *ideal* m of $(S, \cdot, +, 1)$ is a subset of S satisfying

- if $a \in m,\ b \in S,\ b \leq a,$ then $b \in m$,
- if $a, b \in m,$ then $a + b \in S$.

Notice that an ideal of a finite idempotent semiring is either empty or it is of the form $(a] = \{ b \in S \mid b \leq a \},\ a \in S$. We will denote by $\mathsf{Id}(S, \cdot, +, 1)$ (or briefly $\mathsf{Id}(S)$) the set of all ideals of $(S, \cdot, +, 1)$.

A *semiring homomorphism* of $(S, \cdot, +, 1)$ into $(T, \cdot, +, 1)$ is a mapping $\varphi : S \to T$ satisfying $\varphi(1) = 1$ and, for all $a, b \in S$, we have $\varphi(ab) = \varphi(a)\varphi(b)$ and $\varphi(a + b) = \varphi(a) + \varphi(b)$.

A language $L \subseteq A^*$ is *recognized* by a finite idempotent semiring $(S, \cdot, +, 1)$ if there exists an ideal $m \subseteq S$ and a monoid homomorphism $\varphi : A^* \to (S, \cdot, 1)$ such that

$$L = \{ v \in A^* \mid \varphi(v) \notin m \}\ .$$

Note that such φ can be uniquely extended to a semiring homomorphism $\varphi : A^{\sqcup} \to (S, \cdot, +, 1)$. We are going to study rather the disjunctive varieties of languages; the analogous version of recognizability, i.e. $L = \{ v \in A^* \mid \varphi(v) \in m \}$, would be suitable for conjunctive varieties - see [8].

3 The Schützenberger Product of Finite Idempotent Semirings: Definition and the Main Results

Let $(S_0, \cdot, +, 1), \ldots, (S_n, \cdot, +, 1)$ be finite idempotent semirings (often we write only $S_0, \ldots, S_n$). For $0 \leq i \leq j \leq n$, we write

$$S_{i,j} = S_i \times \cdots \times S_j \ \text{ and } \ \mathsf{Id}_{i,j} = \mathsf{Id}(S_1) \times \cdots \times \mathsf{Id}(S_j)$$

and we define on subsets of each $S_{i,j}$ an operator $[\ ]_{i,j}$. In all situations the indices (i, j) would be known from the context; so we write only $[\ ]$. For $X \subseteq S_{i,j}$, we put

$$\begin{gathered}[\,X\,]\ = \{\ (s_i, \ldots, s_j) \in S_{i,j} \mid \\ (\ \forall\ (m_i, \ldots, m_j) \in \mathsf{Id}_{i,j} \text{ such that } s_i \notin m_i, \ldots, s_j \notin m_j\ ) \\ (\ \exists\ (t_i, \ldots, t_j) \in X \text{ such that } t_i \notin m_i, \ldots, t_j \notin m_j\ )\ \}\,.\end{gathered}$$

We will use a "compact" notation: $(s_i, \dots, s_j) \propto (m_i, \dots, m_j)$ would mean that $s_i \notin m_i, \dots, s_j \notin m_j$.

These closure operators deserve a careful study – see Lemma 2. Nevertheless we introduce the crucial notion of our contribution already now. The *Schützenberger product* $\diamond_n(S_0, \dots, S_n)$ (briefly $\diamond(S_0, \dots, S_n)$) of finite idempotent semirings $S_0, \dots, S_n$ is defined as follows:

(i) Elements are all $n+1$ times $n+1$ matrices P where
   - $P_{i,j} = \emptyset$ for $i > j$, and
   - for $i \le j$, $P_{i,j}$ is a closed subset of $S_{i,j}$, i.e. $[\,P_{i,j}\,] = P_{i,j}$.

   Moreover, for $i = 0, \dots, n$, we have $P_{i,i} \neq \emptyset$ (which means $P_{i,i} = (s_i]$ for some $s_i \in S_i$ by Lemma 2).

(ii) We define addition and multiplication in $\diamond(S_0, \dots, S_n)$ by: for $P, Q \in \diamond(S_0, \dots, S_n)$ and $0 \le i \le j \le n$,

$$(P+Q)_{i,j} = [\,P_{i,j} \cup Q_{i,j}\,]\,,$$

$$(P \cdot Q)_{i,j} = [\bigcup_{i \le k \le j} P_{i,k} \circ Q_{k,j}\,] \quad \text{where} \quad P_{i,k} \circ Q_{k,j} =$$

$$= \{\,(p_i, \dots, p_{k-1}, p_k \cdot q_k, q_{k+1}, \dots, q_j) \mid (p_i, \dots, p_k) \in P_{i,k},\ (q_k, \dots, q_j) \in Q_{k,j}\,\}\,.$$

(iii) The neutral element for the multiplication is the matrix E with $E_{i,j} = \emptyset$ for $i \neq j$ and $E_{i,i} = (1]$, for $i, j \in \{0, \dots, n\}$.

We formulate the main results now. Their proofs are postponed into the next section.

Proposition 1. *The structure $(\diamond(S_0, \dots, S_n), \cdot, +, 1)$ is a (finite) idempotent semiring.*

Theorem 1. *Let the language $L_i \subseteq A^*$ be recognized by a finite idempotent semiring $(S_i, \cdot, +, 1)$ for $i = 0, \dots, n$ and let $a_1, \dots, a_n \in A$. Then the language $L = L_0 a_1 L_1 a_2 \dots a_n L_n$ is recognized by the semiring $(\diamond(S_0, \dots, S_n), \cdot, +, 1)$.*

Theorem 2. *Let $(S_i, \cdot, +, 1)$, for $i = 0, \dots, n$, be a finite idempotent semiring. Let the language $L \subseteq A^*$ be recognized by the semiring $(\diamond(S_0, \dots, S_n), \cdot, +, 1)$. Then L is a finite union of the languages $L_{i_0} a_1 L_{i_1} a_2 \dots a_k L_{i_k}$ where $0 \le i_0 < \dots < i_k \le n$, $a_1, \dots, a_k \in A$ and L_{i_p} is recognized by $(S_{i_p}, \cdot, +, 1)$ for $p = 0, \dots, k$.*

4 Auxiliary Results and Proofs of Theorems

Lemma 1. *Let $(S, \cdot, +, 1)$ be a finite idempotent semiring and let $s \in S$, $m \in \mathsf{Id}(S)$. Then there exist $m', m'' \in \mathsf{Id}(S)$ such that*

$$\{\,x \in S \mid xs \notin m\,\} = \{\,x \in S \mid x \notin m'\,\},\ \{x \in S \mid sx \notin m\} = \{x \in S \mid x \notin m''\}\,.$$

We write $m' = ms^{-1}$, $m'' = s^{-1}m$.

Proof. For $m = \emptyset$ we put also $m' = \emptyset$. So let $m = (t]$, $t \in S$. If the set $\{\, x \in S \mid xs \le t \,\}$ is empty, then $\{\, x \in S \mid xs \notin m \,\} = S$ and we put $m' = \emptyset$. Otherwise let $u = \bigvee \{\, x \in S \mid xs \le t \,\}$ (the supremum in $(S, \le)$). Then $\{\, x \in S \mid xs \le t \,\} = \{\, x \in S \mid x \le u \,\}$ and we put $m' = (u]$. The proof of the second equality is analogous. □

Lemma 2. *Let $0 \le i \le k \le j \le n$. Then*

(i) $[\;]$ is a closure operator on $S_{i,j}$; i.e. it is extensive ($X \subseteq [\, X \,]$), isotone ($X \subseteq Y$ implies $[\, X \,] \subseteq [\, Y \,]$) and idempotent ($[\, [\, X \,] \,] = [\, X \,]$),

(ii) a closed subset X of $S_{i,j}$ is hereditary; that is,

$$(s_i, \ldots, s_j) \in X,\ (t_i, \ldots, t_j) \in S_{i,j},\ t_i \le s_i, \ldots, t_j \le s_j \textit{ implies } (t_i, \ldots, t_j) \in X\,,$$

(iii) for a closed subset X of $S_{i,j}$ we have

$$\textit{if } (s_i, \ldots, s_j), (s_i, \ldots, s_{k-1}, s'_k, s_{k+1}, \ldots, s_j) \in X$$

$$\textit{then } (s_i, \ldots, s_{k-1}, s_k + s'_k, s_{k+1}, \ldots, s_j) \in X\,,$$

(iv) the closed subsets of $S_{i,i} = S_i$ are exactly ideals, i.e. they are equal to $\emptyset$ or to $[\{s_i\}] = (s_i]$ for some $s_i \in S_i$,

(v) for $X, Y \subseteq S_{i,j}$, we have

$$[\, [\, X \,] \cup [\, Y \,] \,] = [\, X \cup [\, Y \,] \,] = [\, [\, X \,] \cup Y \,] = [\, X \cup Y \,]\,,$$

(vi) for $X \subseteq S_{i,k}$ and $Y \subseteq S_{k,j}$, we have

$$[\, [\, X \,] \circ [\, Y \,] \,] = [\, X \circ [\, Y \,] \,] = [\, [\, X \,] \circ Y \,] = [\, X \circ Y \,],$$

(vii) for $X_1, \ldots, X_p \subseteq S_{i,k}$ and $Y_1, \ldots, Y_p \subseteq S_{k,j}$, we have

$$[\, [\, X_1 \,] \circ [\, Y_1 \,] \cup \cdots \cup [\, X_p \,] \circ [\, Y_p \,] \,] = [\, X_1 \circ Y_1 \cup \cdots \cup X_p \circ Y_p \,]\,.$$

Proof. The items (i) – (v) are clear.

(vi) We need to prove that $[\, [X] \circ Y \,] \subseteq [\, X \circ Y \,]$. So let $s \in [\, [\, X \,] \circ Y \,]$ and let $s \propto m = (m_i, \ldots, m_j) \in \mathsf{Id}_{i,j}$. We are looking for $t \in X \circ Y$ with $t \propto m$.

There exists $u = (u_i, \ldots, u_{k-1}, u_k v_k, v_{k+1}, \ldots, v_j) \in [X] \circ Y$ such that

$$(u_i, \ldots, u_{k-1}, u_k) \in [\, X \,],\ (v_k, v_{k+1}, \ldots, v_j) \in Y\,,$$

$$(u_i, \ldots, u_{k-1}) \propto (m_i, \ldots, m_{k-1}),\ u_k v_k \notin m_k,\ (v_{k+1}, \ldots, v_j) \propto (m_{k+1}, \ldots, m_j)\,.$$

Using Lemma 1, we can write $u_k \notin m_k (v_k)^{-1}$ instead of $u_k v_k \notin m_k$. Thus there exist

$$(w_i, \ldots, w_k) \in X \text{ such that } (w_i, \ldots, w_{k-1}) \propto (m_i, \ldots, m_{k-1}),\ w_k \notin m_k (v_k)^{-1}\,.$$

The last relation is equivalent to $w_k v_k \notin m_k$ and thus we have found

$$t = (w_i, \ldots, w_{k-1}, w_k v_k, v_{k+1}, \ldots, v_j) \propto m,\ t \in X \circ Y\,.$$

Item (vii) follows from the two preceding ones. □

Proof of Proposition 1. The commutativity and the idempotency of the operation $+$ is clear. Associativity of $+$ and $\cdot$ follow from Lemma 2(v) - (vii). The neutral element for the multiplication is really the matrix E. The distributivity laws follow from Lemma 2(v) and (vi). $\square$

Proof of Theorem 1. Let the language $L_i \subseteq A^*$ be recognized by the semiring $(S_i, \cdot, +, 1)$ using $m_i \in \mathsf{Id}(S_i)$ and $\varphi_i : A^* \to S_i$, for $i = 0, \dots, n$.

Define a mapping $\varphi : A^* \to \diamondsuit(S_0, \dots, S_n)$ where, for $0 \le i \le j \le n,\ u \in A^*$,

$$(\varphi(u))_{i,j} \subseteq S_i \times \cdots \times S_j \ \text{ is given by}$$

$$[\ \{\ (\varphi_i(u_i), \dots, \varphi_j(u_j)) \mid u = u_i a_{i+1} u_{i+1} a_{i+2} \dots a_j u_j,\ u_i, \dots, u_j \in A^*\ \}\ ]\ .$$

We will show that φ is a monoid homomorphism. One can check that $\varphi(1) = E$. Let $0 \le i \le j \le n$ and let $u, v \in A^*$. We have

$$(\varphi(uv))_{i,j} =$$

$$[\ \{\ (\varphi_i(w_i), \dots, \varphi_j(w_j)) \mid uv = w_i a_{i+1} w_{i+1} a_{i+2} \dots a_j w_j,\ w_i, \dots, w_j \in A^*\ \}\ ]$$

$$= [\ \bigcup_{i \le k \le j} \{\ (\varphi_i(u_i), \dots, \varphi_{k-1}(u_{k-1}), \varphi_k(u_k)\varphi_k(v_k), \varphi_{k+1}(v_{k+1}), \dots, \varphi_j(v_j)) \mid$$

$$\mid u = u_i a_{i+1} \dots a_k u_k,\ v = v_k a_{k+1} \dots a_j v_j,\ u_i, \dots, u_k, v_k, \dots, v_j \in A^*\ \}\ ]\ .$$

Using Lemma 2 (vii), we have

$$(\varphi(u) \cdot \varphi(v))_{i,j} = [\ \bigcup_{i \le k \le j} (\varphi(u))_{i,k} \circ (\varphi(v))_{k,j}\ ] =$$

$$= [\ \bigcup_{i \le k \le j} [\ \{\ (\varphi_i(u_i), \dots, \varphi_k(u_k)) \mid u = u_i a_{i+1} \dots a_k u_k,\ u_i, \dots, u_k \in A^*\ \}\ ]$$

$$\circ [\ \{\ (\varphi_k(v_k), \dots, \varphi_j(v_j)) \mid v = v_k a_{k+1} \dots a_j v_j,\ v_k, \dots, v_j \in A^*\ \}\ ]\ ] =$$

$$= [\ \bigcup_{i \le k \le j} \{\ (\varphi_i(u_i), \dots, \varphi_k(u_k)) \mid u = u_i a_{i+1} \dots a_k u_k,\ u_i, \dots, u_k \in A^*\ \}$$

$$\circ \{\ (\varphi_k(v_k), \dots, \varphi_j(v_j)) \mid v = v_k a_{k+1} \dots a_j v_j,\ v_k, \dots, v_j \in A^*\ \}\ ]$$

and the claim follows.

Notice that, for $a \in A$, we have $(\varphi(a))_{i,i} = (\varphi_i(a)],\ (\varphi(a))_{i,i+1} = ((1,1)]$ if $a = a_i$ and $(\varphi(a))_{i,j} = \emptyset$, otherwise.

Let $P \in \diamondsuit(S_0, \dots, S_n)$ be defined by

$$P_{i,j} = \emptyset \text{ for } i > j\ ,$$

$$P_{0,n} = m_0 \times S_1 \times \cdots \times S_n \cup S_0 \times m_1 \times \cdots \times S_n \cup \cdots \cup S_0 \times \cdots \times S_{n-1} \times m_n\ ,$$

$$P_{i,j} = S_i \times \cdots \times S_j,\ \text{otherwise.}$$

Notice that $s \notin P_{0,n}$ if and only if $s \propto (m_0, \dots, m_n)$. Thus $P_{0,n}$ is closed and $P \in \diamondsuit(S_0, \dots, S_n)$.

Now $u \in A^*$ is in the language L recognized by $\Diamond(S_0, \ldots, S_n)$ using φ and the ideal $(P]$ if and only if

$$\varphi(u) \not\leq P, \text{ which is } (\varphi(u))_{0,n} \not\subseteq P_{0,n} \ .$$

The last condition is equivalent to the existence of a decomposition of the word u as $u = u_0 a_1 u_1 a_2 \ldots a_n u_n,\ u_0, \ldots, u_n \in A^*$ satisfying $(\varphi_0(u_0), \ldots, \varphi_n(u_n)) \notin P_{0,n}$. This means that $\varphi_i(u_i) \notin m_i$, for $i = 0, \ldots, n$, which is exactly the fact that $u_i \in L_i$ and $u \in L_0 a_1 L_1 a_2 \ldots a_n L_n$. □

Proof of Theorem 2. Let $\varphi : A^* \to \Diamond(S_0, \ldots, S_n)$. The empty ideal recognizes the language A^* which is also of the form from the theorem. Hence we may assume that $L \neq A^*$. So let $P \in \Diamond(S_0, \ldots, S_n)$ and let

$$L = \{ v \in A^* \mid \varphi(\{v\}) \not\leq P \} \ .$$

Since $\varphi(A^{\sqcup})$ is a subsemilattice of $\Diamond(S_0, \ldots, S_n)$ and $L \neq A^*$, we can take Q to be the greatest element of $\varphi(A^{\sqcup})$ which is less or equal to P. Then

$$L = \{ v \in A^* \mid \varphi(\{v\}) \not\leq Q \} \ .$$

We write often $\varphi_{i,j}(V)$ instead of $(\varphi(V))_{i,j}$, simply $\varphi_{i,j}(v)$ for $\varphi_{i,j}(\{v\})$ and φ_i for $\varphi_{i,i}$.

For each $0 \leq i_0 < \cdots < i_k \leq n$, $a_1, \ldots, a_k \in A$ and $U \in A^{\sqcup}$, we define the set

$$\mathsf{Seq}^{i_0,\ldots,i_k}_{a_1,\ldots,a_k}(U) = \{ (m_0, \ldots, m_k) \in \mathsf{Id}(S_{i_0}) \times \cdots \times \mathsf{Id}(S_{i_k}) \mid \text{ there exists } u \in U,$$

$$u = u_0 a_1 u_1 a_2 \ldots a_k u_k,\ u_0, \ldots, u_k \in A^* \text{ and } \varphi_{i_p}(u_p) \not\subseteq m_p \text{ for } p = 0, \ldots, k \} .$$

Further, we define a relation $\equiv$ on $A^{\sqcup}$ by:

$$V \equiv W \text{ if and only if } (\ \forall\ 0 \leq i_0 < \cdots < i_k \leq n,\ a_1, \ldots, a_k \in A\ )$$

$$\mathsf{Seq}^{i_0,\ldots,i_k}_{a_1,\ldots,a_k}(V) = \mathsf{Seq}^{i_0,\ldots,i_k}_{a_1,\ldots,a_k}(W) \ .$$

In our proof we will use a couple of auxiliary results.

Lemma 3. *The relation $\equiv$ is a congruence relation on the semiring $A^{\sqcup}$ of finite index.*

Proof. Let $U, V, W \in A^{\sqcup}$ be such that $U \equiv V$. Clearly, $(U \cup W) \equiv (V \cup W)$.

We have to show that also $U \cdot W \equiv V \cdot W$ and $W \cdot U \equiv W \cdot V$. It is enough to show these equalities for $W = \{a\},\ a \in A$.

The first formula would follow from the fact that, for $0 \leq i_1 < \cdots < i_k \leq n$ and $a_1, \ldots, a_k \in A,\ a_k = a$, we have

$$\mathsf{Seq}^{i_0,\ldots,i_k}_{a_1,\ldots,a_k}(U\{a\}) =$$

$$= \{ (m_0, \ldots, m_k) \mid (m_0, \ldots, m_k s_k^{-1}) \in \mathsf{Seq}^{i_0,\ldots,i_k}_{a_1,\ldots,a_k}(U) \} \cup$$

$$\cup \{ (m_0, \ldots, m_k) \mid (m_0, \ldots, m_{k-1}) \in \mathsf{Seq}^{i_0,\ldots,i_{k-1}}_{a_1,\ldots,a_{k-1}}(U),\ 1 \notin m_k \} ,$$

where $\varphi_{i_k}(a) = s_k$. In case of $a_k \neq a$ the right-side of the last formula will consist only from the first summand.

To get the second formula one will consider $\{a\}U$ instead of $U\{a\}$.

Clearly, the relation $\equiv$ is of finite index. □

Lemma 4. *Let $\varphi : A^{\sqcup} \to (\Diamond(S_0, \ldots, S_n), \cdot, +, 1)$ be a semiring homomorphism, let $V \in A^{\sqcup}$ and let $0 \leq i \leq j \leq n$. Then*

$$\varphi_{i,j}(V) = [\, \bigcup \varphi_{i_0}(v_0) \circ \varphi_{i_0,i_1}(a_1) \circ \varphi_{i_1}(v_1) \circ \varphi_{i_1,i_2}(a_2) \cdots \varphi_{i_{k-1},i_k}(a_k) \circ \varphi_{i_k}(v_k)\,]\,,$$

where the union is indexed by all

$$i = i_0 < \cdots < i_k = j, v \in V,\, v = v_0 a_1 v_1 \ldots a_k v_k, a_1, \ldots, a_k \in A, v_0, \ldots, v_k \in A^*\,.$$

Proof. It suffices to prove the formula for $V = \{v\}$. Let $v = b_1 \ldots b_r,\ b_1, \ldots, b_r \in A$. Due to $\varphi_{p,q} = \emptyset$ for $q < p$ and Lemma 2 we can write

$$\varphi_{i,j}(v) = [\bigcup_{i=i_0 \leq i_1 \leq \cdots \leq i_r = j} \varphi_{i_0,i_1}(b_1) \circ \varphi_{i_1,i_2}(b_2) \circ \cdots \circ \varphi_{i_{r-1},i_r}(b_r)\ ]\ .$$

Since each $\varphi_i = \varphi_{i,i}$ is a monoid homomorphism we can group together the product $\varphi_i(b_p) \ldots \varphi_i(b_{p+q})$ to get $\varphi_i(b_p \ldots b_{p+q})$. □

For each sets B and C and a mapping $\varphi : B \to C$, we define the relation *kernel* of φ on the set B by

$$(b, b') \in \mathsf{ker}(\varphi) \ \text{ if and only if } \ \varphi(b) = \varphi(b')\,.$$

Lemma 5. *For the relation $\equiv$ on $A^{\sqcup}$ and the kernel of φ, it holds*

$$\equiv\ \subseteq \mathsf{ker}(\varphi).$$

Proof. Let $V, W \in A^{\sqcup}$ satisfy $V \equiv W$. It is enough to show that $\varphi(V) \leq \varphi(W)$ (the opposite relation can be proved analogously). Let $0 \leq i \leq j \leq n$ and consider $\varphi_{i,j}(V)$. Using Lemma 4, we choose

$$i \leq i_0 < \cdots < i_k \leq j,\ v \in V,\ v = v_0 a_1 v_1 a_2 \ldots a_k v_k,$$

$$a_1, \ldots, a_k \in A,\ v_0, \ldots, v_k \in A^*\,.$$

Let

$$\varphi_{i_0}(v_0) = (s_0],\ (t_{1,i_0}, \ldots, t_{1,i_1}) \in \varphi_{i_0,i_1}(a_1),$$

$$\varphi_{i_1}(v_1) = (s_1],\ (t_{2,i_1}, \ldots, t_{2,i_2}) \in \varphi_{i_1,i_2}(a_2), \ldots$$

We will show that

$$x = (s_0 t_{1,i_0}, t_{1,i_0+1}, \ldots, t_{1,i_1-1}, t_{1,i_1} s_1 t_{2,i_1}, t_{2,i_1+1}, \ldots, t_{2,i_2-1}, \ldots) \in \varphi_{i,j}(W)\,.$$

Choose ideals $(m_{i_0}, m_{i_0+1}, \ldots, m_{i_k}) \in \mathsf{Id}(S_{i_0}) \times \cdots \times \mathsf{Id}(S_{i_k})$ such that

$$x \propto (m_{i_0}, m_{i_0+1}, \ldots, m_{i_k})\,.$$

Then using Lemma 1 we have

$$s_0 \notin m_{i_0} t_{1,i_0}^{-1},\ s_1 \notin t_{1,i_1}^{-1}(m_{i_1} t_{2,i_1}^{-1}), \dots .$$

The last sequence of ideals is in $\mathsf{Seq}_{a_1,\dots,a_k}^{i_0,\dots,i_k}(v)$ and thus also in $\mathsf{Seq}_{a_1,\dots,a_k}^{i_0,\dots,i_k}(W)$. Therefore there exist

$$w \in W,\ w = w_0 a_1 w_1 a_2 \dots a_k w_k, w_0, \dots, w_k \in A^*$$

such that

$$(u_0] = \varphi_{i_0}(w_0) \not\subseteq m_{i_0} t_{1,i_0}^{-1},\ (u_1] = \varphi_{i_1}(w_1) \not\subseteq t_{1,i_1}^{-1}(m_{i_1} t_{2,i_1}^{-1}), \dots .$$

Now again by Lemma 1 we have

$$u_0 t_{1,i_0} \notin m_{i_0},\ t_{1,i_0+1} \notin m_{i_0+1}, \dots, t_{1,i_1-1} \notin m_{i_1-1},$$

$$t_{1,i_1} u_1 t_{2,i_1} \notin m_{i_1},\ t_{2,i_1+1} \notin m_{i_1+1}, \dots, t_{2,i_2-1} \notin m_{i_2-1}, \dots .$$

and therefore

$$x \in [\{(u_0 t_{1,i_0}, t_{1,i_0+1}, \dots, t_{1,i_1-1}, t_{1,i_1} u_1 t_{2,i_1}, t_{2,i_1+1}, \dots, t_{2,i_2-1}, \dots)\}] \subseteq$$

$$\subseteq [\{u_0 \circ (t_{1,i_0}, t_{1,i_0+1}, \dots, t_{1,i_1-1}, t_{1,i_1}) \circ u_1 \circ (t_{2,i_1}, t_{2,i_1+1}, \dots, t_{2,i_2-1}) \circ \dots\}] \subseteq$$

$$\subseteq [\, \varphi_{i_0}(w_0) \circ \varphi_{i_0,i_1}(a_1) \circ \varphi_{i_1}(w_1) \circ \varphi_{i_1,i_2}(a_2) \circ \cdots \circ \varphi_{i_{k-1},i_k}(a_k) \circ \varphi_{i_k}(w_k)\,] \subseteq \varphi_{i,j}(W).$$

□

A continuation of the proof of Theorem 2. Due to Lemma 3 we can form the (finite !) quotient structure $A^{\sqcup}/\equiv$ and consider the natural semiring homomorphism $\kappa : V \mapsto V\equiv$, $V \in A^{\sqcup}$. Due to Lemma 5, the homomorphisms φ factorizes through κ, i.e., there is a homomorphism $\psi : A^{\sqcup}/\equiv \to \Diamond(S_0, \dots, S_n)$ such that $\varphi = \psi\kappa$.

Let $U/\equiv$ be the supremum of the finite set $\{\, V/\equiv \mid \psi(V/\equiv) \le Q \,\}$. Then $U/\equiv$ is the greatest element of $A^{\sqcup}/\equiv$ with $\psi(U/\equiv) = Q$ and we can write

$$L = \{\, v \in A^* \mid U \cup \{v\} \ \not\equiv\ U \,\} .$$

Now $v \in L$ if and only if there exist $0 \le i_0 < \cdots < i_k \le n,\ a_1, \dots, a_k \in A$ and

$$(m_0, \dots, m_k) \in \mathsf{Seq}_{a_1,\dots,a_k}^{i_0,\dots,i_k}(v) \setminus \mathsf{Seq}_{a_1,\dots,a_k}^{i_0,\dots,i_k}(U) .$$

Let

$$K = \bigcup\ L_{i_0} a_1 L_{i_1} a_2 \dots a_k L_{i_k}$$

where the union is indexed by all

$$0 \le i_0 < \cdots < i_k \le n,\ a_1, \dots, a_k \in A,\ (m_0, \dots, m_k) \notin \mathsf{Seq}_{a_1,\dots,a_k}^{i_0,\dots,i_k}(U)$$

and where

$$L_{i_p} = \{\, w \in A^* \mid \varphi_{i_p}(w) \not\subseteq m_p \,\} \text{ for } \ p = 1, \dots, k\ .$$

Clearly, $L = K$. □

5 Applications

Well-established classes of regular languages are varieties and positive varieties. By the Eilenberg correspondences there are natural bijections between such classes and pseudovarieties of finite monoids (ordered monoids, respectively) – see, for instance, Pin [4],[5]. Here we consider the disjunctive varieties of languages introduced in [8] and studied among other in [1].

A *disjunctive variety* of languages $\mathcal{D}$ associates to every finite alphabet A a class $\mathcal{D}(A)$ of regular languages over A in such a way that

– $A^* \in \mathcal{D}(A)$,
– $\mathcal{D}(A)$ is closed under finite unions (in particular $\emptyset \in \mathcal{D}(A)$),
– $\mathcal{D}(A)$ is closed under derivatives, i.e. $L \in \mathcal{D}(A),\ u, v \in A^*$ implies $u^{-1}Lv^{-1} = \{ w \in A^* \mid uwv \in L \} \in \mathcal{D}(A)$,
– $\mathcal{D}$ is closed under preimages in semiring homomorphisms, i.e. $f : B^{\sqcup} \to A^{\sqcup},\ L \in \mathcal{D}(A)$ implies $f^{-1}(L) = \{ u \in B^* \mid f(u) \cap L \neq \emptyset \} \in \mathcal{D}(B)$.

Again, an Eilenberg-type correspondence relates such varieties to pseudovarieties of finite idempotent semirings. For a disjunctive variety of languages $\mathcal{D}$, the corresponding pseudovariety $\mathbf{D}$ is generated by syntactic semirings of languages from $\mathcal{D}$ – see [1].

In the third part of this section we formulate general results concerning the polynomial operator and disjunctive varieties. Before that we look at two special cases where we consider relatively simple disjunctive varieties.

We denote by T the trivial semiring, which has only one element. It is natural to denote this element by 1 because it is neutral (with respect to both operations). The next example of idempotent semiring is the semiring consisting of all subsets of T. Here we have two elements $\emptyset = 0$ and $\{1\} = 1$ and the operations on the set $U = \{0, 1\}$ are given by rules $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0,\ 1 \cdot 1 = 1$, $0 + 0 = 0,\ 0 + 1 = 1 + 0 = 1 + 1 = 1$. We apply Schützenberger products of semirings to these examples in this section.

5.1 Starting from the Trivial Semiring

The trivial semiring T contains exactly two ideals, namely $\emptyset$ and T. Therefore, for each alphabet A, it recognizes exactly two languages, A^* and $\emptyset$.

For a fixed natural number n we consider the semiring $\diamond_n(T, \ldots, T)$. By our main result this semiring recognizes (over an alphabet A) exactly the finite unions of languages of the form $A^*a_1A^*a_2 \ldots a_kA^*$, where $a_1, \ldots, a_k \in A$, $k \leq n$. These languages form a disjunctive variety of languages, denoted by $\mathcal{DV}_n$ in [1]. It follows that the semiring $\diamond_n(T, \ldots, T)$ generates the corresponding pseudovariety of finite semirings $\mathbf{DV}_n$, i.e. we have an alternative proof of Proposition 3 (iii) from [1].

Proposition 2 ([1]). *The pseudovariety* $\mathbf{DV}_n$ *is generated by a single idempotent semiring.*

If we look at the definition of $\Diamond_n(T, \ldots, T)$ more carefully, we see that S_{ij} is a one-element set and consequently there are exactly two closed subsets of S_{ij}, namely the empty set and the whole set S_{ij}. From that reason elements of $\Diamond_n(T, \ldots, T)$ can be viewed as matrices over the semiring U. Thus $\Diamond_n(T, \ldots, T)$ is isomorphic to the semiring of all upper triangular matrices $(n + 1)$ times $(n + 1)$ over the semiring U with 1's at the diagonal and with usual operations of matrices.

5.2 Starting from Two Element Idempotent Semiring *U*

The semiring U contains exactly three ideals $\emptyset$, $\{0\}$ and U. Thus U recognizes (over an alphabet A) exactly the empty language and languages of the form B^*, where $B \subseteq A$. For a fixed natural number n we consider the semiring $\Diamond_n(U, \ldots, U)$ which recognizes (over an alphabet A) exactly finite unions of languages of the form $B_0^* a_1 B_1^* a_2 \ldots a_k B_k^*$, where $a_1, \ldots, a_k \in A$, $B_0, \ldots, B_k \subseteq A$, $k \le n$. One can check that these languages form a disjunctive variety (to verify the last condition from the definition is not completely trivial). In accordance with [2] we denote the considered class of languages by $\mathsf{DPol}_n(\mathcal{S}^+)$..

From our main result we obtain the following statement for the corresponding pseudovariety of idempotent semirings $\mathbf{DPol}_n(\mathbf{S}^+)$.

Proposition 3. *For each n, the class $\mathsf{DPol}_n(\mathcal{S}^+)$ is a disjunctive variety of languages and the corresponding pseudovariety of idempotent semirings $\mathbf{DPol}_n(\mathbf{S}^+)$ is generated by $\Diamond_n(U, \ldots, U)$.* □

In this result, in a similar way to the case of Schützenberger products of monoids (see [7]), we can replace the semiring $\Diamond_n(U, \ldots, U)$ by a simpler structure, namely by the semiring D_n of all upper triangular $(n+1)$ times $(n+1)$ matrices over the semiring U with usual operations of multiplication and addition of matrices.

Proposition 4. *The pseudovariety $\mathbf{DPol}_n(\mathbf{S}^+)$ is generated by the idempotent semiring D_n.*

Proof. First we define a mapping $\alpha : \Diamond_n(U, \ldots, U) \to D_n$ by the rule

$$(\alpha(P))_{i,j} = 1 \text{ if and only if } (1, 1, \ldots, 1) \in P_{i,j} \,.$$

One can check that α is a surjective semiring homomorphism from $\Diamond_n(U, \ldots, U)$ onto D_n. Hence each language which is recognized by the idempotent semiring D_n is also recognized by the idempotent semiring $\Diamond_n(U, \ldots, U)$.

Thus, it is enough to prove that D_n recognizes each language L of the form $L = B_0^* a_1 B_1^* a_2 \ldots a_k B_k^*$, where $a_1, \ldots, a_k \in A$, $B_0, \ldots, B_k \subseteq A$, $k \le n$. Each B_i^* is recognized by $\varphi_i : A^* \to U$ using the ideal $\{0\}$. In the proof of Theorem 1 we constructed $\varphi : A^* \to \Diamond_n(U, \ldots, U)$ which recognizes L using a certain ideal. In our situation where $m_0 = \cdots = m_n = \{0\}$ we have $P_{0,n} = U^{n+1} \setminus \{(1, 1, \ldots 1)\}$. Hence we can consider a homomorphism $\alpha \circ \varphi : A^* \to D_n$ and we can consider the ideal consisting of all matrices $M \in D_n$ such that $(M)_{0,n} = 0$. We see that $\alpha \circ \varphi$ recognizes the language L. □

5.3 Schützenberger Product of Varieties of Languages

Let $\mathcal{V}$ be a class of languages and let n be a natural number. We define the class $\mathsf{DPol}_n(\mathcal{V})$ of *polynomials* of length at most n of languages from the class $\mathcal{V}$. Namely, for a finite alphabet A, the class $(\mathsf{DPol}_n(\mathcal{V}))(A)$ consists of finite unions of languages of the form

$$L_0 a_1 L_1 a_2 \dots a_\ell L_\ell, \qquad \text{where} \qquad \ell \leq n,\ a_1, \dots, a_\ell \in A,\ L_0, \dots, L_\ell \in \mathcal{V}(A)\ .$$

Further, the union of all $\mathsf{DPol}_n(\mathcal{V}),\ n = 1, 2, \dots,$ is denoted by $\mathsf{DPol}\,(\mathcal{V})$.

The following propositions are direct consequences of our main results.

Proposition 5. *If $\mathcal{V}$ is a disjunctive variety of languages then $\mathsf{DPol}_n(\mathcal{V})$ is a disjunctive variety of languages and the corresponding pseudovariety of idempotent semirings is generated by the set of all semirings $\diamond_n(S_0, \dots, S_n)$, where S_i are syntactic semirings of languages from $\mathcal{V}$.*

Proposition 6. *If $\mathcal{V}$ is a disjunctive variety of languages then $\mathsf{DPol}\,(\mathcal{V})$ is a disjunctive variety of languages and the corresponding pseudovariety of idempotent semirings is generated by the set of all semirings $\diamond_n(S_0, \dots, S_n)$, where S_i are syntactic semirings of languages from $\mathcal{V}$ and $n = 1, 2, \dots$.*

References

1. Klíma, O., Polák, L.: Hierarchies of piecewise testable languages. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 479–490. Springer, Heidelberg (2008)
2. Klíma, O., Polák, L.: Polynomial operators on classes of regular languages. In: Bozapalidis, S., Rahonis, G. (eds.) Algebraic Informatics. LNCS, vol. 5725, pp. 260–277. Springer, Heidelberg (2009)
3. Pin, J.-E.: Varieties of Formal Languages. North Oxford Academic, Plenum (1986)
4. Pin, J.-E.: A variety theorem without complementation. Russian Mathem (Iz. VUZ) 39, 74–83 (1995)
5. Pin, J.-E.: Syntactic semigroups. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, ch. 10, Springer, Heidelberg (1997)
6. Pin, J.-E.: Algebraic tools for the concatenation product. Theoretical Computer Science 292, 317–342 (2003)
7. Pin, J.-E., Straubing, H.: Monoids of upper triangular boolean matrices. In: Colloquia Mathematica Societatis Janos Bolyal, pp. 259–272 (1981)
8. Polák, L.: A classification of rational languages by semilattice-ordered monoids. Arch. Math. (Brno) 40, 395–406 (2004)
9. Rhodes, J., Steinberg, B.: The q-theory of Finite Semigroups. In: Springer Monographs in Mathematics (2009)
10. Simon, I.: The product of rational languages. In: Lingas, A., Carlsson, S., Karlsson, R. (eds.) ICALP 1993. LNCS, vol. 700, pp. 430–444. Springer, Heidelberg (1993)

On Language Equations $XXK = XXL$ and $XM = N$ over a Unary Alphabet*

Tommi Lehtinen[1,2] and Alexander Okhotin[1,3]

[1] Department of Mathematics, University of Turku, Finland
{tommi.lehtinen,alexander.okhotin}@utu.fi
[2] Turku Centre for Computer Science
[3] Academy of Finland

Abstract. It is shown that the recently discovered computational universality in systems of language equations over a unary alphabet occurs already in systems of the simplest form, with one unknown X and two equations $XXK = XXL$ and $XM = N$, where K, L, M, $N \subseteq a^*$ are four regular constants. Every recursive (r.e., co-r.e.) set can be encoded in a unique (least, greatest) solution of a system of such a form. The proofs are carried out in terms of equations over sets of numbers.

1 Introduction

Language equations are among the basic notions of formal language theory, first encountered in connection with defining the context-free grammars. In the recent decade, some further applications have been found, and a lot of theoretical studies have been conducted [10]. In particular, many connections between language equations and computability have been found [9,14,15]. Until recently, nothing was known about the special case of language equations over a one-letter alphabet, beyond the fact that they are nontrivial: this was demonstrated by Leiss [12], who constructed a single example of an equation $X = \varphi(X)$ with a non-periodic solution, where φ uses concatenation, complementation and constant $\{a\}$.

Recently Jeż [2] has introduced a new method of constructing systems of equations of the form

$$\begin{cases} X_1 = \varphi_1(X_1, \ldots, X_n) \\ \quad\vdots \\ X_n = \varphi_n(X_1, \ldots, X_n) \end{cases}$$

where X_i are unknown unary languages and φ_i are expressions containing the operations of union, intersection and concatenation, as well as singleton constant sets. These equations correspond to conjunctive grammars [13] over a one-letter alphabet, and an example of a conjunctive grammar generating $\{a^{4^n} \mid n \geqslant 0\}$, discovered by Jeż [2], has contributed much to the understanding of the expressive power of these grammars. The method of Jeż [2] was further explored by Jeż

* Supported by the Academy of Finland under grants 134860 and 218315.

and Okhotin [3,6], who have subsequently used it [4] to establish computational completeness of equations over sets of numbers of a more general form

$$\begin{cases} \varphi_1(X_1, \ldots, X_n) = \psi_1(X_1, \ldots, X_n) \\ \qquad\qquad\vdots \\ \varphi_m(X_1, \ldots, X_n) = \psi_m(X_1, \ldots, X_n) \end{cases}$$

where both left-hand and right-hand sides may use union, concatenation and singleton constants. To be precise, it was proved that a unary language $L \subseteq a^*$ is represented by a unique (least, greatest) solution of such a system if and only if it is recursive (r.e., co-r.e., respectively).

The next result due to Jeż and Okhotin [5] was a simulation of a system with union and concatenation by a system using concatenation only, such that every solution $X_i = L_i$ of the original system is represented by a solution $X_i = L'_i = \sigma(L_i)$ of the new system, with $a^{16n+13} \in L'_i$ if and only if $a^n \in L_i$. This, in particular, leads to a representation of a language $\sigma(L)$ by unique (least, greatest) solutions of systems, for every recursive (r.e., co-r.e., respectively) language $L \subseteq a^*$. On the other hand, it was proved by Lehtinen and Okhotin [11] that some quite simple languages cannot be specified as they are (without any encoding) by equations using only concatenation, and therefore language equations with concatenation only are slightly less powerful than equations with union and concatenation.

This paper continues exploring simple cases of language equations over a unary alphabet by demonstrating computational universality of systems with only two equations of the form

$$\begin{cases} XXK = XXL \\ XM = N \end{cases} \qquad (*)$$

where X is the unique unknown and $K, L, M, N \subseteq a^*$ are regular constant languages. The final result is stated as follows: for every recursive (r.e., co-r.e.) language $L \subseteq a^*$ there is a system of two equations of the above form with a unique (least, greatest, respectively) solution L', satisfying $a^{np+d} \in L'$ if and only if $a^n \in L$, for some constants $p \geqslant 1$ and $d \geqslant 0$. It is proved by further transforming a system using concatenation only, as produced according to the theorem of Jeż and Okhotin [5]. In Section 3, all variables are encoded into one. Then, in Section 4, all equations for this variable are encoded into two equations (*). The consequences of this encoding are summarized in Section 5, including the following results on the decision problems for systems (*): testing whether such a system has a solution (has a unique solution) is co-r.e.-complete (Π^0_2-complete, respectively).

Some further limitations of the expressive power of these restricted equations are exposed in Sections 6–7. It is shown that some sets are representable by systems with multiple variables using addition only, but not by univariate systems. There are also sets representable by univariate systems of multiple equations, yet not by any individual equation.

The last contribution of the paper concerns with a generalization of language equations over a unary alphabet to the case of a monogenic free group of "strings"

a^ℓ with $\ell \in \mathbb{Z}$, concatenated as $a^\ell a^m = a^{\ell+m}$. The resulting *equations over sets of integers* have recently been studied by Jeż and Okhotin [7], who determined the expressive power of systems of a general form. In Section 8, this result is subjected to the same encoding as in the case of ordinary language equations, obtaining an encoding of every hyperarithmetical set in a unique solution of a system of the form $XXK = XXL$, $XM = N$ with $K, L, M, N \subseteq \{a^\ell \mid \ell \in \mathbb{Z}\}$.

2 Sets of Numbers and Their Encodings

Languages over a unary alphabet can be regarded as sets of natural numbers, with a unary language $L \subseteq a^*$ corresponding to the set $\{n \mid a^n \in L\}$. This is just a different, more compact notation for the same object, and it will be used in all arguments throughout this paper. In this notation, a concatenation of two languages is represented by an elementwise addition of two sets, defined as follows: for any sets $S, T \subseteq \mathbb{N}$, their *sum* is the set $S+T = \{m+n | m \in S, n \in T\}$.

Vectors of sets of numbers are partially ordered by componentwise inclusion, that is $(S_1, \ldots, S_n) \sqsubseteq (S'_1, \ldots, S'_n)$ if $S_i \subseteq S'_i$ for all i. This order is typically applied to the set of solutions of a system of equations, and one is interested in the *least* or the *greatest* solutions with respect to this order.

Equations over sets of numbers using two operations, union and addition, have recently been proved computationally complete.

Proposition 1 (Jeż, Okhotin [4]). *For every recursive (r.e., co-r.e.) set $S \subseteq \mathbb{N}$ there exists a system of equations over sets of natural numbers*

$$\begin{cases} \varphi_1(X_1, \ldots, X_n) = \psi_1(X_1, \ldots, X_n) \\ \qquad\qquad\vdots \\ \varphi_m(X_1, \ldots, X_n) = \psi_m(X_1, \ldots, X_n) \end{cases}$$

with φ_j, ψ_j using singleton constants and the operations of union and addition, which has a unique (least, greatest, respectively) solution with $X_1 = S$.

As a matching upper bound, unique (least, greatest) solutions of equations over sets of numbers with any Boolean operations and addition are known to be recursive (r.e., co-r.e., respectively), so this result precisely characterizes the families of sets representable by solutions of such equations.

For systems of equations over sets of numbers with addition as the only operation, a computational universality result was recently established by Jeż and Okhotin [5]. The idea behind the proof is taking any system as in Proposition 1 (that is, with the operations of union and addition), and simulating it by another system using addition only, so that there is a bijection between the solutions of these systems, with the solutions of the new system encoding the corresponding solutions of the original system. The constructed system manipulates encodings of sets instead of the sets in their original form, and thus simulates both operations in the original system using addition only.

The *encoding of a set* $S \subseteq \mathbb{N}$ is another set $\sigma(S) \subseteq \mathbb{N}$. In general, there is a number $p \geqslant 1$, and the numbers in $\sigma(S)$ are represented in the form $pn + i$, with $i \in \{0, \ldots, p-1\}$ referred as the number of a *track*. One designated track d contains the encoded set S, in the sense that $pn + d \in \sigma(S)$ if and only if $n \in S$. The rest of the tracks are ultimately periodic; typically, they would be either *empty* (with $pn + i \notin \sigma(S)$ for all $n \geqslant 0$) or *full* (if $pn + i \in \sigma(S)$ for all $n \geqslant 0$).

This kind of encoding requires the following notation: for each $S \subseteq \mathbb{N}$, $p \geqslant 1$ and $i \in \{0, 1, \ldots, p-1\}$, define

$$\tau_i^p(S) = \{pn + i \mid n \in S\}.$$

A set of this form will be called a *track*, and it will be said that the set S is encoded *on the i-th track. Full tracks* can then be denoted by $\tau_i^p(\mathbb{N})$.

The actual encoding used by Jeż and Okhotin [5] uses pre-defined values $p = 16$ and $t = 13$, with the encoding function σ defined as follows:

$$\sigma(S) = \{0\} \cup \tau_6^{16}(\mathbb{N}) \cup \tau_8^{16}(\mathbb{N}) \cup \tau_9^{16}(\mathbb{N}) \cup \tau_{12}^{16}(\mathbb{N}) \cup \tau_{13}^{16}(S),$$

for every set $S \subseteq \mathbb{N}$.

Proposition 2 (Jeż, Okhotin [5]). *For every recursive (r.e., co-r.e.) set $S \subseteq \mathbb{N}$ there exists a system of equations*

$$\begin{cases} \varphi_1(X_1, \ldots, X_n) = \psi_1(X_1, \ldots, X_n) \\ \vdots \\ \varphi_m(X_1, \ldots, X_n) = \psi_m(X_1, \ldots, X_n) \end{cases}$$

with φ_j, ψ_j using the operation of addition and ultimately periodic constants, which has a unique (least, greatest, respectively) solution with $X_i = \sigma(S_i)$ for some $S_i \subseteq \mathbb{N}$, of which $S_1 = S$.

Given a Turing machine recognizing S (the complement of S in the case of a greatest solution), such a system can be effectively constructed.

This paper builds upon this result to construct systems of an even more restricted form that encode computationally universal solutions.

3 Encoding into One Variable

The goal of this section is to simulate any given system of equations over multiple variables (such as a system constructed in Proposition 2) by a system with only one variable.

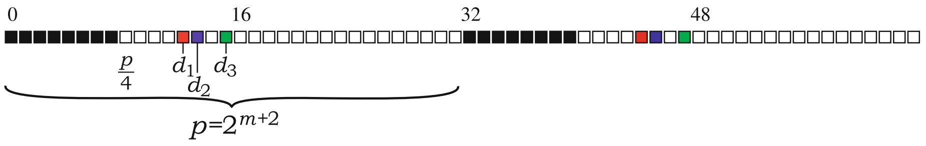
For technical reasons, some assumptions on the form of the simulated system are made. Assume that it has m variables $X_1, X_2, \ldots, X_m$. The equations with a constant-side are $X_i = E_i$ for $i = 1, 2, \ldots, c$, where E_i is an ultimately periodic constant containing zero. The rest of the equations contain only variables and are of the form $X_k + X_\ell = X_{k'} + X_{\ell'}$. Any given system using addition only can be transformed to this form by introducing new variables.

The simulation uses the same idea of *encoding in tracks* as in the proof of Jeż and Okhotin [5] described in the previous section, though this time m sets of numbers shall be encoded in a single set of numbers. Every solution $X_1 = S_1, \ldots, X_m = S_m$ of the original system is encoded into a solution $X = \pi(S_1, \ldots, S_m)$ of the new system of equations. Only those solutions that have zero in all the sets S_i are considered, other solutions are not represented in the new system. This loss of generality is irrelevant, because this simulation shall be applied only to systems with all solutions containing zeroes,

The sets $S_1, \ldots, S_m$ are encoded with $p = 2^{m+2}$ tracks so that the set S_j will be on the track $d_j = \frac{3p}{8} + 2^{j-1} - 1$. The encoding is given by:

$$\pi(S_1, \ldots, S_m) = \bigcup_{i=0}^{\frac{p}{4}-1} \tau_i^p(\mathbb{N}) \cup \bigcup_{j=1}^{m} \tau_{d_j}^p(S_j). \tag{1}$$

The next sketch illustrates this encoding for $m = 3$, where $p = 32$ and $d_1 = 12$, $d_2 = 13$ and $d_3 = 15$.



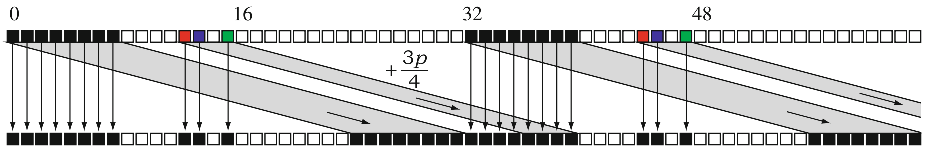
For instance, the membership of 0 in S_3 is encoded by the number 15 in $\pi(S_1, S_2, S_3)$.

This encoding has the following three properties ensuring a correct simulation. The first property is that a set of numbers can be checked for being a valid π-encoding of some m sets by the means of an equation of the form $X + C_1 = C_2$, for the following ultimately periodic constants C_1 and C_2:

Lemma 1. *A set $S \subseteq \mathbb{N}$ is of the form $S = \pi(S_1, \ldots, S_m)$ for some $S_1, \ldots, S_m \subseteq \mathbb{N}$ with $0 \in S_i$ for all i if and only if it satisfies the equation*

$$X + (\tau_0^p(\mathbb{N}) \cup \{\tfrac{3p}{4}\}) = \bigcup_{i=0}^{\frac{p}{4}-1} \tau_i^p(\mathbb{N}) \cup \bigcup_{j=1}^{m} \tau_{d_j}^p(\mathbb{N}) \cup \bigcup_{k=\frac{3p}{4}}^{p-1} \tau_k^p(\mathbb{N}).$$

Since every set S_i contains zero by assumption, the addition of $\tau_0^p(\mathbb{N})$ overwrites each data track to a full track, and preserves the rest of the numbers. Adding $\frac{3p}{4}$ duplicates all tracks, shifting them by this offset. Thus an addition of $\tau_0^p(\mathbb{N}) \cup \{\frac{3p}{4}\}$ to a correct encoding yields the given result:



Conversely, it can be checked that every set satisfying the equation must be a valid π-encoding.

All equations with a constant side in the original system are checked by the following single equation:

Lemma 2. *The sets $0 \in S_i \subseteq \mathbb{N}$ for $i = 1, \dots, c$ satisfy the equations $X_\ell = E_\ell$ for $\ell = 1, \dots, c$ if and only if $\pi(S_1, \dots, S_m)$ satisfies the equation*

$$X + (\tau_0^p(\mathbb{N}) \cup \{p-1-d_c\}) = \\ = \bigcup_{i=0}^{\frac{p}{4}-1} \tau_i^p(\mathbb{N}) \cup \bigcup_{j=1}^{m} \tau_{d_j}^p(\mathbb{N}) \cup \bigcup_{k=p-1-d_c}^{p-1-d_c+\frac{p}{4}-1} \tau_k^p(\mathbb{N}) \cup \bigcup_{\ell=1}^{c} \tau_{p-1-d_c+d_\ell}^p(E_\ell).$$

The equation is similar to the one in Lemma 1; this time the intended values of the constants are embedded in the right-hand side.

Each equation $X_k + X_\ell = X_{k'} + X_{\ell'}$ has a corresponding equation in the new system:

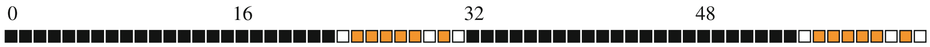
Lemma 3. *The sets $S_1, \dots, S_m \subseteq \mathbb{N}$ satisfy the equation $X_k + X_\ell = X_{k'} + X_{\ell'}$ if and only if $\pi(S_1, \dots, S_m)$ satisfies the equation*

$$X + X + \{0, \tfrac{p}{4}, p-1-d_k-d_\ell\} = X + X + \{0, \tfrac{p}{4}, p-1-d_{k'}-d_{\ell'}\}.$$

Consider the sum of a π-encoding with itself, in which every sum $S_k + S_\ell$ gets to the track $d_k + d_\ell$:

$$\pi(S_1, \dots, S_m) + \pi(S_1, \dots, S_m) = \bigcup_{i=0}^{\frac{3p}{4}-2} \tau_i^p(\mathbb{N}) \cup \bigcup_{k \leqslant \ell} \tau_{d_k+d_\ell}(S_k + S_\ell).$$

Note that the numbers $d_k + d_\ell$ are pairwise different for different (k, ℓ) (cf. the encoding for conjunctive grammars due to Jeż and Okhotin [6], which has the same property), and thus all $\frac{m(m+1)}{2}$ sums $S_k + S_\ell$ get on distinct tracks, as illustrated in the following figure:



Next, the sum $\pi(S_1, \dots, S_m) + \pi(S_1, \dots, S_m) + \{0, \frac{p}{4}\}$ has the empty track $p-1$, with the rest of the tracks full. This empty track is populated by the data from the track $d_k + d_\ell$ by the sum with $p-1-d_k-d_\ell$, and hence the result reflects the set $S_k + S_\ell$ and this set alone.

To sum up the construction, the original system has variables $X_1, \dots, X_m$, equations with constants

$$X_i = E_i, \quad \text{for } i = 1, \dots, c$$

and equations without constants

$$X_k + X_\ell = X_{k'} + X_{\ell'}, \quad (k, \ell, k', \ell') \in V.$$

The new system has only one variable X, equations $X + E_1 = F_1$, $X + E_2 = F_2$, where E_i and F_i are given in Lemmas 1 and 2, and equations

$$X + X + \{0, \tfrac{p}{4}, p - 1 - d_k - d_\ell\} = X + X + \{0, \tfrac{p}{4}, p - 1 - d_{k'} - d_{\ell'}\}$$

for all $(k, \ell, k', \ell') \in V$. The correctness of the construction is stated as follows:

Proposition 3. *A set S is a solution of the constructed system if and only if there are sets $0 \in S_i \subseteq \mathbb{N}$ for $i = 1, \ldots, m$ such that $S = \pi(S_1, \ldots, S_m)$ and $X_i = S_i$ is a solution of the original system.*

4 Encoding into Two Equations

The equations produced in the previous section are defined uniformly and differ only by the constants. They can be merged into four constants as follows.

The system concerned is of the form $\{X + X + C_i = X + X + D_i \mid 0 \leqslant i < m\} \cup \{X + E_i = F_i \mid 0 \leqslant i < m'\}$ with $m \geqslant 0$, $m' \geqslant 1$ and $C_i, D_i, E_i, F_i \subseteq \mathbb{N}$, and assume $E_i \neq \varnothing$ for all i. This system is simulated by a system with only two equations. To do this, let $p = \max(m, m' + 1)$ and define the constants

$$C = \bigcup_{i=0}^{m-1} \{pn + i \mid n \in C_i\}, \qquad D = \bigcup_{i=0}^{m-1} \{pn + i \mid n \in D_i\},$$
$$E = \bigcup_{i=0}^{m'-1} \{pn + i \mid n \in E_i\}, \qquad F = \bigcup_{i=0}^{m'-1} \{pn + i \mid n \in F_i\}.$$

The new system has equations

$$\begin{aligned} X + X + C &= X + X + D, \\ X + E &= F \end{aligned}$$

that simulate all the equations of the original system:

Lemma 4. *The original system has $X = S$ as a solution if and only if $X = S' = \{np | n \in S\}$ is a solution of the new system. All solutions of the new system are of the latter form.*

First, the equation $X + E = F$ implicitly checks that $X = S'$ is a valid encoding of some set, that is, all numbers in S' are zero modulo p, and accordingly $S' = \{np \,|\, n \in S\}$ for some set $S \subseteq \mathbb{N}$. Then the correspondence between the solutions of the original system and of the new pair of equations is ensured in the following way. Each i-th track of the sum $S' + S' + C$ encodes the set $S + S + C_i$, and thus the equation $X + X + C = X + X + D$ implements all equations $X + X + C_i = X + X + D_i$ in the original system. All equations with a constant side are similarly checked by the equation $X + E = F$.

5 Representable Sets and Decision Problems

Theorem 1. *For every recursive (r.e., co-r.e.) set $S \subseteq \mathbb{N}$ there exist numbers $p, d \geqslant 1$, finite sets $C, D \subseteq \mathbb{N}$ and ultimately periodic sets $E, F \subseteq \mathbb{N}$, such that the system of two equations*

$$\begin{cases} X+X+C = X+X+D \\ X+E = F \end{cases}$$

with an unknown $X \subseteq \mathbb{N}$ has a unique (least, greatest, respectively) solution $X = S'$, such that $n \in S$ if and only if $pn+d \in S'$.

Given a Turing machine recognizing S (the complement of S in the case of a greatest solution), such p, d, C, D, E and F can be effectively constructed.

Proof. Consider the system given by Proposition 2 for S, which has variables $X_1, \ldots, X_{\widetilde{m}}$ and all equations of the form $X_{k_1}+\ldots+X_{k_s}+C = X_{k_{s+1}}+\ldots+X_{k_t}+D$. The unique (least, greatest) solution of this system is $(S_1, \ldots, S_{\widetilde{m}})$, where $p_0 n + d_0 \in S_1$ if and only if $n \in S$, where $p_0 = 16$ and $d_0 = 13$. Furthermore, since each S_i is a σ-encoding (as in Proposition 2) of some set, it is known that $0 \in S_i$.

This system can be transformed to a system in variables $X_1, \ldots, X_m$ for some $m \geqslant \widetilde{m}$, and with equations of the form $X_{k_1}+X_{k_2} = X_{k_3}+X_{k_4}$ and $X_1 = C_i, \ldots, X_{m'} = C_{m'}$ for some $1 \leqslant m' \leqslant m$. This is done by separating subexpressions into extra variables, and by permuting the variables so that variables with equations of the form $X_i = C$ have smaller numbers. The unique (least, greatest) solution of the latter system is $(S'_1, \ldots, S'_m)$, with all S'_i containing zero, and there is an index i_0 with $S'_{i_0} = S_1$.

By the constructions in Section 3, there exists a system with a unique variable X and with all equations of the form $X+X+C_i = X+X+D_i$ and $X+E_i = F_i$, and its solutions correspond to the solutions of the previous system in variables $(X_1, \ldots X_m)$ as stated in Proposition 3. In particular, if the previous system has a unique (least, greatest) solution $X_i = S_i$, then $X = \widehat{S} = \pi(S'_1, \ldots, S'_m)$ will be the unique (least, greatest) solution of the constructed system, with $p_1 n + d_1 \in \widehat{S}$ if and only if $n \in S'_{i_0}$, for some $p_1 \geqslant 1$ and $d_1 \geqslant 0$.

Finally, applying Lemma 4 to the latter system gives the system of two equations $X+X+C = X+X+D$ and $X+E = F$, which has a unique (least, greatest) solution $X = S'$, with $p_2 n + d_2 \in S'$ if and only if $n \in \widehat{S}$.

Now $n \in S$ if and only if $p_0 n + d_0 \in S_1$ if and only if $p_1(p_0 n + d_0) + d_1 \in \widehat{S}$ if and only ig $p_2(p_1(p_0 n + d_0) + d_1) + d_2 \in S'$, and setting $p = p_0 p_1 p_2$ and $d = d_0 p_1 p_2 + d_1 p_2 + d_2$ proves the theorem. $\square$

Since the constructions preserve the cardinality of the set of solutions, the decision problems about this cardinality maintain their level of undecidability:

Theorem 2. *The problem of whether a given system of two equations*

$$\begin{cases} X+X+C = X+X+D \\ X+E = F \end{cases}$$

with an unknown $X \subseteq \mathbb{N}$ has a solution (unique solution, finitely many solutions) is Π_1-complete (Π_2-complete, Σ_3-complete), respectively.

These results can be equivalently phrased in terms of language equations over a unary alphabet:

Corollary 1. *For every recursive (r.e., co-r.e.) language $S \subseteq a^*$ there exist numbers $p, d \geqslant 1$, finite languages $K, L \subset a^*$ and regular languages $M, N \subseteq a^*$, such that the system of two language equations*

$$\begin{cases} XXK = XXL \\ XM = N \end{cases}$$

with a variable $X \subseteq a^*$ has a unique (least, greatest, respectively) solution $X = S'$, such that $a^n \in S$ if and only if $a^{pn+d} \in S'$.

Given a Turing machine recognizing S (the complement of S in the case of a greatest solution), such p, d, K, L, M and N can be effectively constructed.

Testing whether a given system of the above form has a solution (unique solution, finitely many solutions) is Π_1-complete (Π_2-complete, Σ_3-complete).

The full path from a Turing machine to a system of two equations involves representing the set of its computation histories by a cellular automaton working on positional notations [3], representing the corresponding set of numbers by equations with union and addition [4], simulating them by a system with addition only [5], and then encoding the latter in two equations as described in this paper.

6 Limitations of One Variable

It was recently proved by the authors [11] that systems of equations with multiple variables using addition only cannot represent any sets satisfying the following two conditions. The first condition is being *prime*, in the sense of having no nontrivial representation as a sum of two sets:

Definition 1. *A set $S \subseteq \mathbb{N}$ is prime if $S = S_1 + S_2$ implies $S_1 = \{0\}$ or $S_2 = \{0\}$.*

The second condition is being *fragile*, which means that the sum of this set with any set containing at least two elements is co-finite.

Definition 2. *A set $S \subseteq \mathbb{N}$ is fragile if $S + \{n_1, n_2\}$ is co-finite for all $n_1, n_2 \in \mathbb{N}$ with $n_1 \neq n_2$.*

This definition is equivalent to the statement that for every $k \geqslant 1$ there are only finitely many numbers $n \in \mathbb{N}$ with $n, n + k \notin S$.

Proposition 4 (Lehtinen, Okhotin [11]). *No set that is prime and fragile is representable by systems of equations over sets of natural numbers with operation of addition and ultimately periodic constants. There exist recursive (and in fact computationally easy) sets that are prime and fragile.*

At the same time, some fragile sets that are not prime can be represented [11]. It turns out that none of these sets are representable using a single variable.

Lemma 5. *If a fragile set is the greatest solution of a univariate system, then it is co-finite.*

Lemma 6. *If a fragile set is the least solution of a univariate system, then it is co-finite.*

Since the sum of a fragile set with anything nontrivial is a co-finite set, a number large enough can be included or removed from this fragile set without affecting the value of the sum. The proofs of both lemmata proceed by supposing that there is such a greatest (least) solution, and by constructing a greater (lesser) solution according to this property of fragile sets, thus coming to a contradiction.

Theorem 3. *There exists a set of natural numbers representable by a unique solution of a multivariate system of equations with addition, which, however, is not a unique (least, greatest) solution of any univariate system.*

7 Limitations of One Equation

Systems of two equations

$$\begin{cases} X + X + C = X + X + D \\ \qquad X + E = F \end{cases}$$

constructed in Theorems 1 and 2 have one equation with a constant side and one equation without constant sides. It turns out that systems with all equations of the same type (that is, either all with constant sides or all without constant sides) have quite limitated expressive power.

If all equations in a system are without constant sides, their least solution is trivial: $X_i = \varnothing$. Greatest solutions are bound to be trivial as well: as shown in the next lemma, they are always co-finite.

Lemma 7. *If a system of equations of the form $X_{i_1} + \ldots + X_{i_\ell} + C = X_{j_1} + \ldots + X_{j_m} + D$ has a solution $(S_1, \ldots, S_n)$, then $(S_1 + \mathbb{N}, \ldots, S_n + \mathbb{N})$ is a solution as well.*

Proof. Consider the smallest number in $S_{i_1} + \ldots + S_{i_\ell} + C$. Then $(S_{i_1} + \mathbb{N}) + \ldots + (S_{i_\ell} + \mathbb{N}) + C$ contains this number and all numbers that are greater. The claim follows, since all equations have the same smallest numbers in the both sides. □

The other type of systems have all equations with constant sides. It can be shown that every non-periodic solution can be extended to a greater periodic solution, which will have the same period as the common period of the constant sides.

Lemma 8. *If a system of equations of the form $X_1 + \ldots + X_m + E = F$ has a solution $(S_1, \ldots, S_n)$, then it has an ultimately periodic solution $(S'_1, \ldots, S'_n)$ with $S_i \subseteq S'_i$.*

Such a statement holds in a much more general case of language equations, and can be inferred from the syntactic monoid and Conway's [1] results.

Proof. Let p and d be numbers, such that for every equation $X_1+\ldots+X_m+E = F$, the set F has period p starting from d. Define $S_i' = S_i \cup \big([S_i \cap (\mathbb{N}+d)] + \{np \mid n \geqslant 0\}\big)$. Then it is easy to check that $X_i = S_i'$ is still a solution. $\square$

Such an argument does not work for least solutions. In fact, it is easy to construct an equation with uncountably many pairwise incomparable solutions [8]: the equation

$$X + \{0, 1\} = \mathbb{N}$$

has the following sets as minimal solutions:

$$\{n \mid \text{the } (n+1)\text{-th bit of } w \text{ is } 1\} \quad (\text{for any } w \in \{10, 110\}^\omega).$$

All these solutions are pairwise incomparable. In case there exists a least solution (or at least countably many solutions), no examples of nontrivial expressive power are known.

8 On Equations over Sets of Integers

Consider systems of equations over set of numbers, in which the constants and the unknowns may contain negative numbers as well, that is, can be any subsets of $\mathbb{Z}$. Such equations have recently been studied by Jeż and Okhotin [7], who determined the expressive power of systems using the operations of union and addition, and ultimately periodic constants. The unique solutions of these equations can represent exactly the *hyperarithmetical sets*, $\Sigma_1^1 \cap \Pi_1^1$ [16, Ch. 16], which strictly include the sets representable in first-order Peano arithmetic, and form the bottom of the analytical hierarchy.

This result is a variant of Proposition 1 for equations over sets of integers. There is an analogue of Proposition 2 as well, which allows encoding any hyperarithmetical set in a unique solution of a system of equations over sets of integers with addition and ultimately periodic constants [7]. Since the general course of the argument is the same as for natural numbers, the methods of this paper can be used to encode the resulting systems further to the following simple form as in Theorem 1:

Theorem 4. *For every hyperarithmetical set $S \subseteq \mathbb{Z}$ there exist numbers $p, d \geqslant 1$, finite sets $C, D \subseteq \mathbb{N}$ and ultimately periodic sets $E, F \subseteq \mathbb{Z}$, such that the system of two equations*

$$\begin{cases} X + X + C = X + X + D \\ X + E = F \end{cases}$$

with an unknown $X \subseteq \mathbb{Z}$ has a unique solution $X = S'$, such that $n \in S$ if and only if $pn + d \in S'$.

Some limitations of equations over sets of integers using addition only have recently been established by the authors [11]: analogously to Proposition 4, if a set $S \subseteq \mathbb{Z}$ is prime and its positive part $S \cap \mathbb{N}$ is fragile, then it cannot be represented by a least solution of a system with addition.

For the univariate case, the argument in Lemma 6 can be applied to show that no set of integers with a fragile positive part can be represented as least solution of a system with a single variable using addition.

References

1. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall, Boca Raton (1971)
2. Jeż, A.: Conjunctive grammars can generate non-regular unary languages. International Journal of Foundations of Computer Science 19(3), 597–615 (2008)
3. Jeż, A., Okhotin, A.: Conjunctive grammars over a unary alphabet: undecidability and unbounded growth. Theory of Computing Systems 46(1), 27–58 (2010)
4. Jeż, A., Okhotin, A.: On the computational completeness of equations over sets of natural numbers. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 63–74. Springer, Heidelberg (2008)
5. Jeż, A., Okhotin, A.: Equations over sets of natural numbers with addition only. In: STACS 2009, Freiburg, Germany, February 26-28, pp. 577–588 (2009)
6. Jeż, A., Okhotin, A.: One-nonterminal conjunctive grammars over a unary alphabet. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 191–202. Springer, Heidelberg (2009)
7. Jeż, A., Okhotin, A.: On equations over sets of integers. In: STACS 2010, Nancy, France, March 4-6, pp. 477–488 (2010)
8. Karhumäki, J., Kunc, M.: Personal Communication (September 2005)
9. Kunc, M.: The power of commuting with finite sets of words. Theory of Computing Systems 40(4), 521–551 (2007)
10. Kunc, M.: What do we know about language equations? In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 23–27. Springer, Heidelberg (2007)
11. Lehtinen, T., Okhotin, A.: On equations over sets of numbers and their limitations. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 360–371. Springer, Heidelberg (2009)
12. Leiss, E.L.: Unrestricted complementation in language equations over a one-letter alphabet. Theoretical Computer Science 132, 71–93 (1994)
13. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics 6(4), 519–535 (2001)
14. Okhotin, A.: Strict language inequalities and their decision problems. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 708–719. Springer, Heidelberg (2005)
15. Okhotin, A.: Decision problems for language equations. Journal of Computer and System Sciences 76(3-4), 251–266 (2010)
16. Rogers Jr., H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)

Around Dot Depth Two

Kamal Lodaya[1], Paritosh K. Pandya[2], and Simoni S. Shah[2]

[1] The Institute of Mathematical Sciences, CIT Campus, Chennai *600113*, India
[2] Tata Institute of Fundamental Research, Colaba, Mumbai *400005*, India

Abstract. It is known that the languages definable by formulae of the logics $FO^2[<,S]$, $\Delta_2[<,S]$, $LTL[\mathsf{F},\mathsf{P},\mathsf{X},\mathsf{Y}]$ are exactly the variety $DA * D$. Automata for this class are not known, nor is its precise placement within the dot-depth hierarchy of starfree languages. It is easy to argue that $\Delta_2[<,S]$ is included in $\Delta_3[<]$; in this paper we show that it is incomparable with $\mathcal{B}(\Sigma_2)[<]$, the boolean combination of $\Sigma_2[<]$ formulae. Using ideas from Straubing's "delay theorem", we extend our earlier work [LPS08] to propose partially-ordered two-way deterministic finite automata with look-around (*po2dla*) and a new interval temporal logic called LITL and show that they also characterize the variety $DA * D$. We give effective reductions from LITL to equivalent *po2dla* and from *po2dla* to equivalent $FO^2[<,S]$. The *po2dla* automata admit efficient operations of boolean closure and the language non-emptiness of *po2dla* is NP-complete. Using this, we show that satisfiability of LITL remains NP-complete assuming a fixed look-around length. (Recall that for $LTL[\mathsf{F},\mathsf{X}]$, it is PSPACE-hard.)

A rich set of correspondences has been worked out between diverse mechanisms for defining the first-order definable word languages and their subclasses (a recent survey is [DGK08]). In the following, CFA refers to counter-free automata, SFRE to star-free regular expressions and Ap refers to the variety of aperiodic monoids [Pin86].

$$CFA \equiv SFRE \equiv Ap \equiv FO[<] \equiv LTL[\mathsf{U},\mathsf{S}] \equiv ITL$$

Further, Thomas showed [Tho82] that by restricting the quantifier-alternation depth in the $FO[<]$ formulae a strict dot-depth hierarchy of star-free languages is obtained, see the paper by Pin and Weil [PW97] for details. For example, $\mathcal{B}(\Sigma_2)[<]$ is the class of languages defined by the boolean combination of $\Sigma_2[<]$ formulae, which are the ones which have one block of existential quantifiers followed by one block of universal quantifiers followed by a quantifierless formula.

For the FO formulations below, given an alphabet A and $a \in A$, the unary predicate $Q_a(x)$ holds iff the letter at position x is a. The binary predicate $S(x,y)$ denotes the successor relation on positions, and $<$ is, as usual, its transitive closure.

Example 1. Let $A = \{a,b\}$ be the alphabet described by $\phi_A \overset{\text{def}}{=} \forall x.\ Q_a(x) \vee Q_b(x)$, which will be an additional conjunct below, not explicitly mentioned.

- $\phi_1 \overset{\text{def}}{=} \exists x \exists y.\ S(x,y) \wedge Q_a(x) \wedge Q_a(y)$ is a $\mathcal{B}(\Sigma_1)[S]$ formula defining $L_1 = A^*aaA^*$.
- $\phi_2 \overset{\text{def}}{=} \exists x \exists y.\ Q_a(x) \wedge Q_a(y) \wedge \forall z.\ (x < z \supset y \leq z)$ is a $\Sigma_2[<]$ formula defining L_1.

Y. Gao et al. (Eds.): DLT 2010, LNCS 6224, pp. 303–315, 2010.

– Let $\phi_3 \overset{\text{def}}{=} (\forall x.\, \mathit{first}(x) \supset Q_a(x)) \wedge (\forall x.\, \mathit{last}(x) \supset Q_b(x)) \wedge$
$(\forall x,y.\, ((x<y) \wedge Q_a(x) \wedge Q_a(y) \supset \exists z.\, x<z \wedge z<y \wedge Q_b(z))) \wedge$
$(\forall x,y.\, ((x<y) \wedge Q_b(x) \wedge Q_b(y) \supset \exists z.\, x<z \wedge z<y \wedge Q_a(z)))$.
Then, ϕ_3 is a $\Pi_2[<]$ formula defining the language $L_2 = (ab)^*$. □

More recently, Thérien and Wilke [TW98] showed that the 2-variable fragment $FO^2[<]$ [Mor75] (where only two variables occur, quantified any number of times), is expressively equivalent to the unambiguous languages and variety DA of Schützenberger [Sch76, TT02] and the subset $\Delta_2[<]$ in the dot-depth hierarchy. Etessami, Vardi and Wilke [EVW02] identified the unary temporal logic $LTL[\mathsf{F},\mathsf{P}]$ and Schwentick, Thérien and Vollmer [STV02] identified partially-ordered 2-way deterministic finite automata (these are also called linear [LT00]) as equivalent formalisms. In [LPS08], we added to these correspondences a “deterministic” interval temporal logic called *UITL*. The papers [TW98, EVW02] also characterized $FO^2[<,S]$, which can define languages not definable in the logic $FO^2[<]$ such as those in Example 1. For a detailed study of these logics, see the recent papers of Weis and Immerman [WI07], and of Kufleitner and Weil [KW09].

$$PO2DFA \equiv UL \equiv DA \equiv FO^2[<] \equiv \Delta_2[<] \equiv LTL[\mathsf{F},\mathsf{P}] \equiv UITL$$

$$DA * D \equiv FO^2[<,S] \equiv \Delta_2[<,S] \equiv LTL[\mathsf{F},\mathsf{P},\mathsf{X},\mathsf{Y}]$$

It is clear that $\Delta_2[<,S] \subseteq \Delta_3[<]$ since successor can be defined using $<$ and one quantifier. In this paper we provide an automaton characterization and an interval logic characterization for this class of languages, and we separate it from $\mathcal{B}(\Sigma_2)[<]$, the languages defined by the boolean combination of $\Sigma_2[<]$ formulae. This also shows that $FO^2[<,S]$ is a *proper* subset of $\Delta_3[<]$, as diagrammatically depicted below.

$$\begin{tikzcd}
 & \Delta_2[<,S] = FO^2[<,S] \arrow[ddrr] & & \\
 & \Pi_2[<] \arrow[dr] & & \\
\Delta_2[<] = FO^2[<] \arrow[uur] \arrow[ur] \arrow[r] & \Sigma_2[<] \arrow[r] & \mathcal{B}(\Sigma_2)[<] \arrow[r] & \Delta_3[<]
\end{tikzcd}$$

Our automaton and logic characterizations are based on Rhodes expansions [Til76]; the two-sided variant below is inspired by Straubing's theorem $DA * D \equiv DA * LI$ [Str85].

Definition 1. *Let A be a finite alphabet, $A' = A \cup \{\triangleright, \triangleleft\}$ be its extension with two endmarkers $\triangleright, \triangleleft \notin A$, and $A^\rho_d = (A')^{2d+1}$ the alphabet whose letters are actually words of length $2d+1$ over A. Let $w = w_1 w_2 \ldots w_n$ be a given word, where $w_i \in A$ is a letter. Let $around_d(w,i) = w_{i-d} \ldots w_i \ldots w_{i+d}$ denote the* two-sided d-lookaround string *at position i. Note that if the position i is near one of the endpoints then $around_d(w,i)$ is padded by repeating the endmarker at that end. We define the **Rhodes-Straubing d-expansion** of w (and for a language L pointwise) for $d \geq 1$ to be $w^\rho_d = u_1 u_2 \ldots u_n$, where each $u_i = around_d(w,i)$. This is a word over A^ρ_d. When $d = 0$ we let w^ρ_0 be w. For example, $(abcab)^\rho_2$ is $(\triangleright\triangleright abc)(\triangleright abca)(abcab)(bcab\triangleleft)(cab\triangleleft\triangleleft)$.* □

Straubing's delay theorem shows that a language, or in our context a formula ϕ of $FO^2[<,S]$, can be seen as a formula ϕ' of $FO^2[<]$ over a Rhodes-Straubing d-expansion where d is the number of occurrences of successor predicates in ϕ. Carrying this intuition to automata, we extend *po2dfa* to partially-ordered 2-way deterministic finite state automata with lookaround (*po2dla*) which essentially make transitions on the Rhodes-Straubing expansion of the word. We also extend our unambiguous interval logic *UITL* to an unambiguous interval logic with lookaround called LITL. With some amount of technical hacking, we are able to show that LITL and *po2dla* have the expressive power of $FO^2[<,S]$.

The resulting automata and interval logic have many interesting features. A significant property of *po2dla* is that the boolean operations (including complementation) can be done within *po2dla* with a linear blowup in size. Language emptiness of *po2dla* is NP-complete and inclusion between *po2dla* is CoNP-complete, assuming a fixed lookaround size k.

The logic LITL inherits the desirable properties of its ancestor *UITL* [LPS08]. It admits unique parsability of models and exploiting this we can provide an efficient PTIME reduction from LITL to *po2dla*. This immediately gives us a small model property for the logic. Moreover, given a formula of length n with alphabet size m and lookaround length k, we can show that the satisfiability problem is in nondeterministic time $O((m^k) \times n)$. Assuming fixed lookaround size k, satisfiability is NP-complete. By comparison, the satisfiability of the logic $LTL[\mathsf{F},\mathsf{X}]$ is PSPACE-hard, although an action-indexed version was shown NP-complete by Muscholl and Walukiewicz [MW05].

The rest of the paper is organized as follows: the next section defines our automata, Section 2 the logic and the reductions from logic to automata and from automata to $FO^2[<,S]$. Section 3 deals with expressiveness and finally brings us back from $FO^2[<,S]$ to our logic.

1 Partially-Ordered Two-Way DFA with Look-Around

Fix an alphabet A and its extension $A' = A \cup \{\triangleright, \triangleleft\}$ with two endmarkers $\triangleright, \triangleleft \notin A$. Given $w \in A^*$, let $dom(w) = \{1, \ldots, |w|\}$. In recognizing w, the two-way automaton actually scans the string $w' = \triangleright w \triangleleft$ with letters $\triangleright$ and $\triangleleft$ at positions 0 and $|w|+1$ respectively. Thus, $dom(w') = \{0, \ldots, |w|+1\}$.

Let $a \in A'$ and let $u, v \in A^*$. We shall consider **patterns** of the form $u\underline{a}v$ with an underlined distinguished position. Given a pattern $u\underline{a}v$ and a word w', a position $i \in dom(w')$ matches the pattern, denoted $(w'[*,i,*] = u\underline{a}v)$, if the letter in w' at position i is a and this is followed by the string v (forward lookaround) and also this a is preceded by the string u (backward lookaround). Formally, $(w'[*,i,*] = u\underline{a}v)$ iff $w'[i] = a$ and $\forall k \in dom(v).\ i+k \in dom(w') \wedge w'[i+k] = v[k]$ and $\forall k \in dom(u).\ i-k \in dom(w') \wedge w'[i-k] = u[k]$. (When clear from the context, $u\underline{a}v$ will be written as uav).

For a string u, let $Pre(u)$ and $Suf(u)$ be the set of all prefixes and suffixes (respectively) of u. Given two patterns $u_1\underline{a_1}v_1$ and $u_2\underline{a_2}v_2$, we say that they are overlapping iff (i) $a_1 = a_2$, (ii) $u_1 \in Suf(u_2)$ or $u_2 \in Suf(u_1)$, and (iii) $v_1 \in Pre(v_2)$ or $v_2 \in Pre(v_1)$.

1.1 Automaton Definition

Partially ordered two-way DFA were introduced by Schwentick, Thérien and Vollmer [STV02] to characterize the unambiguous languages. We present a generalization with forward and backward lookaround. The transitions of the automaton are labelled by patterns over the alphabet, instead of letters. There is a default else transition associated with each state which is taken if no other transition is applicable. This makes our automata total.

Definition 2. *A* ***partially ordered 2DFA*** *(po2dla)* ***with lookaround size*** *k over A' is a tuple $M = (Q, \leq, \delta, s, t, r)$ where $(Q, \leq)$ is a finite partial order of states with distinct start, accept and reject states s, t and r where r and t are the only minimal elements of the poset and s is the only maximal element. Let $\mathcal{P}$ be the set of all patterns with a maximum lookaround of size k, i.e. the set of all $u\underline{a}v$ such that $u, v \in A^*$, $a \in A'$ and $|u|, |v| \leq k$. The transition function δ has two types of transitions: the* matching *transitions form a partial function $\delta_m : (Q \setminus \{t, r\} \times \mathcal{P}) \to (Q \times \{L, R, X\})$ where the first component q' of $\delta_m(q, u)$ satisfies $q' < q$, and the default* else *transition is a total function $\delta_{else} : (Q \setminus \{t, r\}) \to (Q \times \{L, R\})$ where the first component q' satisfies $q' \leq q$.*

Further, for determinism we have that, for all $q \in Q$, and $u_1\underline{a_1}v_1, u_2\underline{a_2}v_2 \in \mathcal{P}$, if $\delta_m(q, u_1\underline{a_1}v_1) = q_1$ and $\delta_m(q, u_2\underline{a_2}v_2) = q_2$ such that $q_1 \neq q_2$, then $u_1\underline{a_1}v_1$ and $u_2\underline{a_2}v_2$ are not overlapping. To ensure that the head of the automaton does not "fall beyond" the end-markers, we have an additional syntactic condition:

$$\forall q \in Q \setminus \{t, r\} \,.\, \exists q', q'' \in Q.\ \delta_m(q, \rhd) = (q', R) \text{ and } \delta_m(q'', \lhd) = (q'', L). \qquad \square$$

A configuration of automaton M running on word w' is a pair (q, p) with $q \in Q$, $p \in dom(w')$. The automaton in a configuration (q, p) takes the unique δ_m transition from q, whose label is matched at the position p. If such a transition does not exist, then the automaton takes the default transition δ_{else} where it must change position.

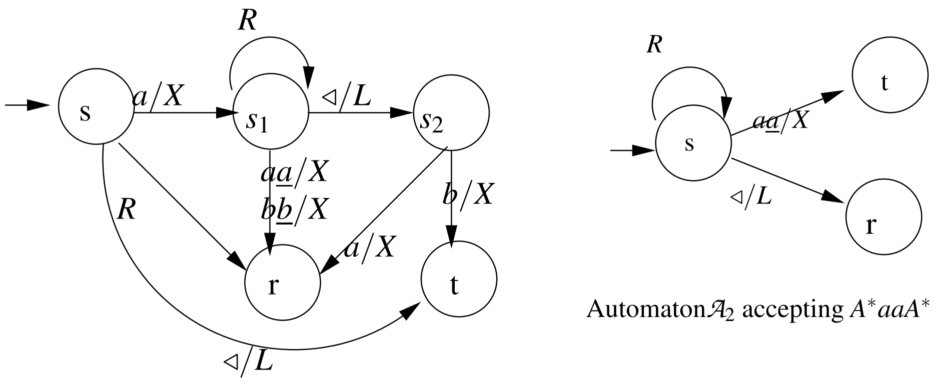
Run. The *run* of the automaton M on a word w' and starting at a position p_0, is a sequence of state-position configurations $(q_0, p_0), (q_1, p_1)...(q_n, p_n)$, where

- $q_0 = s$ and $q_n \in \{t, r\}$. The run is accepting if $q_n = t$ and rejecting if $q_n = r$.
- For all $i \geq 0$, if there exists (unique) $u\underline{a}v$ such that $\delta_m(q_i, u\underline{a}v) = (q', d)$ for some (q', d) and $(w[*, i, *] = uav)$, then (a) $q_{i+1} = q'$ and (b) $p_{i+1} = p_i + 1$ if $d = R$, $p_{i+1} = p_i - 1$ if $d = L$ and $p_{i+1} = p_i$ if $d = X$.
- Otherwise, $q_{i+1} = q'$, where $\delta_{else}(q_i) = (q', d)$, and $p_{i+1} = p_i + 1$ if $d = R$ and $p_{i+1} = p_i - 1$ otherwise.

The outcome of the run is given by the total function $[[M]]$ such that for any $w \in A^*$ and $i \in dom(w')$ is given by $[[M]](w, i) = (q_n, p_n)$, the final configuration of the run. A word w is accepted by M if the unique run of M on $w' = \rhd w \lhd$ starting at position 1 is accepting. The language $\mathcal{L}(M) \subseteq A^*$ is the set of words accepted by M. $\square$

Since the states of M are partially ordered, the only loops allowed are self-loops on the default else transitions. During a sequence of such self-loop transitions the automaton moves in the same direction. Moreover, the automaton must change state on reaching an endmarker. So, because of the partial order, a *po2dla* cannot loop infinitely: it has at most $|Q|-1$ reversals and all its runs are bounded by length $|Q| \times |w|$. Since δ_{else} is a total function, the automaton always has a terminating run on every word: hence the automaton is complete.

Example 2. Figure 1 gives the *po2dla* for the languages $(ab)^*$ and A^*aaA^*. The default else transitions are shown with just a direction. In the automaton A_1, two consecutive a's or b's lead to rejection from state s_1, and in state s_2 which is reached at the end of the word, we check that it ends with b.



Automaton $\mathcal{A}_1$ accepting $(ab)^*$

Automaton $\mathcal{A}_2$ accepting A^*aaA^*

Fig. 1.

Proposition 1. *The po2dla are closed under sequential composition and Boolean operations, constructible with automata of linear size (number of states).*

The proofs follow our earlier paper [LPS08]. Just as we have there, the automata can be described by a syntax of **extended turtle expressions** going beyond those of Schwentick, Thérien and Vollmer [STV02]. We omit these because of lack of space.

1.2 Small Model Property and Decision Problems

We let $INTV(w) = \{[i, j] \mid i, j \in dom(w), i \leq j\}$ be the set of intervals over w, and $w[i, j]$ (or $w, [i, j]$ in the next section) denote the factor of w corresponding to the interval $[i, j]$. We will extend this notation to open and semi-open intervals as usual, as well as to their unions.

Consider a *po2dla* M over an alphabet A with n states and a maximum lookaround of k. Recall that $A' = A \cup \{\triangleright, \triangleleft\}$ and for $w \in A^*$ we have $w' = \triangleright w \triangleleft$. Recall also the defintion of $around_d(w, i)$ given in Definition 1. When clear from the context, we will abbreviate this by $around(w, i)$ or $around(i)$.

Lemma 1 (Membership). *Given a word $w \in A^*$, checking whether $w \in L(M)$ can be carried out by simulating the automaton in deterministic time $O(mnk)$ where m is the number of states of M, n is the length of the word w' and k is the lookaround size.*

Proof. Lookaround is handled by maintaining an array of length $2k+1$ storing the factor around the current head position. Note that there can be at most $m-1$ reversals in the *po2dla*. The algorithm requires space $\log m + \log n + (2k+1)\log|A'|$. □

Now consider the unique run of M accepting w. We say that a position $p \in dom(w')$ is purely-self-looping (PSL) if for all configurations of the form (q,p) in the run having position p, the (unique) enabled transition of M is the *self-looping else transition* (which does not result in change of state).

Call an interval $[m_1,m_2] \in INTV(w)$ a tunnel if all $j \in [m_1,m_2]$ are purely-self-looping (PSL) and $around(w',m_1) = around(w',m_2)$. If the automaton makes a right move at position m_1, it continues moving right without change of state till it reaches m_2; and similarly for a left move at m_2. The following lemma is a direct result of the above and the fact that $around(m_1) = around(m_2)$.

Lemma 2. *Given w' and a tunnel $[m_1,m_2]$, let $v' = w'[0,m_1)[m_2,|w'|]$ be the word obtained by replacing the tunnel by its last letter. Then, $w \in L(M)$ iff $v \in L(M)$.* □

From the above lemma, it is clear that every tunnel in word w' can be collapsed into a single letter preserving membership. Thus, in a word without tunnels, there can be a consecutive sequence of PSL positions which has length at most $|A'|^{2k+1}$ (the number of distinct $around(i)$). Every such sequence must be separated by a non-PSL position. There can be at most $n-1$ non-PSL positions in a run since there can be at most $n-1$ state changing transitions in an n state *po2dla*. Hence, we get the following theorem.

Theorem 1 (Small model). *If $L(M) \neq \emptyset$ then there exists a word $w \in L(M)$ of length at most $(|A'|^{2k+1}+1)(n-1)$.* □

Corollary 1. *Assuming lookaround k to be constant, the language non-emptiness of po2dla is NP-complete and the language inclusion of po2dla is CoNP-complete.*

Proof. The technique is to guess the member word of size $(|A'|^{2k+1}+1)(n-1)$ non-deterministically, and to use the PTIME membership checking algorithm on this. Thus, non-emptiness is in NP. The non-emptiness problem for *po2dfa* is shown to be NP-hard [SP09]. Since *po2dla* are extension of *po2dfa*, we conclude that their non-emptiness problem is NP-complete. We also conclude that the language inclusion problem is CoNP-complete as intersection and complementation of *po2dla* cause only linear blowup in the automaton size, and $L_1 \subseteq L_2$ iff $L_1 \cap \overline{L_2} = \emptyset$. □

2 Logic LITL

Interval temporal logic is based on a *chop* operator which divides an interval into two. Although this yields succinct formulae, the complexity of satisfiability is nonelementary. We proposed unambiguous interval temporal logic [LPS08] replacing chops by marked chop operators F_a and L_a, dividing a given interval at the first/last occurrence of the letter a. Satisfiability of *UITL* is NP-complete. Here we have a simple generalization, chopping an interval at the first/last occurrence of a given pattern $u\underline{a}v$.

Fix an alphabet A. Let $a \in A$ and $u, v \in A^*$. Let D, D_1, D_2 range over formulas in LITL. The abstract syntax of LITL is given below. Here $\top$ denotes the formula *true*.

$$\top \mid pt \mid D_1 \vee D_2 \mid \neg D \mid D_1 F_{u\underline{a}v} D_2 \mid D_1 L_{u\underline{a}v} D_2 \mid \oplus D \mid \ominus D$$

The satisfaction of a formula D is defined over intervals of a word model w as follows. As usual, $w \models D$ iff $w, [1, |w|] \models D$ and $L(D) \overset{\text{def}}{=} \{w \mid w \models D\}$ is the language defined by D. The derived operators $\wedge, \supset, \Leftrightarrow$ have their usual definitions.

$w, [i, j] \models pt$ iff $i = j$
$w, [i, j] \models D_1 F_{u\underline{a}v} D_2$ iff for some $k : k \in [i, j].\ (w[*, k, *] = u\underline{a}v)$ and
for all $m : i \leq m < k.\ \neg(w[*, m, *] = u\underline{a}v)$ and
$w, [i, k] \models D_1$ and $w, [k, j] \models D_2$
$w, [i, j] \models D_1 L_{u\underline{a}v} D_2$ iff for some $k : k \in [i, j].\ (w[*, k, *] = u\underline{a}v)$ and
for all $m : k < m \leq j.\ \neg(w[*, m, *] = u\underline{a}v)$ and
$w, [i, k] \models D_1$ and $w, [k, j] \models D_2$
$w, [i, j] \models \oplus D$ iff $i < j$ and $w, [i+1, j] \models D$
$w, [i, j] \models \ominus D$ iff $i < j$ and $w, [i, j-1] \models D$

Example 3. The LITL formula $\top F_{\underline{a}a} \top$ precisely specifies the language A^*aaA^*. The formula $(pt\ F_a \top) \wedge (\top L_b\ pt) \wedge \neg(\top F_{\underline{a}a} \top) \wedge \neg(\top F_{\underline{b}b} \top)$ specifies the language $(ab)^+$ over alphabet $\{a, b\}$. The first and the second conjunct state that the word begins with a and it ends with b. The last two conjuncts state that subwords aa or bb do not occur within the word.

2.1 Unique Parsability and Reduction to Automata

As for its ancestor *UITL* [LPS08], every word model of a *LITL* formula can be uniquely parsed. Fix an LITL formula ϕ. Consider its subformula ψ occuring in context λ; we denote this by $\phi = \lambda(\psi)$. For any $w \in A^+$, we can uniquely determine if ψ is *relevant* in determining truth of ϕ over w. Moreover, if relevant, we can uniquely assign an interval $Intv_w(\psi)$ such that the truth value of ψ only over this interval is relevant in determining the truth of ϕ over w. The interval $Intv_w(\psi)$ actually depends only on the context λ and not on ψ. Moreover, it is possible to construct *po2dla* $\mathcal{L}(\psi)$ and $\mathcal{R}(\psi)$ which accept at the left and right interval boundaries of $Intv_w(\psi)$ respectively if the subformula is relevant. Using these automata, we can further construct an automaton $M(\psi)$ which accepts if ψ is relevant and it evaluates to true on $Intv_w(\psi)$. Exploiting this unique parsability, the following theorem can be established as a straightforward generalization of the similar theorem for logic *UITL* [LPS08].

Theorem 2. *For any $D \in LITL$ we can effectively construct a po2dla $M(D)$ in polynomial time such that $w \in \mathcal{L}(D) \Leftrightarrow w \in \mathcal{L}(M(D))$. The size $|M(D)| = O(|D|^2)$.*

Proof (sketch). The construction of $M(D)$ is inductive and proceeds bottom-up on the structure of D. Consider $D = \psi_1 F_{u\underline{a}v} \psi_2$. The corresponding *po2dla* $M(D)$ first moves to the left boundary of $Intv_w(D)$, then it checks in a single pass (moving in one direction only), for the existence of first $u\underline{a}v$, and also checks whether it lies within the right boundary of the interval $Intv_w(\psi)$. It then invokes the automata $M(\psi_1)$ and $M(\psi_2)$ in sequence. The details of the construction can be found in the full paper. □

Decision problems. The above translation gives a PTIME reduction from LITL formula of size n to a language equivalent automaton of size (i.e. number of states) $O(n^2)$. Moreover, the lookaround size in automaton is at most the pattern size in the LITL formula. Combining this with NP-complete non-emptiness checking of *po2dla*, we conclude that satisfiability of LITL is NP-complete assuming a fixed lookaround size. Our previous paper [LPS08] gave a LOGDCFL procedure for checking membership for logic *UITL*. This procedure extends to logic LITL with the same complexity.

2.2 From *po2dla* to $FO^2[<,S]$

In this section, we outline a language preserving translation from *po2dla* to $FO^2[<,S]$. Essentially the automaton is a dag with self-loops added on some nodes. For each progress edge $e = (p,\alpha,q,dir), p \neq q$, we define $FO^2[<,S]$ formulae $At_e(x)$ and $After_e(x)$ with one free variable x. These formulae satisfy the lemma below. By substituting these formulae as we go along the dag, we get a formula for the words accepted.

Lemma 3. – $\triangleright w \triangleleft, i \models At_e(x)$ *iff there exists a partial run of M (starting with $(s,1)$) which ends in configuration (p,i) and $(w[*,i,*] = \alpha)$.*
– $\triangleright w \triangleleft, i \models After_e(x)$ *iff there exists a partial run ending with last two configurations $(p,j)(q,i)$ where the last edge of the automaton taken is e.*

Construction. It is easy to construct $After_e(x)$ given $At_e(x)$. For edge $e = (p,\alpha,q,dir)$ we have $After_e(x) \overset{\text{def}}{=} \exists y.\, S(x,y) \wedge At_e(y)$ if $dir = L$; $After_e(x) \overset{\text{def}}{=} \exists y.\, S(y,x) \wedge At_e(y)$ if $dir = R$; and $After_e(x) \overset{\text{def}}{=} At_e(x)$ if $dir = X$.

Given α, there is a $FO^2[<,S]$ formula $\alpha(x)$ stating that the position x matches α. E.g. $da\underline{b}c(x) \overset{\text{def}}{=} b(x) \wedge (\exists y.\, S(y,x) \wedge a(y) \wedge (\exists x.\, S(x,y) \wedge d(x))) \ \wedge \ (\exists y.\, S(x,y) \wedge c(y))$.

Now we give the construction of $At_e(x)$, by induction on the depth of the edge. Consider an edge $e = (p,\alpha,q,dir)$ where the labels of other progress edges from state p are $\alpha_1,\ldots,\alpha_k$. Let the incoming progress edges to p be $e_1,\ldots,e_r$. We consider here the case that $\delta_{else}(p) = (p,R)$, i.e. a self-loop scanning rightwards. The case $\delta_{else}(p) = (p,L)$ is symmetric.

$$\begin{aligned} At_e(x) \overset{\text{def}}{=} \ & \alpha(x) \wedge \vee_{e_i \in \{e_1 \ldots e_r\}} [\\ & \quad (\exists y.\, y \leq x \wedge After_{e_i}(y)) \wedge \\ & \quad (\forall y.\, y < x \Rightarrow ((\neg\alpha(y) \wedge \neg\alpha_1(y) \wedge \ldots \wedge \neg\alpha_k(y)) \vee \\ & \qquad\qquad\qquad (\exists x.\, y < x \wedge After_{e_i}(x))))] \end{aligned}$$

For the start state s we assume that there is a dummy incoming edge e_{init} such that $After_{e_{init}}(x)$ is a formula which holds exactly at position 1 in w. The formula $\phi(M)$ for the whole automaton M is the disjunction of the formulae $At_{e_i}(x)$ for each incoming edge e_i to the accepting state t. Note that the size of $\phi(M)$ is exponential in size of M.

Theorem 3. *Every po2dla can be effectively reduced to a language equivalent formula of $FO^2[<,S]$ of exponential size.*

Hence, using Theorem 2, every LITL formula can be effectively reduced to a language equivalent $FO^2[<,S]$ formula, but a direct quadratic translation from LITL to $FO^2[<,S]$ generalizing the one in [Shah07] can also be worked out. In this paper, Theorem 5 will show that we can go from $FO^2[<,S]$ to LITL.

3 Games and Expressiveness

We now investigate the expressiveness of $FO^2[<,S]$ with respect to the dot-depth hierarchy. Since a successor predicate can be replaced by $<$ with an additional nesting of a quantifier, we get that $FO^2[<,S] \subseteq \Delta_3[<]$.

Theorem 4. *(i)* $\Pi_2[<] \nsubseteq FO^2[<,S]$
(ii) $\Sigma_2[<] \nsubseteq FO^2[<,S]$
(iii) $FO^2[<,S] \nsubseteq \mathcal{B}(\Sigma_2)[<]$

To prove the above results, we use Ehrenfeucht-Fraïssé games [Fra50, Ehr61]. The signature has unary predicates Q_a, Q_b, Q_c and $<$ and S are the binary predicates, with their usual definitions. Let Sig be a signature, and u, v be two word structures over Sig. An EF game $G(u,v,p,r)$ is a game played by 2 players, the *Spoiler* and *Duplicator*, over the word models u, v. A play of the game has r rounds with each player playing p pebbles. The pebbles are colored with p different colors, each player having exactly one pebble of each color.

In each round the *Spoiler* picks (any) one of the words, and places his p pebbles on it. The *Duplicator* then places his corresponding p pebbles on the other word. *Duplicator* wins the game if at the end of r rounds there exists a partial isomorphism between the pebble positions, with respect to all the relations of Sig. Note that this can only happen if each of the intermediate configurations is also a partial isomorphism. Weis and Immerman [WI07] proved the following version of the Ehrenfeucht-Fraïssé theorem.

Definition 3. *Two words u, v are said to be r-$FO^2[<,S]$* equivalent *if for any $FO^2[<,S]$ formula ϕ with quantifier depth $\leq r$, $u \models \phi \Leftrightarrow v \models \phi$, and p-$\mathcal{B}(\Sigma_2)[<]$* equivalent *if for any $\mathcal{B}(\Sigma_2)[<]$ formula ϕ with $\leq p$ variables, $u \models \phi \Leftrightarrow v \models \phi$.*

Lemma 4. *(a) Two word models u, v over the signature $[<,S]$ are r-$FO^2[<,S]$ equivalent iff for the game $G(u,v,2,r)$, the Duplicator always has a winning strategy.*
(b) Two word models u, v over the signature $[<]$ are p-$\mathcal{B}(\Sigma_2)[<]$ equivalent iff for the game $G(u,v,p,2)$, the Duplicator always has a winning strategy.

Proof (of Theorem 4). We note that since $FO^2[<,S]$ is a boolean closed logic, (i) of the theorem will imply (ii) (or vice versa). We consider words over the alphabet $A = \{a,b,c\}$ described by a conjunct $\phi_A = \forall x(Q_a(x) \vee Q_b(x) \vee Q_c(x))$.
(i) We consider the language $(ac^*bc^*)^*$. This language may be expressed by the conjuncts below giving a $\Pi_2[<]$ formula:
$\forall x(\forall y(y < x \Rightarrow x = y)) \Rightarrow Q_a(x)$
$\forall x \exists y(Q_b(y) \wedge (x > y \Rightarrow Q_c(x)))$
$\forall x \forall y((Q_a(x) \wedge Q_a(y) \wedge x < y) \Rightarrow (\exists z.(x < z < y \wedge Q_b(z))))$ and
$\forall x \forall y((Q_b(x) \wedge Q_b(y) \wedge x < y) \Rightarrow (\exists z.(x < z < y \wedge Q_a(z))))$

For some $r > 0$, consider two word models over the signature $[<,S]$:
$u : (ac^r bc^r)^{2r}$, and $v : (ac^r bc^r)^r bc^r (ac^r bc^r)^r$
Here, $u \in (ac^* bc^*)^*$ and $v \notin (ac^* bc^*)^*$. We can show that for any 2-pebble, r-round EF game $G(u,v,2,r)$, the *Duplicator* always has a winning strategy, and hence u, v are *r-*$FO^2[<,S]$ equivalent. This is evident from the observation that the two b's in v that do not have an a between them are separated by r c's and hence can be duplicated by the r^{th} bc^r in u. It is straightforward to see that any of the moves on a's or b's by the *Spoiler* can be duplicated in the other word. So the language $(ac^* bc^*)^*$ cannot be expressed in $FO^2[<,S]$.
(iii) We show that the language given by the LITL formula $(\neg(\top F_{bb} \top))F_{aa}\top$ is not definable in $\mathcal{B}(\Sigma_2)[<]$. Over the signature $[<]$, we first claim that no formula using less than p variables can distinguish in one round between the words $u_1 = (ab)^p bb (ab)^p aa (ab)^p$ and $v_1 = (ab)^p aa (ab)^p bb (ab)^p$. This is because any subsequence of length p in one word can be matched in the other word.

Now consider the pair of words $u_2 = u_1^p$ and $v_2 = v_1^p$ formed by taking p copies of the earlier ones. Now any placement of p pebbles in one word can be matched in the other word so that the subwords of length at most $p-2$ between any two pebbles (or a pebble and an end of the word) are the same. This means that Duplicator has a winning strategy for the second round as well.

Since for every p, the first word u_2 is not in the language given by $(\neg(\top F_{bb} \top))F_{aa}\top$ and the second word v_2 is in the language, this shows that any $\mathcal{B}(\Sigma_2)[<]$ formula (using, say, p variables) fails to define the language. □

3.1 Using Unambiguity on Rhodes-Straubing Expansions

We now show that the expressiveness of $FO^2[<,S]$ is no more than that of LITL. Since the proofs of the lemmas are refinements of those in [TW98], they are omitted here. Let RS_d be the set of all words obtained as Rhodes-Straubing d-expansions (see Definition 1) of words over A, i.e. let $RS_d = (A^*)^{\rho}_d$. Our use of it is reminiscent of the rôle of Dyck languages in CFLs.

Lemma 5. *If a language L is defined by an $FO^2[<,S]$ sentence with at most r quantifier alternations and upto d successor formulas, then its d-expansion L^{ρ}_d is the intersection of RS_d with a language definable by a sentence of $FO^2[<]$ with at most r quantifier alternations.*

The letters occurring in a word x (more generally, in a set of words) are called its content. We will use $\|x\|$ to denote the size of the content of x. If the letter a is in $\|x\|$, the left a-chop of x is vaw where $x = v_1 a v_2$ and a is not in the content of v_1 (not in ithe content of v_2, respectively, for a right a-chop).

Definition 4 (Thérien and Wilke). *Let $n \geq \|x,y\|$. Any two words x and y are said to be $n,0$-equivalent. Two words x and y are* $n,k+1$-equivalent *if they have the same content and for every letter a in the content, if their left a-chops are x_1ax_2, y_1ay_2 respectively, then x_1 and y_1 are $n-1,k+1$-equivalent and x_2 and y_2 are n,k-equivalent, as well as a symmetric condition for right chops. The n,k-*choppable *languages are those which* *are a union of n,k-equivalence classes. The* ***unambiguously choppable*** *languages are those which are n,k-choppable for some $n,k > 0$.*

The next result combines the earlier proposition with the result of Thérien and Wilke [TW98] that an $FO^2[<]$-definable language is unambiguously choppable.

Corollary 2. *If a language L is defined by an $FO^2[<,S]$ sentence with upto d successor formulas, then its d-expansion L^{ρ}_{d} is the intersection of RS_d with an unambiguously choppable language.*

We now traverse the path back to our logic LITL.

Theorem 5. *The languages defined by sentences of $FO^2[<,S]$ can be defined in LITL.*

Proof. From Corollary 2, we know that for an $FO^2[<,S]$-definable language L (using d successors) over the alphabet A, its Rhodes-Straubing d-expansion $L^{\rho} \subseteq RS_d$ is unambiguously choppable over the alphabet $(A')^{2d+1}$. We construct LITL formulae for $\{w \mid w^{\rho} \in C^{\rho}\}$ and for each n,k-equivalence class $C^{\rho} \subseteq L^{\rho}$, by induction on n and k. Taking the disjunction of the formulae for the finitely many equivalence classes saturating L^{ρ} gives an LITL formula for L.

For the base case, an $n,0$-equivalence class determines a content B of letters over $(A')^{2d+1}$. The language recognized by words which map to this equivalence class is B^*, defined by the intersection below. Although B^* is a language over $(A')^{2d+1}$, the LITL formula is over the alphabet A since existence of a letter $u\underline{a}v$ in w^{ρ} is equivalent to validating $w \models trueF_{u\underline{a}v}true$.

- For lookarounds $u\underline{a}v \in (A')^{2d+1} \setminus B$ without padding, the conjunct is $\neg(trueF_{u\underline{a}v}true)$.
- For $\triangleright^i au\underline{b}v \in (A')^{2d+1} \setminus B$ where $i > 0$ and $|au| + i = d$, the conjunct is $\neg(ptF_{\underline{a}ubv}true)$.
- For $u\underline{a}vb\triangleleft^i \in (A')^{2d+1} \setminus B$ where $i > 0$ and $|vb| + i = d$, the conjunct is $\neg(trueL_{uav\underline{b}}pt)$.

For the induction step, an $n,k+1$-equivalence class determines a content B as well as a set of left and right α-chops for $\alpha \in B$. The required formula for an $n,k+1$-equivalence class is given by the following intersection, where in both cases we go through the endmarker analysis above and shift position as required.

- Formulae $D_1F_{\alpha}D_2$ ($D'_1L_{\alpha}D'_2$, respectively) and all allowed left (right, resp.) α-chops, where the lookarounds α range over the content.
- Negations of such formulae for the α-chops in $(A')^{2d+1}$ which are not allowed.

The formulae D_1, D_2, D'_1, D'_2 for n,k-classes over content B and for $n-1,k+1$-classes over content $B \setminus \{\alpha\}$ are obtained from the induction hypothesis. Consider for instance that $x^{\rho} = v^{\rho}\alpha w^{\rho}$ is a left α-chop over a Rhodes-Straubing expansion. The induction hypothesis gives us $v \models D_1$ and $w \models D_2$ and so we have $x \models D_1F_{\alpha}D_2$ using α as a lookaround rather than a letter. □

Acknowledgements. We would like to thank the DLT referees for their detailed and encouraging reports and Sheng Yu, the PC co-chair, for gracefully allowing us to recover from some errors in the submitted version. KL would like to thank Manfred Kufleitner, Howard Straubing and Pascal Weil for discussions about the status of the logic $FO^2[<,S]$ within the dot-depth hierarchy during a visit in May-June '07 to LaBRI, Bordeaux, under the Indo-French research network project (P2R) Modiste-Cover/Timed-Discoveri. Pascal also introduced us to Rhodes expansions [Til76].

References

[DGK08] Diekert, V., Gastin, P., Kufleitner, M.: A survey on small fragments of first-order logic over finite words. Int. J. Found. Comp. Sci. 19(3), 513–548 (2008)

[Ehr61] Ehrenfeucht, A.: An application of games to the completeness problem for formalized theories. Fund. Math. 49, 129–141 (1961)

[EW00] Etessami, K., Wilke, T.: An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic. Inf. Comput. 160, 88–108 (2000)

[EVW02] Etessami, K., Vardi, M.Y., Wilke, T.: First-order logic with two variables and unary temporal logic. Inf. Comput. 179, 279–295 (2002)

[Fra50] Fraïssé, R.: Sur une nouvelle classification des systémes de relations. Compt. Rend. 230, 1022–1024 (1950)

[Imm98] Immerman, N.: Descriptive complexity. Springer, Heidelberg (1998)

[KW09] Kufleitner, M., Weil, P.: On FO^2 quantifier alternation over words. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 513–524. Springer, Heidelberg (2009)

[LPS08] Lodaya, K., Pandya, P.K., Shah, S.S.: Marking the chops: an unambiguous temporal logic. In: Ausiello, G., Karhumäki, J., Mauri, G., Ong, L. (eds.) Proc. 5th IFIP TCS, Milano. IFIP Series, vol. 273, pp. 461–476. Springer, Heidelberg (2008)

[LT00] Löding, C., Thomas, W.: Alternating automata and logics over infinite words. In: Watanabe, O., Hagiya, M., Ito, T., van Leeuwen, J., Mosses, P.D. (eds.) TCS 2000. LNCS, vol. 1872, pp. 521–535. Springer, Heidelberg (2000)

[Mor75] Mortimer, M.: On language with two variables. Zeit. Math. Log. Grund. Math. 21, 135–140 (1975)

[MW05] Muscholl, A., Walukiewicz, I.: An NP-complete fragment of LTL. Int. J. Found. Comp. Sci. 16(4), 743–754 (2005)

[Pin86] Pin, J.-E.: Varieties of formal language. North Oxford (1986)

[PW97] Pin, J.-E., Weil, P.: Polynomial closure and unambiguous products. Theory Comput. Syst. 30, 383–422 (1997)

[Sch76] Schützenberger, M.-P.: Sur le produit de concaténation non ambigu. Semigroup Forum 13, 47–75 (1976)

[STV02] Schwentick, T., Thérien, D., Vollmer, H.: Partially-ordered two-way automata: a new characterization of DA. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 239–250. Springer, Heidelberg (2002)

[Shah07] Shah, S.S.: FO^2 and related logics, Master's thesis TIFR (2007)

[SP09] Shah, S.S., Pandya, P.K.: An automaton normal form for UITL, Technical Report STCS-TR-SP-2009/1 (Computer Science Group, TIFR, 2009)

[Str85] Straubing, H.: Finite semigroup varieties of the form V*D. J. Pure Appl. Algebra 36(1), 53–94 (1985)

[TT02] Tesson, P., Thérien, D.: Diamonds are forever: the variety DA. In: Gomes, G.M.S., Silva, P.V., Pin, J.-E. (eds.) Semigroups, Algorithms, Automata and Languages, pp. 475–500. World Scientific, Singapore (2002)

[TW98] Thérien, D., Wilke, T.: Over words, two variables are as powerful as one quantifier alternation: $FO^2 = \sum_2 \cap \prod_2$. In: Proc. 30th STOC, Dallas, pp. 41–47 (1998)

[Tho82] Thomas, W.: Classifying regular events in symbolic logic. JCSS 25(3), 360–376 (1982)

[Til76] Tilson, B.: Complexity of semigroups and morphisms. In: Eilenberg, S. (ed.) Automata, Languages and Machines, ch. XII, vol. B. Academic Press, London (1976)

[WI07] Weis, P., Immerman, N.: Structure theorem and strict alternation hierarchy for FO^2 on words. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 343–357. Springer, Heidelberg (2007)

Input Products for Weighted Extended Top-Down Tree Transducers

Andreas Maletti*

Universitat Rovira i Virgili, Departament de Filologies Romàniques
Avinguda de Catalunya 35, 43002 Tarragona, Spain
andreas.maletti@urv.cat

Abstract. A weighted tree transformation is a function $\tau\colon T_\Sigma \times T_\Delta \to A$ where T_Σ and T_Δ are the sets of trees over the ranked alphabets Σ and Δ, respectively, and A is the domain of a semiring. The input and output product of τ with tree series $\varphi\colon T_\Sigma \to A$ and $\psi\colon T_\Delta \to A$ are the weighted tree transformations $\varphi \triangleleft \tau$ and $\tau \triangleright \psi$, respectively, which are defined by $(\varphi \triangleleft \tau)(t, u) = \varphi(t) \cdot \tau(t, u)$ and $(\tau \triangleright \psi)(t, u) = \tau(t, u) \cdot \psi(u)$ for every $t \in T_\Sigma$ and $u \in T_\Delta$. In this contribution, input and output products of weighted tree transformations computed by weighted extended top-down tree transducers (wxtt) with recognizable tree series are considered. The classical approach is presented and used to solve the simple cases. It is shown that input products can be computed in three successively more difficult scenarios: nondeleting wxtt, wxtt over idempotent semirings, and weighted top-down tree transducers over rings.

1 Introduction

Top-down tree series transducers [1,2,3] are a weighted version of top-down tree transducers [4,5,6,7,8]. Here we consider weighted extended top-down tree transducers (wxtt) [9,10,11], which are a generalization of top-down tree series transducers to allow several symbols in the left-hand side of rules. The framework TIBURON [12] implements wxtt over various weight semirings like the BOOLEAN semiring $(\{\bot, \top\}, \vee, \wedge, \bot, \top)$, the arctic semiring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$, and the probability semiring $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$, where elements outside the interval $[0, 1]$ are included because a semiring is closed under addition.

In this contribution, we will consider two standard operations for wxtt: input and output product. Given a wxtt M and a (suitable) recognizable tree series φ [13,14,15,16,3], their input product[1] should be a wxtt that computes the weight $(\varphi, t) \cdot \tau_M(t, u)$ for every input tree t and output tree u, where τ_M is the weighted tree transformation computed by M. In other words, the obtained wxtt shall scale the weight $\tau_M(t, u)$ assigned by M with the weight (φ, t) assigned by φ to the input tree. The output product is defined analogously. For

* Financially supported by *Ministerio de Educación y Ciencia* grants JDCI-2007-760 and MTM-2007-63422.

[1] Formally, the input product is a weighted tree transformation that might or might not be computable by another wxtt. In applications the former case is very desirable.

completeness' sake, we note that those products are partial; i.e., there exist wxtt and recognizable tree series whose product cannot be computed by a wxtt.

Input and output product have several applications. First, they offer the possibility to integrate a stand-alone parser into a wxtt M. We can encode the parser (e.g., the Collins parser [17]) as a recognizable tree series and then perform the input product. The obtained wxtt will multiply the parse weight of the input tree to the weight of each tree pair in M. Another common usage is restriction, in which we want to limit the input (or output) trees to be of a given (recognizable) shape. For every recognizable tree language L [18,19] we can obtain a recognizable tree series 1_L that assigns the weight 1 (neutral element of the multiplication) to each tree of L and weight 0 (neutral element of the addition) to each remaining tree. The input product with 1_L then restricts the weighted tree transformation to input trees of L. More precisely, the weight of any tree pair (t, u) will be 0 if $t \notin L$, whereas the weight of the tree pair remains $\tau_M(t, u)$ if $t \in L$. This particular use of the input product is also known as (generalized) Bar-Hillel construction, which originally restricts (or intersects) a context-free grammar with a regular language [20]. If L is a singleton, then the Bar-Hillel construction essentially yields a representation of the parses of the element of L. Consequently, the Bar-Hillel construction can be understood as a parser. This use is explained in detail in [21]. Finally, the input product is equivalent to recognizable look-ahead [8,11], so that devices with such look-ahead can be simulated by an input product. This can be used to prove that certain devices with recognizable look-ahead are as powerful as without. For example, Theorem 2 of Sect. 4 easily yields a generalization to the weighted case of [11, Theorem 4.4], which shows that nondeleting wxtt have recognizable look-ahead in the unweighted case (i.e., over the Boolean semiring).

The output product can easily be obtained with existing composition constructions [22,7,2,23]. The same applies to the input product if the wxtt is linear and nondeleting [2,11]. The main results of this paper concern nonlinear wxtt and we obtain input product constructions for:

– nondeleting wxtt over commutative semirings,
– some-copy nondeleting wxtt over idempotent commutative semirings, and
– some-copy nondeleting wxtt over commutative rings.

The first construction is rather standard. Some-copy nondeleting wxtt are defined such they fully explore at least one copy of each input subtree. In contrast, in a nondeleting wxtt every copy of an input subtree is fully explored. Whenever the wxtt copies a subtree, the second construction nondeterministically guesses a copy that fully explores the subtree. The idempotence of the semiring guarantees that the correct weight is obtained even if several copies are fully explored (i.e., several guesses are successful). The final construction for rings (note that no nontrivial idempotent commutative semiring is a commutative ring) is more involved because a scheme needs to be developed such that several successful explorations cancel each other out in a systematic way. The main problem is that it cannot be enforced with the state behavior alone that only one guess is successful.

2 Preliminaries

The set $\mathbb{N}$ is the set of nonnegative integers. We let $[k] = \{i \mid 1 \leq i \leq k\}$ for every $k \in \mathbb{N}$. Note that $[0] = \emptyset$. The set of all strings over a set Q is denoted by Q^*, of which the empty string is ε. The length $|w|$ of a string $w \in Q^*$ is the number of occurrences of symbols. The ith symbol in w is denoted by w_i. We use a fixed set $X = \{x_i \mid i \in \mathbb{N}\}$ of *formal variables* and its finite subsets $X_k = \{x_i \mid i \in [k]\}$ for every $k \in \mathbb{N}$. Consequently, $X_0 = \emptyset$.

A *ranked alphabet* (Σ, rk) is a finite set of symbols Σ together with a rank mapping $\mathrm{rk}\colon \Sigma \to \mathbb{N}$, which associates a rank to each symbol. We often just write Σ instead of (Σ, rk) and write Σ_k for the set of all symbols in Σ that have rank k. The set of *Σ-trees* indexed by a set V, which is denoted by $T_\Sigma(V)$, is the smallest set such that (i) $V \subseteq T_\Sigma(V)$ and (ii) $\sigma(t_1, \dots, t_k) \in T_\Sigma(V)$ for every $\sigma \in \Sigma_k$ and $t_1, \dots, t_k \in T_\Sigma(V)$. We write α for $\alpha()$ with $\alpha \in \Sigma_0$. In addition, we write T_Σ for $T_\Sigma(\emptyset)$. Let $t \in T_\Sigma(V)$ be a tree. The set $\mathrm{pos}(t)$ of *positions* (or *nodes*) in t is inductively defined by $\mathrm{pos}(v) = \{\varepsilon\}$ for every $v \in V$ and

$$\mathrm{pos}(\sigma(t_1, \dots, t_k)) = \{\varepsilon\} \cup \{iw \mid i \in [k], w \in \mathrm{pos}(t_i)\}$$

for every $\sigma \in \Sigma_k$ and $t_1, \dots, t_k \in T_\Sigma(V)$. We write $t(w)$ for the label of t at position w. Moreover, $\mathrm{pos}_\ell(t) = \{w \in \mathrm{pos}(t) \mid t(w) = \ell\}$ for every $\ell \in \Sigma \cup V$. The tree t is *linear* (respectively, *nondeleting*) *in* V if $|\mathrm{pos}_v(t)| \leq 1$ (respectively, $|\mathrm{pos}_v(t)| \geq 1$) for every $v \in V$. We write $C_\Sigma(X_k)$ for the subset of all trees of $T_\Sigma(X)$ that are linear and nondeleting in X_k.

Clearly, the positions $\mathrm{pos}(t) \subseteq \mathbb{N}^*$ are lexicographically ordered. Thus, the occurrences $\mathrm{pos}_v(t)$ of a variable $v \in V$ in t are also ordered. Given n pairwise distinct variables $v_1, \dots, v_n \in V$ and $t_{v_i 1}, \dots, t_{v_i m_i} \in T_\Sigma(V)$ for every $i \in [n]$ with $m_i = |\mathrm{pos}_{v_i}(t)|$, the substitution

$$t[v_1 \leftarrow (t_{v_1 1}, \dots, t_{v_1 m_1}), \dots, v_n \leftarrow (t_{v_n 1}, \dots, t_{v_n m_n})]$$

or just $t[v_i \leftarrow (t_{v_i 1}, \dots, t_{v_i m_i}) \mid i \in [n]]$ denotes the tree obtained by replacing, for every $i \in [n]$, the m_i occurrences of v_i in t by $(t_{v_i 1}, \dots, t_{v_i m_i})$ in order; i.e., the leftmost occurrence of v_i is replaced by $t_{v_i 1}$ and the rightmost occurrence is replaced by $t_{v_i m_i}$. If $t \in C_\Sigma(X_n)$ and $X_n = \{v_1, \dots, v_n\}$, then we just write $t[t_{x_1 1}, \dots, t_{x_n 1}]$ instead of the cumbersome $t[x_1 \leftarrow (t_{x_1 1}), \dots, x_n \leftarrow (t_{x_n 1})]$. Moreover, for every $t \in T_\Sigma$, let $\mathrm{match}(t)$ be the finite set

$$\mathrm{match}(t) = \{(l, t_1, \dots, t_k) \mid l \in C_\Sigma(X_k), t_1, \dots, t_k \in T_\Sigma, l[t_1, \dots, t_k] = t\} \ .$$

A *(commutative) semiring* $\mathcal{A} = (A, +, \cdot, 0, 1)$ is an algebraic structure such that $(A, +, 0)$ and $(A, \cdot, 1)$ are commutative monoids, $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ for every $a, b, c \in A$, and $a \cdot 0 = 0 = 0 \cdot a$ for every $a \in A$. It is *idempotent* if $1 + 1 = 1$. In an idempotent semiring $\mathcal{A}$ the *natural order* $\leq \, \subseteq A \times A$, which is given by $a \leq b$ if and only if $a + b = b$ for every $a, b \in A$, is a partial order for which the operations $+$ and $\cdot$ are monotone. Finally, the semiring $\mathcal{A}$ is a *ring* if there exists an element $(-1) \in A$ such that $1 + (-1) = 0$. For the rest of the paper, let $\mathcal{A} = (A, +, \cdot, 0, 1)$ be a commutative semiring.

Let Σ be a ranked alphabet. Every mapping $\varphi\colon T_\Sigma \to A$ is a *tree series*, which is also expressed by $\varphi \in A\langle\langle T_\Sigma\rangle\rangle$. For every $t \in T_\Sigma$ the value $\varphi(t)$ of t in φ is usually written as (φ, t). Next, we recall recognizable tree series [13,24,3]. A *weighted tree automaton* (wta) [24] is a system $N = (P, \Sigma, F, \mu)$ where (i) P is a finite set of *states*, (ii) Σ is a ranked alphabet of *input symbols*, (iii) $F\colon P \to A$ is a *final weight vector*, and (iv) $\mu = (\mu_k)_{k\in\mathbb{N}}$ is a family of *weighted transitions* with $\mu_k\colon P^k \times \Sigma_k \times P \to A$ for every $k \in \mathbb{N}$. We generalize our wta to work on trees of $T_\Sigma(X_n)$ with $n \in \mathbb{N}$. Note that $T_\Sigma = C_\Sigma(X_0) \subseteq T_\Sigma(X_n)$. For all $p_1, \ldots, p_n \in P$ we extend μ to a mapping $h_\mu^{p_1\cdots p_n}\colon T_\Sigma(X_n) \to A^P$ by

$$h_\mu^{p_1\cdots p_n}(x_i)_p = \begin{cases} 1 & \text{if } p = p_i \\ 0 & \text{otherwise} \end{cases}$$

$$h_\mu^{p_1\cdots p_n}(\sigma(t_1, \ldots, t_k))_p = \sum_{q_1,\ldots,q_k\in P} \mu_k(q_1\cdots q_k, \sigma, p) \cdot \prod_{i=1}^{k} h_\mu^{p_1\cdots p_n}(t_i)_{q_i}$$

for every $i \in [n]$, $\sigma \in \Sigma_k$, $t_1, \ldots, t_k \in T_\Sigma(X_n)$, and $p \in P$. The wta N *recognizes* the tree series $\|N\| \in A\langle\langle T_\Sigma\rangle\rangle$, which is given by $(\|N\|, t) = \sum_{p\in P} F(p) \cdot h_\mu(t)_p$ for every $t \in T_\Sigma$. Recall that $\mathcal{A}$ is commutative, so $F(p) \cdot h_\mu(t)_p = h_\mu(t)_p \cdot F(p)$. A tree series that is recognized by some wta is *recognizable.*

Next, we define our main tree transducer model: the weighted extended top-down tree transducer [9,10,11]. Our definition will be slightly non-standard, but the particular syntax will prove useful for our constructions. We assure the reader that our semantics will be equivalent to the existing definitions [9,10,11,25]. A *weighted extended top-down tree transducer* (wxtt) is a system $(Q, \Sigma, \Delta, I, R)$ where (i) Q is a finite set of *states*, (ii) Σ and Δ are ranked alphabets of *input* and *output symbols*, respectively, (iii) $I\colon Q \to A$ is an *initial weight vector*, and (iv) R is a finite set of *rules* of the form $(q, l) \xrightarrow{a} (w, r)$ with $q \in Q$, $l \in C_\Sigma(X_k)$, $a \in A$, $w \in (Q^*)^k$, and $r \in T_\Delta(X_k)$ such that $|w_i| = |\mathrm{pos}_{x_i}(r)|$ for every $i \in [k]$. Intuitively speaking, a rule $(q, l) \xrightarrow{a} (w, r)$ consists of a *state* q, a *left-hand side* l, a *weight* $a \in A$, a *control word* w, and a *right-hand side* r. The control word consists of k words $w_1, \ldots, w_k$ of states. For each $i \in [k]$, the ith word records the states (in order) that are associated to the occurrences (in lexicographic order) of the variable x_i in r. A more classical rule shape, which we use in graphical representations, assumes that the states have rank 1 and presents the rule $(q, l) \xrightarrow{a} (w, r)$ as $q(l) \xrightarrow{a} r[x_i \leftarrow ((w_i)_1(x_i), \ldots, (w_i)_{|w_i|}(x_i)) \mid 1 \le i \le |w|]$. An example is displayed in Fig. 1 (left). The rule $(q, l) \xrightarrow{a} (w, r) \in R$ is *linear* (respectively, *nondeleting*) if $|w_i| \le 1$ (respectively, $|w_i| \ge 1$) for every $1 \le i \le |w|$. The wxtt M is *linear* (respectively, *nondeleting*) if every rule $\rho \in R$ is so. It is a *weighted top-down tree transducer* (wtt) if for every $(q, l) \xrightarrow{a} (w, r) \in R$ there exists $\sigma \in \Sigma_k$ such that $l = \sigma(x_1, \ldots, x_k)$. The example rule of Fig. 1 (left) is nondeleting, but not linear. Any wxtt with that rule is not a wtt.

In the following, let $M = (Q, \Sigma, \Delta, I, R)$ be a wxtt. To simplify the development, we assume henceforth that $l \neq x_1$ for every rule $(q, l) \xrightarrow{a} (w, r) \in R$. In

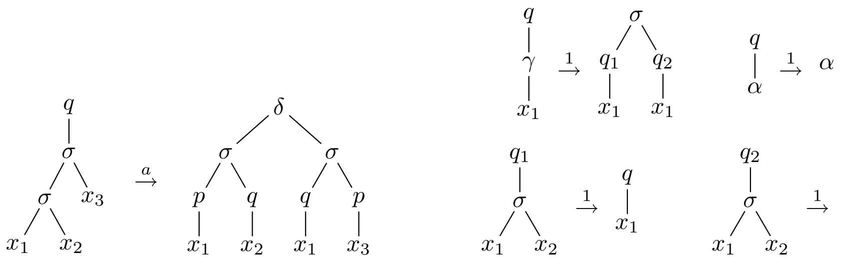


Fig. 1. Left: Graphical representation of the example rule $(q, l) \xrightarrow{a} ((pq, q, p), r)$ where $l = \sigma(\sigma(x_1, x_2), x_3)$ and $r = \delta(\sigma(x_1, x_2), \sigma(x_1, x_3))$. Right: Example of deleting rules.

other words, we disallow *ε-rules*.[2] We use the symbol q with and without additional subscripts for elements of Q. Consequently, if we write $w = q_1 \cdots q_n$, then implicitly $q_i \in Q$ for every $i \in [n]$. The semantics of the wxtt M (without epsilon rules) is defined as follows: Let $h_R \colon T_\Sigma \times T_\Delta \to A^Q$ be the mapping defined for every $t \in T_\Sigma$, $u \in T_\Delta$, and $q \in Q$ by

$$h_R(t, u)_q = \sum_{\substack{(q,l) \xrightarrow{a} (q_{11} \cdots q_{1n_1}, \dots, q_{k1} \cdots q_{kn_k}, r) \in R \\ (l, t_1, \dots, t_k) \in \text{match}(t), \forall i \in [k], \forall j \in [n_i] \colon u_{ij} \in T_\Delta \\ u = r[x_i \leftarrow (u_{i1}, \dots, u_{in_i}) \mid i \in [k]]}} a \cdot \prod_{\substack{i \in [k] \\ j \in [n_i]}} h_R(t_i, u_{ij})_{q_{ij}} \enspace .$$

The wxtt M *computes* the tree transformation $\tau_M \colon T_\Sigma \times T_\Delta \to A$, which is given by $\tau_M(t, u) = \sum_{q \in Q} I(q) \cdot h_R(t, u)_q$ for every $t \in T_\Sigma$ and $u \in T_\Delta$.

3 Input and Output Product

In this section, we formally define input and output product and then discuss the standard approach to solve the associated algorithmic problems. Let $\tau \colon T_\Sigma \times T_\Delta \to A$, $\varphi \in A\langle\langle T_\Sigma \rangle\rangle$, and $\psi \in A\langle\langle T_\Delta \rangle\rangle$. The *input product* $\varphi \triangleleft \tau$ of τ with φ and the *output product* $\tau \triangleright \psi$ of τ with ψ are $(\varphi \triangleleft \tau)(t, u) = (\varphi, t) \cdot \tau(t, u)$ and $(\tau \triangleright \psi)(t, u) = \tau(t, u) \cdot (\psi, u)$ for every $t \in T_\Sigma$ and $u \in T_\Delta$.

Classical solutions to the problems of input and output products are specialized Bar-Hillel constructions [20,21] or compositions [22,7]. The composition approach first embeds a tree series $\varphi \in A\langle\langle T_\Sigma \rangle\rangle$ into the identity mapping $\mathrm{id}_\varphi \colon T_\Sigma \times T_\Sigma \to A$, which is defined by $\mathrm{id}_\varphi(t, t') = (\varphi, t)$ if $t = t'$ and 0 otherwise for every $t, t' \in T_\Sigma$. Given two tree transformations $\tau_1 \colon T_\Sigma \times T_\Delta \to A$ and $\tau_2 \colon T_\Delta \times T_\Gamma \to A$, the *composition* $\tau_1 \,;\, \tau_2 \colon T_\Sigma \times T_\Gamma \to A$ of τ_1 and τ_2 is given by $(\tau_1 \,;\, \tau_2)(t, v) = \sum_{u \in T_\Delta} \tau_1(t, u) \cdot \tau_2(u, v)$ for every $t \in T_\Sigma$ and $v \in T_\Gamma$. Note that, in general, the sum in the definition of $\tau_1 \,;\, \tau_2$ may be infinite, but in our compositions $\tau_1 \,;\, \tau_2$ one of the tree transformations τ_1 or τ_2 will always be an

[2] Aside from well-definedness issues, there is no additional complexity with ε-rules. All our constructions can easily be adapted to work for general wxtt with well-defined semantics, but we would like to avoid an in-depth discussion of well-definedness.

identity mapping id_φ. This yields that the sum in the definition of the composition $\tau_1 \mathbin{;} \tau_2$ essentially degenerates into a single summand. Thus, we can express input and output product as $\varphi \triangleleft \tau = \mathrm{id}_\varphi \mathbin{;} \tau$ and $\tau \triangleright \psi = \tau \mathbin{;} \mathrm{id}_\psi$, respectively.

In this contribution, we consider input and output products of tree transformations that are computed by wxtt with recognizable tree series. The identity mappings id_φ and id_ψ can be computed by linear and nondeleting wtt for all recognizable tree series $\varphi \in A\langle\langle T_\Sigma\rangle\rangle$ and $\psi \in A\langle\langle T_\Delta\rangle\rangle$. Thus, the main question with regard to the composition approach is whether the given compositions for input and output product can be computed by another wxtt.

First, let us discuss the question for an unweighted (i.e., BOOLEAN weighted[3]) wtt M [5,4]. By the composition results of [22,7], the composition approach works for: (i) the output product and (ii) the input product if M is linear and nondeleting. Further results can be obtained for special recognizable tree series such as deterministic top-down recognizable tree series [26,18], but we will focus on general recognizable tree series in this contribution. The two mentioned results generalize to wtt over commutative semirings [2,23] and can easily be extended to wxtt as well. If we were to discuss input and output product also for weighted extended bottom-up tree transducers [9,2,27], then the roles would essentially exchange. The input product would be easy with the help of the composition approach and output products could be achieved following our approaches for input products of wxtt.

4 Nondeleting Transducers

From now on, let $M = (Q, \Sigma, \Delta, I, R)$ be a wxtt and $N = (P, \Sigma, F, \mu)$ be a wta. The aim of this and the following sections is to present constructions of M' such that $\tau_{M'} = \|N\| \triangleleft \tau_M$. This problem is simple if M is nondeleting. Each input subtree is visited at least once by M due to nondeletion, and we can arbitrarily select one call[4] to perform the input product. We select the first call here. As a notational convenience, we sometimes use angled brackets '$\langle$' and '$\rangle$' instead of parentheses '(' and ')'.

Definition 1. *The input product $N \triangleleft_{\mathrm{n}} M$ is the wxtt $(Q', \Sigma, \Delta, I', R \cup R')$ where*

- *$Q' = Q \cup (Q \times P)$,*
- *$I'(q) = 0$ and $I'(\langle q, p\rangle) = I(q) \cdot F(p)$ for every $q \in Q$ and $p \in P$, and*
- *$(\langle q, p\rangle, l) \xrightarrow{a \cdot a'} (\langle q_{11}, p_1\rangle q_{12} \cdots q_{1n_1}, \ldots, \langle q_{k1}, p_k\rangle q_{k2} \cdots q_{kn_k}, r) \in R'$ for every nondeleting rule $(q, l) \xrightarrow{a} (q_{11} \cdots q_{1n_1}, \ldots, q_{k1} \cdots q_{kn_k}, r) \in R$ and all states $p, p_1, \ldots, p_k \in P$ where $a' = h_\mu^{p_1 \cdots p_k}(l)_p$.*

Mind that in a nondeleting wxtt all rules are nondeleting, which allows us to arbitrarily select any call because each call will fully explore its subtree.

[3] The BOOLEAN semiring is $\mathbb{B} = (\{\bot, \top\}, \vee, \wedge, \bot, \top)$.

[4] In a rule $(q, l) \xrightarrow{a} (w, r) \in R$ any $(w_i)_j \in Q$ with $1 \le i \le |w|$ and $1 \le j \le |w_i|$ is also called '*call*' to the subtree represented by x_i. If $|w_i| \ge 2$, then there are several calls to the same subtree.

Theorem 2. *If M is nondeleting, then $\tau_{(N \triangleleft_{\mathrm{n}} M)} = \|N\| \triangleleft \tau_M$.*

Our construction can easily be generalized to settings, where based on the rule shape, the call that fully explores its subtree can be infered. The next sections will deal with cases, in which such a prediction is not possible.

5 Idempotent Semirings

In this and the next section, we develop successively more complicated input-product constructions for deleting wxtt. Clearly, if M deletes a particular input subtree in all copies, then we cannot obtain a general input-product construction because the deleted subtree might be essential for the wta N. Consequently, we restrict our attention to wxtt that visit each input subtree at least once. In principle, algorithms for the input product can be imagined in this setting, but such an algorithm would require extensive book-keeping. This is due to the fact that we would need to record which subtree is visited by which call, which seems unfeasible in a general-purpose construction. Nevertheless such a construction should eventually be established for the particular wxtt that are the result of binarizing nondeleting wxtt.

Let us consider the rules of Fig. 1 (right). Each subtree in the tree $\gamma(\sigma(\alpha, \alpha))$ is visited at least once starting with state q. The first rule creates two copies of the input subtree $\sigma(\alpha, \alpha)$, of which the first call explores the node labeled 'σ' and the left subtree α (and deletes the right subtree α). The second call explores the 'σ'-node and the right subtree α. Consequently, there is no single call that explores the full subtree $\sigma(\alpha, \alpha)$, which yields that the product construction needs to keep track which parts are explored by which call. To avoid this additional book-keeping, we introduce *some-copy nondeletion*, which demands that for each input subtree there is at least one call that fully explores the whole subtree.

Definition 3. *A state $q \in Q$ is t-*nondeleting *with $t \in T_\Sigma$ if for every rule $(q, l) \xrightarrow{a} (w, r) \in R$, $(l, t_1, \ldots, t_k) \in \mathrm{match}(t)$, and $1 \leq i \leq |w|$ there exists $1 \leq j \leq |w_i|$ such that $(w_i)_j$ is t_i-non-deleting. The wxtt M is* some-copy nondeleting *if q is t-nondeleting for every $t \in T_\Sigma$ and $q \in Q$ such that $I(q) \neq 0$.*

Note that any rule fulfilling the premise in Definition 3 must be nondeleting, however, not all rules R need to be nondeleting for M to be some-copy nondeleting. Classical nondeletion might be called "all-copies nondeletion" in analogy. We further note that some-copy nondeletion is a semantic property. It is decidable (because for every state $q \in Q$ the set of all trees $t \in T_\Sigma$ such that q is t-nondeleting is a recognizable tree language [18,19]), but it might not be a practical property. However, it encompasses several interesting forms of "nondeletion" (such as the ones mentioned at the end of the previous section), and thus allows us to present one construction (and its proof of correctness) for several "nondeletion" properties. We already argued that the state q with the rules of Fig. 1 (right) is not $\gamma(\sigma(\alpha, \alpha))$-nondeleting because neither q_1 nor q_2 are $\sigma(\alpha, \alpha)$-nondeleting. However, the wxtt with the rules of Fig. 2 is some-copy

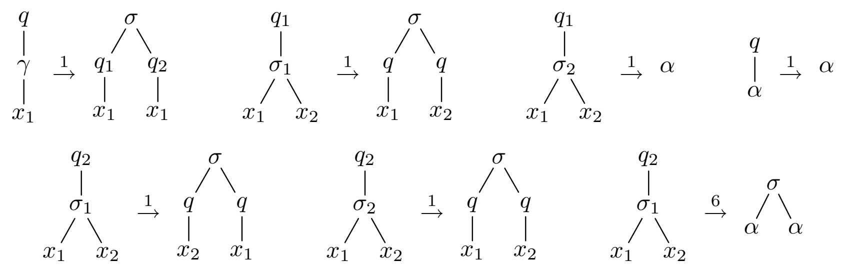


Fig. 2. Rules of a some-copy nondeleting wxtt

nondeleting if q is the only initial state. Note that the call that fully explores a subtree here depends on the input tree. For example, the first call created in the first rule fully explores the subtree $\sigma_1(\alpha, \alpha)$ in the tree $\gamma(\sigma_1(\alpha, \alpha))$, whereas the second call fully explores $\sigma_2(\alpha, \alpha)$ in the tree $\gamma(\sigma_2(\alpha, \alpha))$. The example also demonstrates that it is not sufficient that one (non-deterministic) choice fully explores a subtree. Although the state q_2 can fully explore the subtree $\sigma_1(\alpha, \alpha)$, it is not $\sigma_1(\alpha, \alpha)$-nondeleting because of the last rule of Fig. 2.

Since we cannot infer the correct call that fully explores a subtree from the information present in a rule, we have to guess it. If M is some-copy nondeleting, then we know that at least one such guess must be correct. However, it might happen that several calls eventually fully explore the same subtree. The author is unaware of any method to exclude this behavior with the help of the states alone. Consequently, we only prove that our construction is correct in idempotent semirings $\mathcal{A}$. The idempotency yields that several equivalent (or even partial) explorations can be performed without effect on the weight.

Definition 4. *The input product $N \triangleleft_{\mathrm{i}} M$ is the wxtt $(Q', \Sigma, \Delta, I', R \cup R')$ where*

- $Q' = Q \cup (Q \times P)$,
- $I'(q) = 0$ *and* $I'(\langle q, p\rangle) = I(q) \cdot F(p)$ *for every* $q \in Q$ *and* $p \in P$, *and*
- $(\langle q, p\rangle, l) \xrightarrow{a \cdot a'} (q_{11} \cdots \langle q_{1j_1}, p_1\rangle \cdots q_{1n_1}, \dots, q_{k1} \cdots \langle q_{kj_k}, p_k\rangle \cdots q_{kn_k}, r) \in R'$ *for every nondeleting rule* $(q, l) \xrightarrow{a} (q_{11} \cdots q_{1n_1}, \dots, q_{k1} \cdots q_{kn_k}, r) \in R$, *all states* $p, p_1, \dots, p_k \in P$, *and* $j_1 \in [n_1], \dots, j_k \in [n_k]$ *where* $a' = h_\mu^{p_1 \cdots p_k}(l)_p$.

Note that only one state of each part in the control word is replaced. Let us illustrate the construction on the rules of Fig. 2.

Example 5. Suppose that $\mathcal{A} = (\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ is the arctic semiring, which is idempotent. Moreover, suppose that M is given by the rules of Fig. 2 and that N is the trivial wta with $P = \{p\}$ such that $(\|N\|, t) = 0$ for every $t \in T_\Sigma$. Clearly, $\|N\| \triangleleft \tau_M = \tau_M$, so we would not need a construction at all, but let us consider the input tree $t = \sigma_1(\alpha, \alpha)$ and the output tree $u = \sigma(\alpha, \alpha)$. A simple calculation shows that $h_R(t, u)_{q_1} = 3$ and $h_R(t, u)_{q_2} = \max(3, 6)$. Now, let $N \triangleleft_{\mathrm{i}} M = (Q', \Sigma, \Delta, I', R')$. Then $h_{R'}(t, u)_{\langle q_1, p\rangle} = 3$ and $h_{R'}(t, u)_{\langle q_2, p\rangle} = 3$, which shows that the derivation with weight 6 is not possible in R' with a paired state because the last rule (which resulted in the weight 6) of Fig. 2 is deleting and thus no new rules (with a paired state) are created for it.

Theorem 6. *If M is some-copy nondeleting and $\mathcal{A}$ an idempotent semiring, then $\tau_{(N \triangleleft_{\mathrm{i}} M)} = \|N\| \triangleleft \tau_M$.*

6 Rings

In this final section, we will discuss the input product for wxtt over rings. From here on, we assume that $\mathcal{A}$ is a ring. To keep the presentation simple, we make the additional restriction that M is a wtt. This will make the discussion of the t-nondeletion property easier. We note that the restriction is done only for convenience. More elaborate versions of our construction could easily overcome the restriction. In the previous section we could easily allow spurious derivations because idempotence would "hide" the wrong weights corresponding to those spurious derivations. This is no longer possible in rings because no non-trivial (i.e., $0 \neq 1$) ring is idempotent. Consequently, our final construction needs to take care of those spurious derivations. To this end, we use a simple *elimination pattern*. Let $Q \times C$ be such that $(Q \times C) \cap Q = \emptyset$ for some set C. For every set Q' such that $Q \subseteq Q'$ and $w \in (Q')^*$, we let $|w|_Q = |\{i \mid w_i \in Q\}|$ be the number of Q-symbols in w. Our elimination pattern $f\colon (Q')^* \to A$ is given by $f(w) = 1$ if $|w| - |w|_Q$ is odd, and $f(w) = -1$ otherwise. Intuitively speaking, an elimination mapping is a strategy such that the derivations properly cancel to just one derivation irrespective of the set of calls that fully explore a certain subtree. This set of calls is non-empty because we will again assume some-copy nondeletion. Let us illustrate this for $Q' = Q \cup (Q \times C)$. For every $w \in (Q')^*$ let $\mathrm{base}(w)$ be the unique word such that $\mathrm{base}(\varepsilon) = \varepsilon$ and

$$\mathrm{base}(q'w') = \begin{cases} q' \,\mathrm{base}(w') & \text{if } q' \in Q \\ q \,\mathrm{base}(w') & \text{if } q' = \langle q, c \rangle \end{cases}$$

for every $q' \in Q'$ and $w' \in (Q')^*$. Let $q \in Q$ be an arbitrary state and $c \in C$. Given a nonempty set $J \subseteq [n]$ with $m = |J|$, which represents the calls that fully explore a given subtree, $w \in Q^n$, and $c_1, \dots, c_n \in \mathcal{C}$ we have

$$\sum_{\substack{w' \in (Q')^* \setminus Q^* \\ \mathrm{base}(w') = w \\ \forall i \in [n]\colon w'_i = \langle w_i, c_i \rangle \text{ if } w'_i \notin Q \\ \forall i \in [n] \setminus J\colon w'_i \in Q}} f(w') = \sum_{w' \in \{q, \langle q, c \rangle\}^m \setminus \{q\}^m} f(w') = 1 \enspace ,$$

where the last step is a trivial consequence of Pascal's triangle. Consequently, for any choice of calls that fully explore a certain subtree, our elimination strategy ensures that all but one cancel.

Before we proceed, let us consider the negation of t-nondeletion for wtt. Let $t = \sigma(t_1, \dots, t_k)$ for some $\sigma \in \Sigma_k$ and $t_1, \dots, t_k \in T_\Sigma$. A state $q \in Q$ is *t-deleting* if there exist a rule $(q, \sigma(x_1, \dots, x_k)) \xrightarrow{a} (w, r) \in R$ and $i \in [k]$ such that $(w_i)_j$ is

t_i-deleting for every $1 \leq j \leq |w_i|$. Consequently, a state q is $\sigma(t_1, \ldots, t_k)$-deleting for one of two reasons: (i) It has a deleting rule $(q, \sigma(x_1, \ldots, x_k)) \xrightarrow{a} (w, r) \in R$, or (ii) it has a nondeleting rule $(q, \sigma(x_1, \ldots, x_k)) \xrightarrow{a} (w, r) \in R$ such that all states called on a subtree t_i are t_i-deleting. In the first case, the subtrees $t_1, \ldots, t_k$ are obviously irrelevant. This characterization yields a simple check of t-deletion inside a wtt, which will be coded in the next construction. We need some additional notation. Let $\mathcal{C} = \mathcal{P}(\mathcal{P}(Q))$ where $\mathcal{P}(S)$ denotes the powerset of S. Moreover, for every $\sigma \in \Sigma_k$, $i \in [k]$, $S \subseteq Q$, $C \in \mathcal{C}$, and $q \in Q$, let

$$\begin{aligned}
\text{next}_i(\sigma, q) &= \{\{(w_i)_1, \ldots, (w_i)_{|w_i|}\} \mid (q, \sigma(x_1, \ldots, x_k)) \xrightarrow{a} (w, r) \in R\} \\
\text{next}_i(\sigma, S) &= \{\textstyle\bigcup_{q \in S} w_q \mid \forall q \in S \colon w_q \in \text{next}_i(\sigma, q)\} \\
\text{next}_i(q, \sigma, C) &= \bigcup_{S \in C} \text{next}_i(\sigma, S) \cup \text{next}_i(\sigma, q) \enspace .
\end{aligned}$$

Definition 7. *The input product $N \lhd_{\mathrm{r}} M$ is the wtt $(Q', \Sigma, \Delta, I', R'')$ where*

- *$Q' = Q \cup (Q \times \mathcal{C}) \cup (Q \times \mathcal{C} \times P)$ and $R'' = R \cup \overline{R} \cup R'$,*
- *for every $q \in Q$, $C \in \mathcal{C}$, and $p \in P$, let $I'(q) = I'(\langle q, C\rangle) = 0$ and*

$$I'(\langle q, C, p\rangle) = \begin{cases} I(q) \cdot F(p) & \textit{if } C = \{\{q\}\} \\ 0 & \textit{otherwise,} \end{cases}$$

- *for every $q \in Q$, $C \in \mathcal{C}$ such that $\emptyset \notin C$, $\sigma \in \Sigma_k$, and every nondeleting rule $(q, \sigma(x_1, \ldots, x_k)) \xrightarrow{a} (w, r) \in R$, let*

$$(\langle q, C\rangle, \sigma(x_1, \ldots, x_k)) \xrightarrow{a \cdot f(w'_1) \cdot \ldots \cdot f(w'_k)} (w'_1 \cdots w'_k, r) \in \overline{R}$$

for every $w'_1, \ldots, w'_k \in (Q \cup (Q \times C))^* \setminus Q^*$ such that for every $i \in [k]$

$$w'_i = \begin{cases} w_i & \textit{if } w'_i \in Q \\ (w_i, \text{next}_i(q, \sigma, C)) & \textit{otherwise,} \end{cases}$$

- *for every rule $(\langle q, C\rangle, \sigma(x_1, \ldots, x_k)) \xrightarrow{a} (w, r) \in \overline{R}$ and $p, p_1, \ldots, p_k \in P$ let $a' = h_\mu^{p_1 \cdots p_k}(l)_p$ and*

$$(\langle q, C, p\rangle, \sigma(x_1, \ldots, x_k)) \xrightarrow{a \cdot a'} (w'_1 \cdots w'_k, r) \in R' \enspace ,$$

where, for every $i \in [k]$, the control word w'_i is obtained from w_i by replacing the first paired state $\langle q', C'\rangle$, by $\langle q', C', p_i\rangle$.

In other words, the paired states guess the calls that fully explore their subtrees. Note that if there exists any deleting rule $(q, l) \xrightarrow{a} (w, r) \in R$ with $l = \sigma(x_1, \ldots, x_k)$ and $w_i = \varepsilon$ for some $i \in [k]$, then for every $(\langle q, C\rangle, l) \xrightarrow{e} (w', r')$ in $\overline{R}$ there exists $1 \leq j \leq |w'_i|$ such that $(w'_i)_j = \langle q', C'\rangle$ or $(w'_i)_j = \langle q', C', p\rangle$ with $\emptyset \in C'$. This is due to the fact that $\emptyset \in \text{next}_i(q, \sigma, C)$ because $\emptyset \in \text{next}_i(\sigma, q)$.

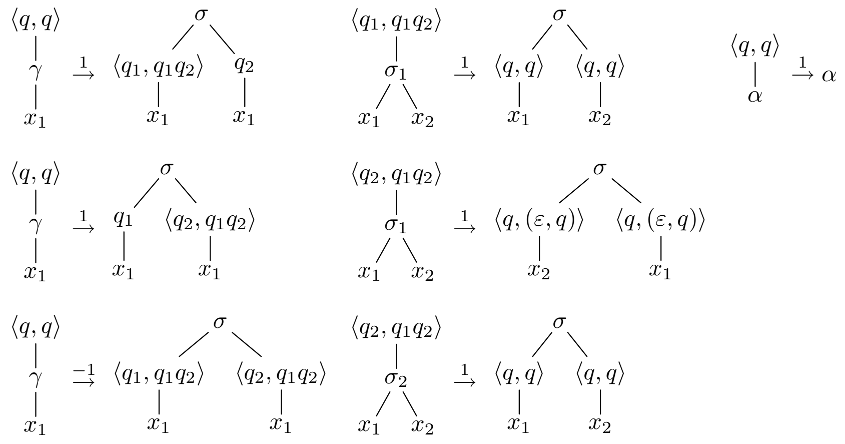


Fig. 3. Some constructed rules of $\overline{R}$ where R is the set of rules of Fig. 2. We use a string notation for the elements of $\mathcal{C}$ here.

A similar statement holds for R'. In essence, this blocks the computations, in which the guess was wrong because a deletion would be possible. It is not quite obvious that (aside from one) the various correct guesses cancel each other out. Let us illustrate the construction on the example rules of Fig. 2.

Example 8. Suppose that $\mathcal{A} = (\mathbb{Z}, +, \cdot, 0, 1)$ is the ring of integers, and let M be given by the rules of Fig. 2. To keep the example short, Fig. 3 only presents some relevant rules of $\overline{R}$ that are constructed in the third item of Definition 7. For $t = \gamma(\sigma_1(\alpha, \alpha))$ and $u = \sigma(\sigma(\alpha, \alpha), \sigma(\alpha, \alpha))$ we can compute that

$$h_R(t, u)_q = 1 + 6 = h_{R \cup \overline{R}}(t, u)_{\langle q, \{\{q\}\}\rangle} \enspace .$$

Now, let S be the set R without the rule with weight 6, and let $\overline{S}$ be the corresponding rule set constructed in the third item of Definition 7. Then

$$h_S(t, u)_q = 1 = 1 + 1 - 1 = h_{S \cup \overline{S}}(t, u)_{\langle q, \{\{q\}\}\rangle} \enspace .$$

Theorem 9. *If M is a some-copy nondeleting wtt and $\mathcal{A}$ is a ring, then*

$$\tau_{(N \lhd_{\mathrm{r}} M)} = \|N\| \lhd \tau_M \enspace .$$

References

1. Kuich, W.: Tree transducers and formal tree series. Acta Cybernet. 14(1), 135–149 (1999)
2. Engelfriet, J., Fülöp, Z., Vogler, H.: Bottom-up and top-down tree series transformations. J. Autom. Lang. Combin. 7(1), 11–70 (2002)
3. Fülöp, Z., Vogler, H.: Weighted tree automata and tree transducers. In: Handbook of Weighted Automata, pp. 313–403. Springer, Heidelberg (2009)

4. Rounds, W.C.: Mappings and grammars on trees. Math. Systems Theory 4(3), 257–287 (1970)
5. Thatcher, J.W.: Generalized^2 sequential machine maps. J. Comput. System Sci. 4(4), 339–367 (1970)
6. Thatcher, J.W.: Tree automata: An informal survey. In: Currents in the Theory of Computing, pp. 143–172. Prentice Hall, Englewood Cliffs (1973)
7. Engelfriet, J.: Bottom-up and top-down tree transformations: A comparison. Math. Systems Theory 9(3), 198–231 (1975)
8. Engelfriet, J.: Top-down tree transducers with regular look-ahead. Math. Systems Theory 10(1), 289–303 (1977)
9. Arnold, A., Dauchet, M.: Morphismes et bimorphismes d'arbres. Theoret. Comput. Sci. 20(1), 33–93 (1982)
10. Graehl, J., Knight, K., May, J.: Training tree transducers. Computational Linguistics 34(3), 391–427 (2008)
11. Maletti, A., Graehl, J., Hopkins, M., Knight, K.: The power of extended top-down tree transducers. SIAM J. Comput. 39(2), 410–430 (2009)
12. May, J., Knight, K.: Tiburon: A weighted tree automata toolkit. In: Ibarra, O.H., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 102–113. Springer, Heidelberg (2006)
13. Berstel, J., Reutenauer, C.: Recognizable formal power series on trees. Theoret. Comput. Sci. 18(2), 115–148 (1982)
14. Bozapalidis, S.: Equational elements in additive algebras. Theory Comput. Systems 32(1), 1–33 (1999)
15. Kuich, W.: Formal power series over trees. In: DLT, pp. 61–101. Aristotle University of Thessaloniki (1998)
16. Borchardt, B., Vogler, H.: Determinization of finite state weighted tree automata. J. Autom. Lang. Combin. 8(3), 417–463 (2003)
17. Collins, M.: Head-Driven Statistical Models for Natural Language Parsing. PhD thesis, University of Pennsylvania (1999)
18. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)
19. Gécseg, F., Steinby, M.: Tree languages. In: Handbook of Formal Languages, vol. 3, pp. 1–68. Springer, Heidelberg (1997)
20. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. In: Language and Information: Selected Essays on their Theory and Application, pp. 116–150. Addison Wesley, Reading (1964)
21. Satta, G.: Translation algorithms by means of language intersection (2010) (manuscript), www.dei.unipd.it/~satta/publ/paper/inters.pdf
22. Baker, B.S.: Composition of top-down and bottom-up tree transductions. Inform. and Control 41(2), 186–213 (1979)
23. Maletti, A.: Compositions of tree series transformations. Theoret. Comput. Sci. 366(3), 248–271 (2006)
24. Borchardt, B.: The Theory of Recognizable Tree Series. PhD thesis, Technische Universität Dresden (2005)
25. Maletti, A.: Why synchronous tree substitution grammars? In: NAACL, Association for Computational Linguistics, pp. 876–884 (2010)
26. Virágh, J.: Deterministic ascending tree automata I. Acta Cybernet. 5(1), 33–42 (1980)
27. Engelfriet, J., Lilin, E., Maletti, A.: Composition and decomposition of extended multi bottom-up tree transducers. Acta Inform. 46(8), 561–590 (2009)

Regular Hedge Language Factorization Revisited

Mircea Marin[1,⋆] and Temur Kutsia[2,⋆⋆]

[1] Department of Computer Science, University of Tsukuba, Japan
[2] RISC, Johannes Kepler University, Linz, Austria

Abstract. We consider the factorization problem of regular hedge languages. This problem is strongly related to the type checking problem in rule based transformations of valid XML documents. We propose the representation of regular hedge languages by reduced, complete, and deterministic linear hedge automata, and indicate algorithms for the computation of right factors and factor matrix.

1 Motivation

Regular hedge languages, or simply RHLs, are a natural generalization of regular languages where strings are replaced by sequences of unranked trees, also known as hedges. They were first studied by Thatcher [8,9], who developed the basic theory of unranked tree automata and investigated their regular extensions. Interest in their study was reignited by the advent of XML as the de facto standard for the exchange and manipulation of data on the Web [1], and by the recognition of the fact that RHLs are a suitable formalism to specify restrictions on the structure of XML documents. Hedge automata [7] were invented to type check (or validate) input data against specifications of RHLs, and regular expression types were introduced in XML processing languages [4] as a means to specify membership constraints to RHLs.

Several results from the theory of regular languages carry over nicely to regular hedge languages. In particular, the factorization theory of regular languages [2] has a natural generalization to regular hedge languages [6,5]. One of the motivations behind the study of regular hedge language factorizations is the type checking problem that arises in the context of XML transformation. Assume we are given a transformation rule $P \to r$ for input documents ranging over an RHL H_{in}, and we want to check if the result belongs to an output type given by an RHL H_{out}. This problem amounts to inferring the types of the pattern variables of P when matching input documents from H_{in}, using them to infer the type (or type over-approximation) H of the result r, and then checking if H is a subtype of H_{out}. We mention two possible approaches:

1. Type inference for pattern variables, used in the XML programming language XDuce [3]. Certain syntactic restrictions and a pattern matching strategy

⋆ Supported by JSPS Grant-in-Aid no. 20500025 for Scientific Research (C).
⋆⋆ Supported by EC FP6 under the project SCIEnce—Symbolic Computation Infrastructure for Europe (Contract No. 026133).

are imposed in order to enable an easy static type reconstruction algorithm for the variables of P when matched against inputs of a given type. The types inferred for the pattern variables can be used to compute an over-approximation of the type of output r, and check if it is a subtype of H_{out}.

2. Type inference for the tuple of all variables of a pattern. In this case, we compute for every pattern P with variables $x_1, \ldots, x_n$ a finite set

$$\overline{H}_p = \{H_{i,1} \times \ldots \times H_{i,n} \mid i \in \{1, \ldots, p\}\}$$

of cartesian products of RHLs such that: $\{x_1 \mapsto h_1, \ldots, x_n \mapsto h_n\}$ is a matcher of P against some input from H_{in} iff $(h_1, \ldots, h_n)$ belongs to some cartesian product from $\overline{H}_p$. Alternatively, we could say that we infer the type $\sum_{i=1}^{p}(H_{i,1} \times \ldots \times H_{i,n})$ for the tuple of pattern variables $(x_1, \ldots, x_n)$. There are p possibilities for the type of $(x_1, \ldots, x_n)$, which can be used to compute an over-approximation of the type of output r and then check if it is a subtype of H_{out}.

This type inference approach works for the class of patterns[1] proposed by us in [5]. They are similar to the patterns of XDuce, but lack features such as global pattern names and pattern bindings. However, these patterns have some extra features that are desirable for XML querying: (a) they can be nonlinear, (b) variables can occur below iteration, and (c) there is no predefined matching strategy.

The first type checking approach is easier, but the second approach is more accurate and relies on factorizations of regular hedge languages, which are described in Sect. 2.1. The following example illustrates a situation when the second type checking approach is more accurate.

Example 1. Consider the type checking problem for input type $f(a^\star\, b^\star)^\star$, output type $f(a^\star b^\star a^\star b^\star a^\star)$, and transformation rule $f(x\,y)\,f(y\,x) \to f(x\,y\,x)$.

The first type checking approach infers that x is of type $a^\star b^\star$, and y is of type $a^\star b^\star$. Therefore $f(x\,y\,x)$ is of type $f(a^\star b^\star a^\star b^\star a^\star b^\star)$. Since this is not a subtype of the output type $f(a^\star b^\star a^\star b^\star a^\star)$, type checking fails.

The second approach works by noting that $f(x\,y)\,f(y\,x)$ matches an input from $f(a^\star b^\star)^\star$ if and only if both $x\,y$ and $y\,x$ match inputs from $a^\star b^\star$. Then, instead of computing the types for x and y independently of each other, we compute the type of a pair (x, y). From the type of (x, y), by easy simplifications we obtain that the type of $f(x\,y\,x)$ is $f(a^\star b^\star a^\star b^\star)$. Since this is a subtype of the output type $f(a^\star b^\star a^\star b^\star a^\star)$, type checking succeeds. The crucial step in this approach is the computation the type of the tuple (x, y), which heavily relies on factorization computations: First factorizing $f(a^\star b^\star)^\star$, then factorizing $a^\star b^\star$. We will return to this example at the end of the paper to see how the computations are done. □

Regular hedge language factorization algorithms have been described in [6,5]. In this paper we propose a significant improvement over those, by developing a new

[1] They are called regular hedge expressions in [5].

algorithm to compute the factor matrix of an RHL that is more efficient and easier to analyze. The improvement is based on a new representation of RHLs, by deterministic, reduced, and complete linear hedge automata.

The paper is structured as follows. Section 2 introduces the main notions we use in the investigation of regular hedge language factorization, and recalls some well known results. Section 3 presents a simple algorithm for the computation of automaton representations for all right factors of an RHL. In Sect. 4 we propose an algorithm to compute automata representations for the elements of the factor matrix of an RHL. The algorithm makes use of the automaton representations of the right factors of the RHL, and of their corresponding specification by linear systems of hedge language equations. Section 5 concludes by comparing the algorithms presented here with those presented in [6] and [5].

2 Preliminaries

Hedges over an unranked alphabet Σ of hedge labels and finite set of constants $\mathcal{K}$ are finite sequences of trees generated by the grammar

$$h ::= \epsilon \mid k \mid a(h)\, h$$

where ϵ denotes the empty sequence, $k \in \mathcal{K}$, and $a \in \Sigma$. A *tree* over Σ and $\mathcal{K}$ is a hedge of the form k or $a(h)$. Trees of the form $a(\epsilon)$ are abbreviated by a. We write $\mathcal{H}(\Sigma, \mathcal{K})$ for the set of hedges over Σ and $\mathcal{K}$, and $\mathcal{T}(\Sigma, \mathcal{K})$ for the set of trees over Σ and $\mathcal{K}$. Also, we abbreviate $\mathcal{H}(\Sigma, \emptyset)$ by $\mathcal{H}(\Sigma)$, and $\mathcal{T}(\Sigma, \emptyset)$ by $\mathcal{T}(\Sigma)$. A *hedge language* over Σ is a subset of $\mathcal{H}(\Sigma)$. From now on we assume implicitly that H, possibly subscripted, denotes regular hedge languages. The *concatenation* of H_1 and H_2 is the hedge language $H_1\, H_2 := \{h_1\, h_2 \mid h_1 \in H_1, h_2 \in H_2\}$; and the *asterate* of H is the hedge language $H^* := \{\epsilon\} \cup \bigcup_{n=1}^{\infty} \{h_1\, \ldots\, h_n \mid h_1, \ldots, h_n \in H\}$.

A *regular hedge language* (RHL) over an unranked alphabet Σ is a language accepted by a *hedge automaton* (HA). According to [7], such an automaton is a tuple $M = (Q, \Sigma, F, \Delta_M)$ where: Q is a finite set of states; Δ_M is a set of transition rules of the form $a(R) \to q$ where $q \in Q$ and R is a regular language over Q; and F is a regular set over Q, called the *final state sequence set* of M. If we write $\to_M$ for the rewrite relation induced on $\mathcal{H}(\Sigma, Q)$ by the rewrite system $\{a(w) \to q \mid a(R) \to q \in \Delta_M, w \in R\}$, then the language accepted by M is $L(M) := \{h \in \mathcal{H}(\Sigma) \mid \exists w \in F.\, h \to_M^* w\}$.

In this paper we propose another representation of RHLs, by so called linear hedge automata. A *linear hedge automaton* (LHA) over an unranked alphabet Σ is a tuple $\mathcal{A} = (Q, \Sigma, Q_{\mathtt{f}}, \Delta)$ where: Q is a finite set of states; Δ is a set of transition rules of the form $\epsilon \to q$ or $a(q_1)\, q_2 \to q$ with $q, q_1, q_2 \in Q$ and $a \in \Sigma$; and $Q_{\mathtt{f}} \subseteq Q$ is the set of final states of $\mathcal{A}$. An *ϵ-transition* is a transition rule of the form $\epsilon \to q$. $\mathcal{A}$ is *deterministic* if there are no two transition rules with the same left hand side.

Let $\to_{\mathcal{A}}$ be the rewrite relation induced by the rewrite system Δ on $\mathcal{H}(\Sigma, Q)$. Then the language accepted by $\mathcal{A}$ in a state $q \in Q$ is $L(\mathcal{A}, q) := \{h \in \mathcal{H}(\Sigma) \mid h \to_{\mathcal{A}}^* q\}$. For every $Q' \subseteq Q$ we define $L(\mathcal{A}, Q') := \bigcup_{q \in Q'} L(\mathcal{A}, q)$. The language *accepted* by $\mathcal{A}$ is $L(\mathcal{A}) := L(\mathcal{A}, Q_{\mathtt{f}})$. A state q of an LHA $\mathcal{A}$ is *accessible* if $L(\mathcal{A}, q) \neq \emptyset$. The LHA $\mathcal{A}$ is *reduced* if all its states are accessible. $\mathcal{A}$ is *complete* if for every $a \in \Sigma$ and $q_1, q_2 \in Q$ there exists a transition rule $a(q_1)\, q_2 \to q \in \Delta$.

A convenient representation of LHAs is by a *linear system of hedge language equations* (LSH). The *LSH representation* of an LHA $\mathcal{A} = (\{q_1, \ldots, q_n\}, \Sigma, Q_{\mathtt{f}}, \Delta)$ is the pair $(Q_{\mathtt{f}}, S)$ where

$$S : \begin{cases} q_1 = c_1 + \ell_{1,1}\, q_1 + \cdots + \ell_{1,n}\, q_n \\ \vdots \\ q_n = c_n + \ell_{n,1}\, q_1 + \cdots + \ell_{n,n}\, q_n \end{cases} \qquad \text{with } c_i := \begin{cases} 1 \text{ if } \epsilon \to q_i \in \Delta, \\ 0 \text{ otherwise} \end{cases}$$

and $\ell_{i,j} := \sum_{a(q)\, q_j \to q_i \in \Delta} a(q)$ for all $1 \leq i, j \leq n$. The variables of this system of equations are $q_1, \ldots, q_n$, and they correspond to the states of the LHA that is being represented. Therefore, from now on, we will refer to the elements of Q either as states of the LHA or as variables of the corresponding LSH.

The equations of S are between expressions which belong to the larger class of regular hedge expressions $\mathrm{HReg}(\Sigma, Q)$ defined by the grammar

$$e ::= 0 \mid 1 \mid q \mid a(e) \mid e + e \mid e\, e \mid e^{\star}.$$

Such expressions are interpreted with respect to an *assignment* for Q, which is a mapping $\sigma : Q \to 2^{\mathcal{H}(\Sigma)}$. The interpretation of $e \in \mathrm{HReg}(\Sigma, Q)$ with respect to an assignment σ is defined as follows: $[\![0]\!]_\sigma := \emptyset$, $[\![1]\!]_\sigma := \{\epsilon\}$, $[\![q]\!]_\sigma := \sigma(q)$, $[\![a(e)]\!]_\sigma := \{a(h) \mid h \in [\![e]\!]_\sigma\}$, $[\![e_1 + e_2]\!]_\sigma := [\![e_1]\!]_\sigma \cup [\![e_2]\!]_\sigma$, $[\![e_1\, e_2]\!]_\sigma := [\![e_1]\!]_\sigma\, [\![e_2]\!]_\sigma$, and $[\![e^\star]\!]_\sigma := [\![e]\!]_\sigma^*$. A *solution* of S is an assignment σ such that $\sigma(q_i) = [\![c_i + \ell_{i,1}\, q_1 + \ldots + \ell_{i,n}\, q_n]\!]_\sigma$ for all $1 \leq i \leq n$. We recall from [6] that an LSH S has a unique solution σ_S, and that σ_S binds the elements of Q to RHLs. In general, $L(\mathcal{A}, q) = \sigma_S(q)$ for all $q \in Q$.

Example 2. Let $\mathcal{A} = (\{q_1, q_2, q_3\}, \{a, b\}, \{q_1\}, \Delta)$ with

$$\Delta := \{\epsilon \to q_1,\ a(q_1)\, q_2 \to q_1,\ b(q_2)\, q_2 \to q_1,\ a(q_1)\, q_1 \to q_2,\\ a(q_1)\, q_2 \to q_2,\ a(q_3)\, q_2 \to q_3\}.$$

Then $\mathcal{A}$ is nondeterministic because the transition rules $a(q_1)\, q_2 \to q_1$ and $a(q_1)\, q_2 \to q_2$ have the same left hand side. The LSH representation of $\mathcal{A}$ is the pair $(\{q_1\}, S)$ where S is the LSH

$$S : \begin{cases} q_1 = 1 + (a(q_1) + b(q_2))\, q_2 \\ q_2 = 0 + a(q_1)\, q_1 + a(q_1)\, q_2 \\ q_3 = 0 + a(q_3)\, q_2 \end{cases}$$

It is not hard to see from the structure of S that state q_3 is not accessible, therefore $\mathcal{A}$ is not a reduced LHA. □

The subset construction of a deterministic, complete, and reduced finite tree automaton equivalent to a nondeterministic finite tree automaton can be easily adapted to linear hedge automata, so we can conclude the following lemma.

Lemma 1. *For every LHA $\mathcal{A}$ there exists a deterministic, complete, and reduced LHA $\mathcal{A}_{\mathrm{d}}$ such that $L(\mathcal{A}) = L(\mathcal{A}_{\mathrm{d}})$.*

Thus we can represent any RHL by a deterministic, complete, and reduced LHA.

Example 3. The determinization of the LHA $\mathcal{A}$ from Example 2 produces the deterministic, complete, and reduced LHA $\mathcal{A}_{\mathrm{d}} := (\{s_1, s_2, s_3, s_4\}, \Sigma, \{s_1, s_4\}, \Delta_{\mathrm{d}})$ whose LSH representation is $(\{s_1, s_4\}, S)$ with

$$S : \begin{cases} s_1 = 1 + (b(s_2) + b(s_4))\, s_2 + (b(s_2) + b(s_4))\, s_4 \\ s_2 = 0 + (a(s_1) + a(s_4))\, s_1 \\ s_3 = 0 + (a(s_2) + a(s_3) + \sum_{i=1}^4 b(s_i))\, s_1 \\ \qquad\quad + (a(s_2) + a(s_3) + b(s_1) + b(s_3))\, s_2 \\ \qquad\quad + (\sum_{i=1}^4 a(s_i) + \sum_{i=1}^4 b(s_i))\, s_3 \\ \qquad\quad + (a(s_2) + a(s_3) + b(s_1) + b(s_3))\, s_4 \\ s_4 = 0 + (a(s_1) + a(s_4))\, s_2 + (a(s_1) + a(s_4))\, s_4. \end{cases}$$

2.1 Factorizations of Regular Hedge Languages

The factorization theory of RHLs [5] is a natural generalization of the factorization theory of regular languages [2]. We recall here main definitions and results from [5]. An *n-subfactorization* of an RHL H is a tuple $(H_1, \ldots, H_n)$ of hedge languages such that the concatenation $H_1 \cdots H_n$ is a subset of H. If we define the relation

$$(H_1, \ldots, H_n) < (H'_1, \ldots, H'_n) :\Leftrightarrow \begin{array}{l} H_i \subseteq H'_i \text{ for all } i \in \{1, \ldots, n\} \text{ and} \\ H_j \neq H'_j \text{ for some } j \in \{1, \ldots, n\} \end{array}$$

then we can talk about $<$-maximal n-subfactorizations of H, also known as *n-factorizations* of H. The components of such n-factorizations are called *factors.* A *left factor* of H is the first factor of an n-factorization of H, and a *right factor* is the last factor of an n-factorization of H. For the rest of this section we assume implicitly that H, L, M are RHLs, $\mathtt{F}(H)$ is the set of factors of H, $\mathtt{LF}(H)$ is the set of left factors of H and $\mathtt{RF}(H)$ is the set of right factors of H.

Example 4. The RHL $H = \{a(\epsilon)^m\, b(\epsilon)^n a(\epsilon)^p \mid m, n, p \in \mathbb{N}\}$ over signature $\Sigma = \{a, b, c\}$ has five possible 2-factorizations (H_1, H_2): either $(\mathcal{H}(\{a, b, c\}), \emptyset)$, or $(\{a(\epsilon)^m \mid m \in \mathbb{N}\}, H)$, or $(\{a(\epsilon)^m b(\epsilon)^n \mid m, n \in \mathbb{N}\}, \{b(\epsilon)^n a(\epsilon)^p \mid n, p \in \mathbb{N}\})$, or $(H, \{a(\epsilon)^p \mid p \in \mathbb{N}\})$, or $(\emptyset, \mathcal{H}(\{a, b, c\}))$. In this case we have

$$\mathtt{LF}(H) = \{\mathcal{H}(\{a, b, c\}), \{a(\epsilon)^m \mid m \in \mathbb{N}\}, \{a(\epsilon)^m b(\epsilon)^n \mid m, n \in \mathbb{N}\}, H, \emptyset\} \text{ and}$$
$$\mathtt{RF}(H) = \{\emptyset, H, \{b(\epsilon)^n a(\epsilon)^p \mid n, p \in \mathbb{N}\}, \{a(\epsilon)^p \mid p \in \mathbb{N}\}, \mathcal{H}(\{a, b, c\})\}.$$

A remarkable fact is that the factors of an RHL H are finitely many [5]. Moreover, the factors of H can be indexed such that $\mathtt{F}(H) = \{F_{i,j} \mid 1 \leq i, j \leq p\}$ and:

- $F_{i,k}\, F_{k,j} \subseteq F_{i,j}$ for all $i, j, k \in \{1, \ldots, p\}$, and
- There exist $l, r \in \{1, \ldots, p\}$ such that $F_{l,r} = H$ and, for any k-subfactorization $(H_1, \ldots, H_k)$ of H there exist $u_1, \ldots, u_{k+1} \in \{1, \ldots, p\}$ with $u_1 = l$ and $u_{k+1} = r$, such that $H_i \subseteq F_{u_i,u_{i+1}}$ for all $i \in \{1, \ldots, k\}$.

The matrix $(F_{i,j})_{1 \le i,j \le p}$ is called the *factor matrix* of H.

In order to achieve a better characterization of the factors of an RHL, we introduce the following auxiliary notions:

- $h^{-1}H := \{h' \in \mathcal{H}(\Sigma) \mid h\, h' \in H\}$ is the *left quotient* of H with respect to a hedge h,
- $H\, h^{-1} := \{h' \in \mathcal{H}(\Sigma) \mid h'\, h \in H\}$ is the *right quotient* of H with respect to a hedge h,
- $L \rhd M := \{h' \in \mathcal{H}(\Sigma) \mid \forall h \in L.\, h\, h' \in M\}$ is the *product derivative* of M with respect to L,
- $M \lhd L := \{h' \in \mathcal{H}(\Sigma) \mid \forall h \in L.\, h'\, h \in M\}$ is the *product antiderivative* of M with respect to L.
- The set of *hedge derivatives* of H is $\partial(H) := \{h^{-1}H \mid h \in \mathcal{H}(\Sigma)\}$.

We recall from [5,6] that for any RHLs H, M, L we have:

- $\partial(H)$ is a finite set of RHLs,
- $\mathtt{RF}(H)$ is the closure of $\partial(H)$ under intersection, i.e., $R \in \mathtt{RF}(H)$ iff $R \in \partial(H)$ or there exist $M_1, \ldots, M_p \in \partial(H)$, $p > 1$, such that $R = M_1 \cap \cdots \cap M_p$,
- $M \lhd L$ is an RHL, and $M \lhd L = \bigcap_{h \in L}(M\, h^{-1})$,
- If $\mathtt{LF}(H) = \{L_1, \ldots, L_p\}$ then we can define $F_{i,j} := L_i \rhd L_j$,
- If $\mathtt{RF}(H) = \{R_1, \ldots, R_p\}$ then we can define $F_{i,j} := R_i \lhd R_j$.

In the rest of this paper we will investigate how to compute the factor matrix of an RHL by making use of these properties and of the representation of RHLs by deterministic, complete, and reduced LHAs. Our approach is to compute first $\{R_1, \ldots, R_p\} := \mathtt{RF}(H)$ and then to define the factor matrix by $F_{i,j} := R_i \lhd R_j$ for all $i, j \in \{1, \ldots, p\}$.

3 Computation of Right Factors

In this section we investigate the problem of computing all right factors of an RHL H accepted by a deterministic, complete, and reduced LHA $\mathcal{A}$. We solve this problem in two steps. First, we compute LHAs for the RHLs of the finite set $\partial(H)$. Then, we compute LHAs for the right factors by using the automata computed in the first step.

Suppose $\mathcal{A} = (Q, \Sigma, Q_{\mathtt{f}}, \Delta)$ with $Q = \{q_1, \ldots, q_n\}$, and let $(Q_{\mathtt{f}}, S)$ be the LSH representation of $\mathcal{A}$. To simplify the analysis of the structure of $\partial(H)$, we define for every $q, q' \in Q$, $Q' \subseteq Q$, and $a \in \Sigma$ the sets

$$\begin{aligned}\mathtt{cut}_{\Delta}(q, a(q')) &:= \{q'' \in Q \mid a(q')\, q'' \to q \in \Delta\},\\ \mathtt{cut}_{\Delta}(Q', a(q')) &:= \bigcup_{q \in Q'} \mathtt{cut}_{\Delta}(q, a(q')).\end{aligned}$$

These notions will help us to identify a finitary characterization of the set of RHLs $\partial(H)$. The key observation is the following lemma.

Lemma 2. *$a(h)^{-1}L(\mathcal{A}, Q') = L(\mathcal{A}, \mathtt{cut}_{\Delta}(Q', a(q)))$ holds for all $a \in \Sigma$, $q \in Q$, $Q' \subseteq Q$, and $h \in L(\mathcal{A}, q)$.*

The following result is an easy corollary of Lemma 2.

Corollary 1. *Let $\mathcal{G}_{\Delta}$ be the directed graph whose nodes are the subsets of Q, and set of edges is $\{Q' \to Q'' \mid a \in \Sigma, q \in Q, Q'' = \mathtt{cut}_{\Delta}(Q', a(q))\}$. Then*

$$\partial(H) = \{L(\mathcal{A}, Q') \mid \textit{there exists a path from } Q_{\mathtt{f}} \textit{ to } Q' \textit{ in } \mathcal{G}_{\Delta}\}.$$

Once we know $\partial(H)$, we can compute $\mathtt{RF}(H)$ as the closure of $\partial(H)$ under intersections. At this stage, it is very useful to recall that, since $\mathcal{A}$ is deterministic, we have $L(\mathcal{A}, q) \cap L(\mathcal{A}, q') = \emptyset$ whenever $q, q' \in \mathcal{A}$ and $q \neq q'$. Therefore $L(\mathcal{A}, Q_1) \cap L(\mathcal{A}, Q_2) = L(\mathcal{A}, Q_1 \cap Q_2)$ for all subsets Q_1 and Q_2 of Q. Thus, if $\partial(H) = \{L(\mathcal{A}, Q_i) \mid 1 \leq i \leq m\}$, then

$$\mathtt{RF}(H) = \Big\{L(\mathcal{A}, \bigcap_{i \in I} Q_i) \mid \emptyset \neq I \subseteq \{1, \ldots, m\}\Big\}.$$

Example 5. Let $\mathcal{A}_{\mathrm{d}} = (\{s_1, s_2, s_3, s_4\}, \Sigma, \Delta_{\mathrm{d}}, \{s_1, s_4\})$ be the LHA from Example 3, and $H = L(\mathcal{A}_{\mathrm{d}})$. The set of nodes reachable from $\{s_1, s_4\}$ in $\mathcal{G}_{\Delta_{\mathrm{d}}}$ is $\{Q_1, Q_2, Q_3\}$ where $Q_1 := \emptyset$, $Q_2 := \{s_2, s_4\}$, and $Q_3 := \{s_1, s_2, s_4\}$. Therefore $\partial(H) = \{L(\mathcal{A}_{\mathrm{d}}, Q_i) \mid 1 \leq i \leq 3\}$. Since $\{\bigcap_{i \in I} Q_i \mid \emptyset \neq I \subseteq \{1, 2, 3\}\} = \{Q_1, Q_2, Q_3\}$, we conclude that $\mathtt{RF}(H) = \{L(\mathcal{A}_{\mathrm{d}}, Q_i) \mid 1 \leq i \leq 3\}$.

A slightly surprising fact is that $L(\mathcal{A}_{\mathrm{d}}, \{s_1, s_4\}) = H \in \mathtt{RF}(H)$ but $\{s_1, s_4\} \notin \{Q_1, Q_2, Q_3\}$. As it turns out, we have $L(\mathcal{A}_{\mathrm{d}}, \{s_1, s_4\}) = L(\mathcal{A}_{\mathrm{d}}, Q_3)$. □

4 Factor Matrix Computation

In this section we address the problem of computing the factor matrix of an RHL represented by an LHA. To be more specific, from now on we assume $H = L(\mathcal{A})$ where $\mathcal{A} = (Q, \Sigma, Q_{\mathtt{f}}, \Delta)$ with $Q = \{q_1, \ldots, q_n\}$ is a deterministic, complete, and reduced LHA. We saw in the previous section how to compute $Q_1, \ldots, Q_p \subseteq Q$ such that $\mathtt{RF}(H) = \{L(\mathcal{A}, Q_i) \mid 1 \leq i \leq p\}$. Then we can define the factor matrix $(F_{i,j})_{1 \leq i,j \leq p}$ where $F_{i,j} := L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$.

In the following two subsections we will show how to compute LHAs for the product antiderivatives $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$ when $1 \leq i, j \leq p$. First, we indicate in Subsect. 4.1 a simple algorithm to compute LHAs for the product antiderivatives $L(\mathcal{A}, Q') \lhd L(\mathcal{A}, q)$ when $q \in Q$ and $Q' \subseteq Q$. Then, in Subsect. 4.2 we propose an algorithm that computes LHAs for the RHLs of the product antiderivatives $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$, $1 \leq i, j \leq p$ from LHAs for the product antiderivatives $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, q)$, $1 \leq i \leq p$, where $q \in Q$.

From now on we assume that the LSH representation of $\mathcal{A}$ is $(Q_{\mathtt{f}}, S)$ where

$$S : \begin{cases} q_1 = c_1 + \ell_{1,1}\, q_1 + \cdots + \ell_{1,n}\, q_n \\ \vdots \\ q_n = c_n + \ell_{n,1}\, q_1 + \cdots + \ell_{n,n}\, q_n \end{cases}$$

and that σ is the unique solution of S. Also, we assume that Q' is subset of Q.

4.1 LHA Computation for $L(\mathcal{A}, Q') \lhd L(\mathcal{A}, q)$

Let $Q = \{q_1, \ldots, q_n\}$. We fix arbitrarily a state $q \in Q$. Then for every $1 \leq i \leq n$ and $h \in L(\mathcal{A}, q)$ we have

$$[\![q_i]\!]_\sigma\, h^{-1} = [\![c_i + \ell_{i,1}\, q_1 + \cdots + \ell_{i,n}\, q_n]\!]_\sigma\, h^{-1} = [\![d_i + \ell_{i,1}\, r_1^{(q)} + \cdots + \ell_{i,n}\, r_n^{(q)}]\!]_{\theta_q}$$

where

- $d_i = 1$ if $h \in [\![q_i]\!]_\sigma$ and $d_i = 0$ otherwise,
- θ_q is the extension of σ with the following bindings for the fresh variables $r_1^{(q)}, \ldots, r_n^{(q)}$: $\theta_q(r_i^{(q)}) := [\![q_i]\!]_\sigma\, h^{-1}$ for all $1 \leq i \leq n$.

Since $\mathcal{A}$ is deterministic, we have $L(\mathcal{A}, q_i) \cap L(\mathcal{A}, q) = \emptyset$ whenever $q_i \neq q$. Therefore $d_i = 1$ iff $h \in [\![q_i]\!]_\sigma$ iff $\emptyset \neq [\![q_i]\!]_\sigma \cap [\![q]\!]_\sigma$ iff $q_i = q$. Thus, if we define for all q', q'' and $1 \leq i \leq n$:

$$\rho_q(q_i) := r_i^{(q)}, \quad \rho_q(Q') := \{\rho_q(q') \mid q' \in Q'\}, \quad \delta_{q',q''} := \begin{cases} 1 \text{ if } q' = q'' \\ 0 \text{ otherwise} \end{cases}, \quad \text{and}$$

$$S_q : \begin{cases} r_l^{(q)} = \delta_{q_l,q} + \ell_{l,1}\, r_1^{(q)} + \cdots + \ell_{l,n}\, r_n^{(q)} & (1 \leq l \leq n) \\ \quad q_l = c_l \quad + \ell_{l,1}\, q_1 + \cdots + \ell_{l,n}\, q_n & (1 \leq l \leq n) \end{cases}$$

then $(S_q, \rho_q(Q'))$ is an LSH representation for $L(\mathcal{A}, Q')\, h^{-1}$. Since

$$L(\mathcal{A}, Q') \lhd L(\mathcal{A}, q) = \bigcap_{h \in L(\mathcal{A}, q)} L(\mathcal{A}, Q')\, h^{-1} \quad \text{and}$$

$$L(\mathcal{A}, Q')\, h_1^{-1} = L(\mathcal{A}, Q')\, h_2^{-1} \text{ for all } h_1, h_2 \in L(\mathcal{A}, q),$$

we learn that $L(\mathcal{A}, Q') \lhd L(\mathcal{A}, q) = L(\mathcal{A}, Q')\, h^{-1}$. Let $R_q := \{r_1^{(q)}, \ldots, r_n^{(q)}\}$ and $\mathcal{A}_q := (Q \cup R_q, \Sigma, \rho_q(Q_{\mathtt{f}}), \Delta_q)$ be the LHA whose LSH representation is (Q, S_q). Then $L(\mathcal{A}, Q') \lhd L(\mathcal{A}, q) = L(\mathcal{A}_q, \rho_q(Q'))$.

Note that the computation of the LSH representation of $\mathcal{A}_q$ involves only duplications of the equations of S followed by some trivial variable renamings and alterations of their constant parts. Another useful observation is that every LHA $\mathcal{A}_{q_i}$ with $1 \leq i \leq n$ is *almost* deterministic, because the only transition rules of Δ_{q_i} with same left hand side are $\epsilon \to r_i^{(q_i)}$ and the ϵ-transition of Δ. The following lemma is an easy consequence of this observation.

Lemma 3. *$L(\mathcal{A}_q, r_i^{(q)}) \cap L(\mathcal{A}_q, r_j^{(q)}) = \emptyset$ for all $i, j \in \{1, \ldots, n\}$ and $i \neq j$.*

4.2 LHA Computation for $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$

In order to compute an LHA for $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$, we can proceed as follows. If $Q_j = \emptyset$ then $L(\mathcal{A}, Q_j) = \emptyset$ and $(L(\mathcal{A}, Q_i) \lhd \emptyset) = \mathcal{H}(\Sigma) = L(\mathcal{A}, Q)$, where the last equality follows from the fact that $\mathcal{A}$ is complete. If $Q_j \neq \emptyset$ then

$$L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j) = \bigcap_{q \in Q_j} (L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, q)) = \bigcap_{q \in Q_j} L(\mathcal{A}_q, \rho_q(Q_i)).$$

Let $Q_j := \{q_{k_1}, \ldots, q_{k_d}\}$ with $d \geq 1$, and $\mathtt{I}_i := \{l \mid q_l \in Q_i\}$. For every $q \in Q_j$, the system of equations S_q can be rewritten as follows:

$$S_q : \begin{cases} r_l^{(q)} = \delta_{q_l, q} + \sum_{a \in \Sigma} \sum_{u=1}^{n} a(q_u)\, e_{a(q_u),l,q} & (1 \leq l \leq n) \\ q_l = c_l \qquad\qquad +\ell_{l,1}\, q_1 + \cdots + \ell_{l,n}\, q_n & (1 \leq l \leq n) \end{cases}$$

where $e_{a(q_u),l,q} = \sum_{r \in \mathtt{cut}_{\Delta_q}(r_l^{(q)}, a(q_u))} r$. Suppose θ_q is the unique solution of S_q for every $q \in Q_j$. Then

$$\begin{aligned} L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j) &= \textstyle\bigcap_{q \in Q_j} L(\mathcal{A}_q, \rho_q(Q_i)) \\ &= \textstyle\bigcap_{q \in Q_j} \bigcup_{l \in \mathtt{I}_i} [\![r_l^{(q)}]\!]_{\theta_q} = \bigcup_{(l_1,\ldots,l_d) \in \mathtt{I}_i^d} \bigcap_{s=1}^{d} [\![r_{l_s}^{(q_{k_s})}]\!]_{\theta_{q_{k_s}}}. \end{aligned}$$

where $\mathtt{I}_i^d = \underbrace{\mathtt{I}_i \times \cdots \times \mathtt{I}_i}_{d \text{ times}}$. At this stage, we can start generating an LSH representation for the product antiderivative $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$. Our algorithm works in stages, by keeping track of (1) a list E of equations generated so far, (2) a set V_p of variables produced so far, and (3) a set V_a of variables assigned so far. In addition to the variables from Q and the equations of S, the algorithm produces fresh variables of the form $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}$ with $l_1, \ldots, l_d \in \{1, \ldots, n\}$, and corresponding equations such that the solution of the new system of equations binds $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}$ to $\bigcap_{s=1}^{d} [\![r_{l_s}^{(q_{k_s})}]\!]_{\theta_{q_{k_s}}}$. The main goal is to produce equations for all variables of the set $\{r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})} \mid (l_1, \ldots, l_d) \in \mathtt{I}_i^d\}$ and the other fresh variables that show up in the construction process. The initial values of V_p and V_a are $V_\mathsf{p} = Q \cup \{r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})} \mid (l_1, \ldots, l_d) \in \mathtt{I}_i^d\}$, $V_\mathsf{a} = Q$, and E consists of the equations of S. At every stage, the algorithm enlarges V_p, V_a, and E by performing the following operations:

– Select $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})} \in V_\mathsf{p} \setminus V_\mathsf{a}$; $V_\mathsf{a} := V_\mathsf{a} \cup \{r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}\}$
– Produce an equation for $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}$ such that E extended with it has a solution that binds $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}$ to $\bigcap_{s=1}^{d} [\![r_{l_s}^{(q_{k_s})}]\!]_{\theta_{q_{k_s}}}$. The right hand side of the equation for $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}$ is produced by intersecting the right hand sides of the equations

$$\begin{aligned} &(r_{l_1}^{(q_{k_1})} = \delta_{q_{l_1}, q_{k_1}} + \textstyle\sum_{a \in \Sigma} \sum_{u=1}^{n} a(q_u)\, e_{a(q_u),l_1,q_{k_1}}) \in S_{q_{k_1}} \text{ with solution } \theta_{k_1} \\ &\quad \vdots \\ &(r_{l_d}^{(q_{k_d})} = \delta_{q_{l_d}, q_{k_d}} + \textstyle\sum_{a \in \Sigma} \sum_{u=1}^{n} a(q_u)\, e_{a(q_u),l_d,q_{k_d}}) \in S_{q_{k_d}} \text{ with solution } \theta_{k_d} \end{aligned}$$

and using the facts that $[\![a(q_u)\, e_{a(q_u),l_s,q_{k_s}}]\!]_{\theta_{q_{k_s}}} = [\![a(q_u)]\!]_\sigma\, [\![e_{a(q_u),l_s,q_{k_s}}]\!]_{\theta_{q_{k_s}}}$, and $[\![a(q)]\!]_\sigma \cap [\![b(q')]\!]_\sigma = \emptyset$ whenever $(a,q) \neq (b,q')$. It follows that $[\![a(q)\, e]\!]_{\theta_{q_{k_i}}} \cap [\![b(q')\, e']\!]_{\theta_{q_{k_j}}} = \emptyset$ whenever $(a,q) \neq (b,q')$. We obtain

$$r^{(q_{k_1},\ldots,q_{k_d})}_{(l_1,\ldots,l_d)} = \prod_{s=1}^{d} \delta_{q_{l_s},q_{k_s}} + \sum_{a\in\Sigma}\sum_{u=1}^{n} a(q_u)\Big(\sum_{(m_1,\ldots,m_d)\in \mathtt{S}^{a(q_u)}_{(l_1,\ldots,l_d)}} r^{(q_{k_1},\ldots,q_{k_d})}_{(m_1,\ldots,m_d)}\Big) \quad (1)$$

where

$$\mathtt{S}^{a(q_u)}_{(l_1,\ldots,l_d)} = \Big\{(m_1,\ldots,m_d) \mid \forall s \in \{1,\ldots,d\}.\, r^{(q_{k_s})}_{m_s} \in \mathtt{cut}_{\Delta_{q_{k_s}}}(r^{(q_{k_s})}_{l_s}, a(q_u))\Big\}.$$

– Append equation (1) to E;
  $V_\mathtt{p} := V_\mathtt{p} \cup \bigcup_{a\in\Sigma}\bigcup_{u=1}^{n}\{r^{(q_{k_1},\ldots,q_{k_d})}_{(m_1,\ldots,m_d)} \mid (m_1,\ldots,m_d) \in \mathtt{S}^{a(q_u)}_{(l_1,\ldots,l_d)}\}$.
– If $V_\mathtt{p} = V_\mathtt{a}$, then stop and return $(\{r^{(q_{k_1},\ldots,q_{k_d})}_{(l_1,\ldots,l_d)} \mid (l_1,\ldots,l_d) \in \mathtt{I}^d_i\}, E)$. This is the LSH representation of an LHA for $L(\mathcal{A},Q_i) \lhd L(\mathcal{A},Q_j)$.

The algorithm will eventually stop because $V_\mathtt{p}$ and $V_\mathtt{a}$ are subsets of the finite set $Q \cup \{r^{(q_{k_1},\ldots,q_{k_d})}_{(l_1,\ldots,l_d)} \mid l_1,\ldots,l_d \in \{1,\ldots,n\}\}$, therefore they can not increase forever. □

Let us now look back at Example 1 and see how factorization theory helps there to infer the type of the tuple (x,y) of variables of pattern $f(x\,y)\,f(y\,x)$ when matched against inputs of type $f(a^\star b^\star)^\star$. Let's write $P \ll T$ for the type inference problem of the tuple $(x_1,\ldots,x_n)$ of variables of a pattern P when matched against inputs of type T. This problem can be solved by recursion on the structure of P. In this example, we have $P_1\,P_2 \ll T$ where $P_1 = f(x\,y)$, $P_2 = f(y\,x)$, and $T = f(a^\star\, b^\star)^\star$. This problem can be reduced to solving the problems $P_1 \ll T_1$ and $P_2 \ll T_2$, where T_1, T_2 are nonempty and $\subseteq$-maximal types such that $T_1\,T_2 \subseteq T$. Such tuples (T_1,T_2) are called 2-factorizations of T, and can be computed effectively. The only such 2-factorization of $f(a^\star b^\star)^\star$ is $(T_1,T_2) = (f(a^\star b^\star)^\star, f(a^\star b^\star)^\star)$, therefore $P \ll f(a^\star b^\star)^\star$ is reduced to $f(x\,y) \ll f(a^\star b^\star)^\star$ and $f(y\,x) \ll f(a^\star b^\star)^\star$. The first subproblem is equivalent to $x\,y \ll a^\star b^\star$, and the second subproblem is equivalent to $y\,x \ll a^\star\, b^\star$. To solve these subproblems, we look for nonempty types T_3, T_4 such that (T_3,T_4) is a 2-factorization of $a^\star b^\star$. There are two possibilities: either $(T_3,T_4) = (a^\star, a^\star b^\star)$, or $(T_3,T_4) = (a^\star b^\star, b^\star)$. Thus $x\,y \ll a^\star b^\star$ reduces to $(x \ll T_3$ and $y \ll T_4)$ where $(T_3,T_4) \in \{(a^\star, a^\star b^\star), (a^\star b^\star, b^\star)\}$. This means (x,y) is of type $(a^\star \times a^\star b^\star) + (a^\star b^\star \times b^\star)$ where $\times$ denotes cartesian product and $+$ denotes union. Similarly, the second subproblem $y\,x \ll a^\star b^\star$ reduces to the fact that (x,y) is of type $(a^\star b^\star \times a^\star) + (b^\star \times a^\star b^\star)$. We conclude that (x,y) is of type

$$\begin{aligned}&((a^\star \times a^\star b^\star) + (a^\star b^\star \times b^\star)) \cap ((a^\star b^\star \times a^\star) + (b^\star \times a^\star b^\star))\\ &= (a^\star \times a^\star) + (1 \times a^\star b^\star) + (a^\star b^\star \times 1) + (b^\star \times b^\star).\end{aligned}$$

From here, one can easily compute an over-approximation of the type of $x\,y\,x$ as $a^\star a^\star a^\star + 1\,a^\star b^\star\,1 + a^\star b^\star\,1 a^\star b^\star + b^\star\,b^\star\,b^\star = a^\star b^\star a^\star b^\star$, therefore $f(x\,y\,x)$ is of type $f(a^\star b^\star a^\star b^\star)$.

5 Discussion

The approach to compute the factor matrix of an RHL proposed in this paper is more efficient than that from our previous works [5,6]. Our main insight is that, if we want to compute the factor matrix of an RHL, it is better to start with a representation by a deterministic, reduced, and complete LHA. Next, we noticed that the computation of the factor matrix of an RHL is easier to compute via computations of product antiderivatives between right factors than via product derivatives between left factors.

The following discussion aims at explaining the main differences between our former algorithms and those given in this paper. In [5,6], we considered an RHL H with an LSH representation of the form $(\{x_1\}, S)$. Such a representation corresponds to a nondeterministic LHA with one final state. The computation of the factor matrix is achieved in 3 steps. Step 1 computes representations for the elements of the finite set $\{H\,h^{-1} \mid h \in \mathcal{H}(\Sigma)\}$. Step 2 computes representations for the elements of $\mathtt{LF}(H)$ by using the fact that $\mathtt{LF}(H)$ coincides with the closure of $\{H\,h^{-1} \mid h \in \mathcal{H}(\Sigma)\}$ under intersection. Step 3 makes use of the fact that, if we know $\{L_1, \ldots, L_p\} := \mathtt{LF}(H)$, then the factor matrix of H is $(F_{i,j})_{1\leq i,j\leq p}$ with $F_{i,j} := L_i \rhd L_j$. Step 1 amounts to the computation of the least fixed point of a monotone operator on the set $2^{2^{\mathcal{X}}\times 2^{\mathcal{X}}}$ where $\mathcal{X}$ is the set of variables of S. The construction of LSHs for the elements of $\mathtt{LF}(H)$ performs intersections of equations in a similar way as the construction of product antiderivatives of right factors, except for the fact that it must account for the possible nonempty intersections $[\![a(x)]\!]_\sigma \cap [\![b(x')]\!]_\sigma$ when $(a, x) \neq (b, x')$. Finally, step 3 finds the LSH representations of the factor matrix of H via product derivative computations between the left factors of H.

Our new approach is to compute the factor matrix of H starting from a reduced, complete, and deterministic LHA $\mathcal{A} = (Q, \Sigma, Q_{\mathtt{f}}, \Delta)$ for H. To achieve a fair comparison with our former approach, we should account for the fact that the determinization algorithm of a nondeterministic LHA with n states has, in the worst case, 2^n states. Thus, we assume the worst situation: we compare the former approach when $\mathcal{X}$ has n elements, with the new approach when Q has 2^n states. The newly proposed computation of the factor matrix is carried out in 3 steps. The first step computes a set $\{Q_1, \ldots, Q_p\}$ of subsets of Q such that $\partial(H) = \{L(\mathcal{A}, Q_i) \mid i \in \{1, \ldots, p\}\}$. The sets $\{Q_1, \ldots, Q_p\}$ are the nodes reachable from $Q_{\mathtt{f}}$ in a directed graph $\mathcal{G}_\Delta$ whose nodes are the subsets of Q, thus, a graph with 2^{2^n} nodes.

We claim that new step 1 is more efficient than former step 1 because

- Former step 1 computes at most 2^n LSH representations for the RHLs in $\{H\,h^{-1} \mid h \in \mathcal{H}(\Sigma)\}$ via the computation of the least fixed point of a monotone operator over a set of size $2^{2^{2n}}$.
- New step 1 computes at most 2^n sets $Q_1, \ldots, Q_p \subseteq Q$ such that $\partial(H) = \{L(\mathcal{A}, Q_i) \mid i \in \{1, \ldots, p\}\}$.

– The computation of $Q_1, \ldots, Q_p$ is straightforward, whereas the operations required to compute the LSH representations for $\{H\,h^{-1} \mid h \in \mathcal{H}(\Sigma)\}$ are much more involved.

Also, we claim that new step 2 is more efficient then former step 2 because

– Former step 2 computes LSHs for left factors as intersections of at most 2^n RHLs represented by the nondeterministic LSHs computed in former step 1.
– In new step 2, a right factor is $L(\mathcal{A}, Q')$ where Q' is an intersection of sets from $\{Q_1, \ldots, Q_p\}$. Since $p \leq 2^n$, Q' is obtained by intersecting at most 2^n sets. This is more efficient than intersecting at most 2^n RHLs represented by nondeterministic LHSs.

New step 3 computes the factor matrix of H via product antiderivatives of right factors, whereas former step 3 computes it via product derivatives of left factors. The new approach is more efficient than the former one, mainly because both of them rely on computing representations for intersections of RHLs, but these computations can be simplified under the assumption $L(\mathcal{A}, q) \cap L(\mathcal{A}, q') = \emptyset$ whenever $q \neq q'$, which follows from the fact that $\mathcal{A}$ is deterministic.

References

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, San Francisco (2000)
2. Conway, J.H.: Regular Algebra and Finite Machines. Mathematics series. Chapman and Hall, Boca Raton (1971)
3. Hosoya, H., Pierce, B.C.: Regular expression pattern matching for XML. Journal of Functional Programming 13(6), 961–1004 (2002)
4. Hosoya, H., Pierce, B.C.: XDuce: A statically typed XML processing language. ACM Trans. Internet Techn. 3(2), 117–148 (2003)
5. Marin, M., Crăciun, A.: Factorizations of regular hedge languages. In: Proceedings of 11th Intl. Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2009), Timişoara, Romania. IEEE, Los Alamitos (September 2009), http://www.score.cs.tsukuba.ac.jp/~mmarin/synasc2009.pdf
6. Marin, M., Kutsia, T.: Linear systems for regular hedge languages. In: Grundspenkis, J., Kirikova, M., Manolopoulos, Y., Novickis, L. (eds.) ADBIS 2009 Workshops. LNCS, vol. 5968, pp. 104–112. Springer, Heidelberg (2010)
7. Murata, M.: Extended path expressions for XML. In: Proceedings of the 20th Symposium on Principles of Database Systems (PODS 2001), Santa Barbara, California, USA, pp. 126–137. ACM, New York (2001)
8. Thatcher, J.: There is a lot more to finite automata theory than you would have thought. Technical Report RC-2852 (#13407), IBM Thomas J. Watson Research Center, Yorktown, New York (1970)
9. Thatcher, J., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. Mathematical Systems Theory 2(1), 57–81 (1968)

Fast Parsing for Boolean Grammars: A Generalization of Valiant's Algorithm

Alexander Okhotin[⋆]

Department of Mathematics, University of Turku, Turku FI–20014, Finland
Academy of Finland
alexander.okhotin@utu.fi

Abstract. The well-known parsing algorithm for the context-free grammars due to Valiant ("General context-free recognition in less than cubic time", *Journal of Computer and System Sciences*, 10:2 (1975), 308–314) is refactored and generalized to handle the more general Boolean grammars. The algorithm reduces construction of the parsing table to computing multiple products of Boolean matrices of various size. Its time complexity on an input string of length n is $O(BMM(n) \log n)$, where $BMM(n)$ is the number of operations needed to multiply two Boolean matrices of size $n \times n$, which is $O(n^{2.376})$ as per the current knowledge.

1 Introduction

Context-free grammars are the universally accepted mathematical model of syntax, and their status is well-justified. On the one hand, their expressive means are *natural*, in the sense whatever they define is intuitively seen as the syntax of something. On the other hand, they can be implemented in a variety of efficient algorithms, including a straightforward cubic-time parser, as well as many practical parsing algorithms working much faster in special cases.

The main idea of the context-free grammars is *inductive definition* of syntactically correct strings. For example, a context-free grammar $S \to aSb \mid \varepsilon$ represents a definition of the form: a string has the property S if and only if either it is representable as awb for some string w with the property S, or if it is the empty string. Note that the vertical line in the above grammar is essentially a disjunction of two syntactical conditions. *Boolean grammars*, introduced by the author [7], are an extension of the context-free grammars, which maintains the main principle of inductive definition, but allows the use of any Boolean operations to combine syntactical conditions in the rules. At the same time, they inherit the basic parsing algorithms from the context-free grammars, including the Cocke–Kasami–Younger [7] along with its variant for unambiguous grammars [10], the Generalized LR [8], as well as the linear-time recursive descent [9].

The straightforward upper bound on the complexity of parsing for Boolean grammars is the same as in the context-free case: $O(n^3)$, where n is the length of

⋆ Supported by the Academy of Finland under grants 134860 and 218315.

the input string [7]. However, for the context-free grammars, there also exists an asymptotically faster parsing algorithm due to Valiant [12]: this algorithm computes the same parsing table as the simple Cocke–Kasami–Younger algorithm, but does so by offloading the most intensive computations into calls to a Boolean matrix multiplication procedure. The latter can be efficiently implemented in a variety of ways. Given two $n \times n$ Boolean matrices, a straightforward calculation of their product requires n^3 conjunctions and $(n-1)n^2$ disjunctions. An improved algorithm by Arlazarov et al. [2] reduces the number of bit operations to $O\big(\frac{n^3}{\log n}\big)$, which is achieved by pre-computing products of all bit vectors of length $\log n$ with certain submatrices. An asymptotically more significant acceleration is obtained by using fast algorithms for multiplying $n \times n$ numerical matrices, such as Strassen's [11] algorithm that requires $O(n^{2.81})$ arithmetical operations, or the algorithm of Coppersmith and Winograd [3] with the theoretical running time $O(n^{2.376})$. These algorithms can be applied to multiplying $n \times n$ Boolean matrices by calculating their product in the ring of residues modulo $n+1$ [1].

Taking a closer look at Valiant's algorithm, one can see that first the entire grammar is encoded in a certain semiring, then the notion of a *transitive closure* of a Boolean matrix is extended to matrices over this semiring, so that the desired parsing table could be obtained as a closure of this kind, and finally it is demonstrated that such a closure can be efficiently computed using Boolean matrix multiplication. This approach essentially relies on having two operations in a grammar, concatenation and union, which give rise to the product and the sum in the semiring. Because of that, Valiant's algorithm as it is cannot be applied to Boolean grammars.

This paper aims at refactoring Valiant's algorithm to make it work in the more general case of Boolean grammars. It is shown that using matrices over a semiring as an intermediate abstraction is in fact unnecessary, and it is sufficient to employ matrix multiplication to compute the concatenations only, with the Boolean operations evaluated separately. Furthermore, the proposed algorithm maintains one fixed data structure, the parsing table, and whenever the matrix is to be cut as per Valiant's divide-and-conquer strategy, the new algorithm only distributes the ranges of positions in the input string among the recursive calls. This leads to an improved parsing algorithm, which, besides being applicable to a larger family of grammars, is also better understandable than Valiant's algorithm, has a succinct proof of correctness and is ready to be implemented.

2 Boolean Grammars

Let Σ be a finite nonempty set used as an *alphabet*, let Σ^* be the set of all finite strings over Σ. For a string $w = a_1 \ldots a_\ell \in \Sigma^*$ with $a_i \in \Sigma$, the *length* of the string is denoted by $|w| = \ell$. The unique *empty string* of length 0 is denoted by ε. For a string $w \in \Sigma^*$ and for every its partition $w = uv$, u is a *prefix* of w and v is its *suffix*; furthermore, for every partition $w = xyz$, the string y is a *substring* of w.

Any subset of Σ^* is a *language* over Σ. The basic operations on languages are the *concatenation* $K \cdot L = \{\, uv \mid u \in K,\ v \in L \,\}$ and the Boolean set operations:

union $K \cup L$, intersection $K \cap L$, and complementation $\overline{L}$. *Boolean grammars* are a family of formal grammars in which all these operations can be explicitly specified.

Definition 1. [7] *A Boolean grammar is a quadruple $G = (\Sigma, N, P, S)$, where Σ and N are disjoint finite non-empty sets of terminal and nonterminal symbols respectively; P is a finite set of rules of the form*

$$A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n, \tag{1}$$

where $m + n \geqslant 1$, $\alpha_i, \beta_i \in (\Sigma \cup N)^*$; $S \in N$ is the start symbol of the grammar.

If negation is not allowed, that is, $m \geqslant 1$ and $n = 0$ in every rule, the resulting grammars are known as *conjunctive grammars* [6]. If conjunction is also prohibited, and thus every rule must have $m = 1$ and $n = 0$, then the *context-free grammars* are obtained.

The intuitive semantics of a Boolean grammar is fairly clear: a rule (1) specifies that every string that satisfies each of the conditions α_i and none of the conditions β_i is therefore generated by A. However, formalizing this definition has proved to be rather nontrivial in the general case. In the case of conjunctive grammars (including the context-free grammars), the semantics can be equivalently defined by a least solution of language equations and by term rewriting. The definition by language equations carries on to Boolean grammars of the general form as follows.

A grammar is interpreted as a system of language equations in variables N, in which the equation for each $A \in N$ is

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i \ \cap \ \bigcap_{j=1}^{n} \overline{\beta_j} \right] \tag{2}$$

The vector $(\ldots, L_G(A), \ldots)$ of languages generated by the nonterminals of the grammar is defined by a solution of this system. In general, such a system may have no solutions (as in the equation $S = \overline{S}$ corresponding to the grammar $S \to \neg S$) or multiple solutions (with $S = S$ being the simplest example), but the below simplest definition of Boolean grammars dismisses such systems as ill-formed, and considers only systems with a unique solution; to be more precise, a subclass of such systems:

Definition 2. *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar, let (2) be the associated system of language equations. Suppose that for every number $\ell \geqslant 0$ there exists a unique vector of languages $(\ldots, L_C, \ldots)_{C \in N}$ ($L_C \subseteq \Sigma^{\leqslant \ell}$), such that a substitution of L_C for C, for each $C \in N$, turns every equation (2) into an equality modulo intersection with $\Sigma^{\leqslant \ell}$.*

Then G complies to the semantics of a strongly unique solution, and, for every $A \in N$, the language $L_G(A)$ can be defined as L_A from the unique solution of this system. The language generated by the grammar is $L(G) = L_G(S)$.

This fairly rough restriction ensures that the membership of a string in the language depends only on the membership of shorter strings, which is essential for the grammars to represent inductive definitions.

Example 1. The following Boolean grammar generates the language $\{ a^m b^n c^n \mid m, n \geqslant 0, m \neq n \}$:

$$\begin{aligned} S &\to AB \& \neg DC \\ A &\to aA \mid \varepsilon \\ B &\to bBc \mid \varepsilon \\ C &\to cC \mid \varepsilon \\ D &\to aDb \mid \varepsilon \end{aligned}$$

The rules for the nonterminals A, B, C and D are context-free, and they define $L_G(AB) = \{ a^i b^n c^n \mid i, n \geqslant 0 \}$ and $L_G(DC) = \{ a^m b^m c^j \mid j, m \geqslant 0 \}$. Then the propositional connectives in the rule for S specify the following combination of the conditions given by AB and DC:

$$L(AB) \cap \overline{L(DC)} = \{ a^i b^j c^k \mid j = k \text{ and } i \neq j \} = \underbrace{\{ a^m b^n c^n \mid m, n \geqslant 0, m \neq n \}}_{L(S)}$$

Assuming Definition 2, every Boolean grammar can be transformed to an equivalent grammar in *binary normal form* [7], in which every rule in P is of the form

$$\begin{aligned} &A \to B_1 C_1 \& \ldots \& B_n C_m \& \neg D_1 E_1 \& \ldots \& \neg D_n E_n \& \neg \varepsilon \\ &\qquad\qquad\qquad\qquad (m \geqslant 1,\ n \geqslant 0,\ B_i, C_i, D_j, E_j \in N) \\ &A \to a \\ &S \to \varepsilon \quad \text{(only if } S \text{ does not appear in right-hand sides of rules)} \end{aligned}$$

In the general case, the transformation requires an exponential blowup in the size of the grammar.

An alternative, more general definition of the semantics of Boolean grammars will be presented in Section 7.

3 Simple Cubic-Time Parsing

Let $G = (\Sigma, N, P, S)$ be a Boolean grammar in binary normal form, let $w = a_1 \ldots a_n$ be an input string. The simple cubic-time parsing algorithm constructs a table $T \in (2^N)^{n \times n}$, with

$$T_{i,j} = \{ A \in N \mid a_{i+1} \ldots a_j \in L_G(A) \}$$

for all $0 \leqslant i < j \leqslant n$. The elements of this table can be computed inductively on the length $j - i$ of the substring, starting with the elements $T_{i,i+1}$ each depending only on the symbol a_{i+1}, and continuing with larger and larger substrings, until the element $T_{0,n}$ is computed. The induction step is given by the equality

$$T_{i,j} = f\Big(\bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}\Big),$$

where the function $f\colon 2^{N\times N} \to 2^N$ is defined by

$$f(R) = \{A \mid \exists A \to B_1C_1\& \ldots \& B_mC_m \& \neg D_1E_1 \& \ldots \& \neg D_{m'}E_{m'} \in P : \\ (B_t, C_t) \in R \text{ and } (D_t, E_t) \notin R \text{ for all } t\}.$$

In total, there are $\Theta(n^2)$ elements, and each of them takes $\Theta(n)$ operations to compute, which results in a cubic time complexity.

The full algorithm can be stated as follows:

Algorithm 1 (Extended Cocke–Kasami–Younger [6,7]). *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar in the binary normal form. Let $w = a_1 \ldots a_n$, where $n \geqslant 1$ and $a_i \in \Sigma$, be an input string. For all $0 \leqslant i < j \leqslant n$, let $T_{i,j}$ be a variable ranging over subsets of N. Let R be a variable ranging over subsets of $N \times N$.*

```
1: for $i = 1$ to $n$ do
2:     $T_{i-1,i} = \{ A \mid A \to a_i \in P \}$
3: for $\ell = 2$ to $n$ do
4:     for $i = 0$ to $n - \ell$ do
5:         $R = \varnothing$
6:         for all $k = i + 1$ to $i + \ell - 1$ do
7:             $R = R \cup (T_{i,k} \times T_{k,i+\ell})$
8:         $T_{i,i+\ell} = f(R)$
9: accept if and only if $S \in T_{0,n}$
```

The most time-consuming operation in the algorithm is computing the unions $R_{i,j} = \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$, in which $R_{i,j}$ represents all concatenations BC that generate the substring $a_{i+1} \ldots a_j$ and the index k is a cutting point of this substring, with B generating $a_{i+1} \ldots a_k$ and with C generating $a_{k+1} \ldots a_j$. If each union is computed individually, as it is done in the above algorithm, then spending linear time for each $R_{i,j}$ is unavoidable. However, if such unions are computed for several sets $T_{i,j}$ at a time, much of the work can be represented as Boolean matrix multiplication. This is illustrated in the following example:

Example 2. Let $w = a_1a_2a_3a_4a_5$ be an input string and consider the partially constructed parsing table depicted in Figure 1, with $T_{i,j}$ constructed for $1 \leqslant i < j \leqslant 3$ and for $3 \leqslant i < j \leqslant 5$, that is, for the substrings $a_1a_2a_3$ and $a_3a_4a_5$ together with their substrings. Denote by $T^A_{i,j}$ the Boolean value indicating whether A is in $T_{i,j}$ or not. Then the following product of Boolean matrices

$$\begin{pmatrix} T^B_{0,2} & T^B_{0,3} \\ T^B_{1,2} & T^B_{1,3} \end{pmatrix} \times \begin{pmatrix} T^C_{2,4} & T^C_{2,5} \\ T^C_{3,4} & T^C_{3,5} \end{pmatrix} = \begin{pmatrix} X_{0,4} & X_{0,5} \\ X_{1,4} & X_{1,5} \end{pmatrix}$$

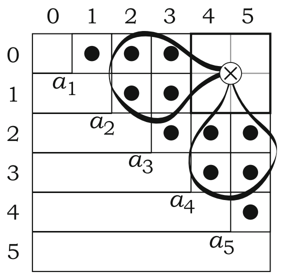


Fig. 1. Product of two Boolean matrices in Example 2

represents partial information on whether the pair (B, C) should be in the following four elements: $\begin{pmatrix} R_{0,4} & R_{0,5} \\ R_{1,4} & R_{1,5} \end{pmatrix}$. To be precise, $X_{1,4}$ computes the membership of (B, C) in $R_{1,4}$ exactly; $X_{0,4}$ does not take into account the factorization $a_1 \cdot a_2 a_3 a_4$, which actually requires knowing whether C is in $T_{1,4}$; the element $X_{1,5}$ is symmetrically incomplete; finally, $X_{0,5}$ misses the factorizations $a_1 \cdot a_2 a_3 a_4 a_5$ and $a_1 a_2 a_3 a_4 \cdot a_5$, which can be properly obtained only using $T_{0,4}$ and $T_{1,5}$. In total, this matrix product computes 8 conjunctions out of 12 needed for these four elements of R.

Already in this small example, using one matrix product requires changing the order of computation of the elements $\{T_{i,j}\}$: the elements $T_{0,3}$ and $T_{2,5}$ need to be calculated before $T_{1,4}$. In the next section, the whole algorithm will be restated as a recursive procedure, which arranges the computation so that as much work as possible is offloaded into products of the largest possible matrices.

4 Parsing Reduced to Matrix Multiplication

Let $w = a_1 \ldots a_n$ be an input string. For the time being, assume that $n + 1$ is a power of two, that is, the length of the input string is a power of two minus one; this restriction can be relaxed in an implementation, which will be discussed in the next section.

The algorithm uses the following data structures. First, there is an $(n + 1) \times (n + 1)$ table T with $T_{i,j} \subseteq N$, as in Algorithm 1, and the goal is to set each entry to $T_{i,j} = \{\, A \mid a_{i+1} \ldots a_j \in L(A) \,\}$ for all $0 \leqslant i < j \leqslant n$. The second table R has elements $R_{i,j} \subseteq N \times N$ each corresponding to the value of R computed by Algorithm 1 in the iteration $(\ell = j - i,\ i)$. The target value is $R_{i,j} = \{\, (B, C) \mid a_{i+1} \ldots a_j \in L(B)L(C) \,\}$ for all $0 \leqslant i < j \leqslant n$.

Initially, the elements of the tables are set as follows: $T_{i-1,i} = \{\, A \mid A \to a_i \in P \,\}$ for all $1 \leqslant i \leqslant n$, and the rest of values of T are undefined; $R_{i,j} = \varnothing$. The rest of the entries are gradually constructed using the following two recursive procedures:

- The first procedure, *compute*(ℓ, m), constructs the correct values of $T_{i,j}$ for all $\ell \leqslant i < j < m$.

– The other procedure, *complete*(ℓ, m, ℓ', m'), assumes that the elements $T_{i,j}$ are already constructed for all i and j with $\ell \leqslant i < j < m$, as well as for all i, j with $\ell' \leqslant i < j < m'$; it is furthermore assumed that for all $\ell \leqslant i < m$ and $\ell' \leqslant j < m'$, the current value of $R_{i,j}$ is

$$R_{i,j} = \{ (B, C) \mid \exists k \, (m \leqslant k < \ell') : \, a_{i+1} \ldots a_k \in L(B), \, a_{k+1} \ldots a_j \in L(C) \},$$

which is a subset of the intended value of $R_{i,j}$.
Then *complete*(ℓ, m, ℓ', m') constructs $T_{i,j}$ for all $\ell \leqslant i < m$ and $\ell' \leqslant j < m'$.

– Matrix multiplication is performed by one more procedure, *product*(d, ℓ, ℓ', ℓ''), whose task is to add to each $R_{i,j}$, with $\ell \leqslant i < \ell + d$, and $\ell'' \leqslant j < \ell'' + d$, all such pairs (B, C), that $B \in T_{i,k}$ and $C \in T_{k,j}$ for some k with $\ell' \leqslant k < \ell' + d$. This can generally be done by computing $|N|^2$ products of $d \times d$ Boolean matrices, one for each pair (B, C).

Algorithm 2 (Parsing through matrix multiplication)

```
Main procedure:
 1: for $i = 1$ to $n$ do
 2:     $T_{i-1,i} = \{ A \mid A \to a_i \in P \}$
 3: compute$(0, n+1)$
 4: Accept if and only if $S \in T_{0,n}$
Procedure compute$(\ell, m)$:
 5: if $m - \ell > 4$ then
 6:     compute$(\ell, \frac{\ell+m}{2})$
 7:     compute$(\frac{\ell+m}{2}, m)$
 8: complete$(\ell, \frac{\ell+m}{2}, \frac{\ell+m}{2}, m)$
Procedure complete$(\ell, m, \ell', m')$, which requires $m - \ell = m' - \ell'$:
 9: if $m - \ell > 1$ then                                        /* see Figure 2 */
10:     /* compute $\mathcal{C}$ */
11:     complete$(\frac{\ell+m}{2}, m, \ell', \frac{\ell'+m'}{2})$
12:     /* compute $\mathcal{D}_1$ */
13:     product$(\frac{m-\ell}{2}, \ell, \frac{\ell+m}{2}, \ell')$              /* $\mathcal{D}_1 \leftarrow \mathcal{B}_1 \times \mathcal{C}$ */
14:     complete$(\ell, \frac{\ell+m}{2}, \ell', \frac{\ell'+m'}{2})$
15:     /* compute $\mathcal{D}_2$ */
16:     product$(\frac{m-\ell}{2}, \frac{\ell+m}{2}, \ell', \frac{\ell'+m'}{2})$        /* $\mathcal{D}_2 \leftarrow \mathcal{C} \times \mathcal{B}_2$ */
17:     complete$(\frac{\ell+m}{2}, m, \frac{\ell'+m'}{2}, m')$
18:     /* compute $\mathcal{E}$ */
19:     product$(\frac{m-\ell}{2}, \ell, \frac{\ell+m}{2}, \frac{\ell'+m'}{2})$        /* $\mathcal{E} \leftarrow \mathcal{B}_1 \times \mathcal{D}_2$ */
20:     product$(\frac{m-\ell}{2}, \ell, \ell', \frac{\ell'+m'}{2})$              /* $\mathcal{E} \leftarrow \mathcal{D}_1 \times \mathcal{B}_2$ */
21:     complete$(\ell, \frac{\ell+m}{2}, \frac{\ell'+m'}{2}, m')$
22: else if $m \neq \ell'$ then
23:     $T_{\ell,\ell'} = f(R_{\ell,\ell'})$
```

The partition of the matrix in *complete*() is illustrated in Figure 2.

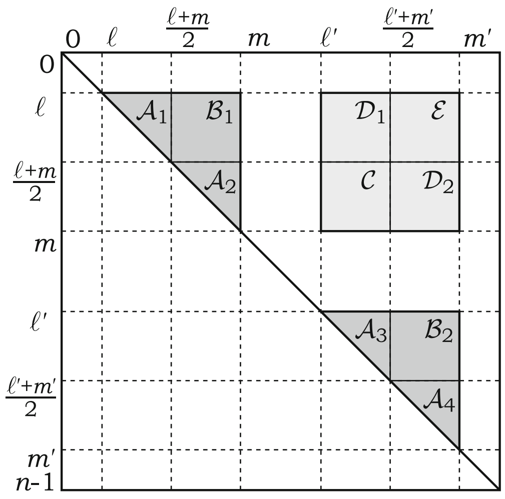


Fig. 2. Matrix partition in *complete*(ℓ, m, ℓ', m')

Lemma 1. *Let $\ell < m \leqslant \ell' < m'$ with $m - \ell = m' - \ell'$ being a power of two, and assume that $T_{i,j} = \{ A \mid a_{i+1} \ldots a_j \in L(A) \}$ for all i and j with $\ell \leqslant i < j < m$, as well as for all i, j with $\ell' \leqslant i < j < m'$. Furthermore, assume that, for all $\ell \leqslant i < m$ and $\ell' \leqslant j < m'$,*

$$R_{i,j} = \{ (B, C) \mid \exists k \, (m \leqslant k < \ell') : \; a_{i+1} \ldots a_k \in L(B), \; a_{k+1} \ldots a_j \in L(C) \}.$$

Then complete(ℓ, m, ℓ', m') returns with $T_{i,j} = \{ A \mid a_{i+1} \ldots a_j \in L(A) \}$ for all $\ell \leqslant i < m$ and $\ell' \leqslant j < m'$.

Lemma 2. *The procedure compute(ℓ, m), executed on ℓ and m with $m-\ell$ being a power of two, returns with $T_{i,j} = \{ A \mid a_{i+1} \ldots a_j \in L(A) \}$ for all $\ell \leqslant i < j < m$.*

In order to determine the algorithm's complexity on an input of length $2^k - 1$, consider how many times the procedures *compute*() and *complete*() are called for subproblems of each size. For each $i \in \{1, \ldots, k-1\}$, *compute*(ℓ, m) with $m - \ell = 2^{k-i}$ is called exactly 2^i times; *complete*(ℓ, m, ℓ', m') with $m - \ell = 2^{k-i}$ is called exactly $2^{2i-1} - 2^{i-1}$ times; *product*() is called for $2^{k-i} \times 2^{k-i}$ submatrices exactly $2^{2i-1} - 2^i$ times, and multiplies $O(|G|)$ pairs of Boolean matrices.

Theorem 1. *For every Boolean grammar G in binary normal form, Algorithm 2 constructs the parsing table for a string of length n in time $O(|G| \cdot BMM(n) \log n)$, where $BMM(n)$ is the time needed to multiply two $n \times n$ Boolean matrices. Assuming $BMM(n) = \Omega(n^{2+\varepsilon})$, the complexity is $\Theta(|G| \cdot BMM(n))$.*

5 Notes on Implementation

The restriction on the length of the string being a power of two minus one is convenient for the algorithm's presentation, but it would be rather annoying for any implementation. This essential condition can be circumvented as follows. Let $w = a_1 \dots a_n$ be an input string of any length $n \geqslant 1$. The algorithm shall construct a table of size $(n+1) \times (n+1)$, yet while doing so, it will imagine a larger table of size rounded up to the next power of two. Whenever a subroutine call is concerned entirely with the elements beyond the edge of the table, this call is skipped. The matrix products with one of the matrices split by the edge are changed to products of rectangular matrices fitting into the table. More details shall be presented in the full version of the paper.

Another question concerns the possible data structures for the algorithm. In general, not everything mentioned in the theoretical presentation of the algorithm would need to be computed for an actual grammar. First assume that the grammar is context-free. In this case, whenever a pair (B, C) is added to $R_{i,j}$, it will eventually make all nonterminals A with a rule $A \to BC$ be added to $T_{i,j}$. Accordingly, the data structure R is not needed, and all matrix multiplication procedures can output their result directly into the appropriate elements of T.

If the grammar is conjunctive or Boolean, there is a genuine need for using R, yet only for the rules involving multiple conjuncts. Simple context-free rules with a unique conjunct can be treated in the simplified way described above, with all matrix products being directly flushed into T. If there exists a rule $A \to BC \& \dots$ with at least two conjuncts, or any rule $A \to \neg BC \& \dots$, then all data about the pair (B, C) needs to be stored in R as described in the algorithm. This data shall be used in the calculation of f, which takes into account the complex rules.

With this optimization of the algorithm, the following data structures naturally come to mind:

- For each nonterminal $A \in N$, an $(n+1) \times (n+1)$ upper-triangular Boolean matrix T^A, with $T^A_{i,j}$ representing the membership of A in the set $T_{i,j}$. All matrix products computed in the algorithm shall have some submatrices of this matrix as the arguments.
- For every such pair $(B, C) \in N \times N$ that occurs in multiple-conjunct rules $A \to BC \& \dots$ or is negated in any rule $A \to \neg BC \& \dots$, the algorithm shall maintain an $(n+1) \times (n+1)$ upper-triangular Boolean matrix R^{BC}.

6 Generalized Algorithm

The original Valiant's algorithm was presented in a generalized form, in which it computes a certain kind of closure of a matrix over a semiring. While the updated algorithm no longer uses any semiring, its computation can also be generalized to operations over abstract structures.

Let X and Y be two sets, let $\circ : X \times X \to Y$ be a binary operator mapping pairs of elements of X to elements of Y, let $\sqcup : Y \times Y \to Y$ be an associative and commutative binary operator on Y, and let $f : Y \to X$ be any function. Let $x = x_1 \dots x_n$ with $x_i \in X$ be a sequence of elements of X and consider the matrix $T = T(x) \in X^{n \times n}$ defined by the following equations:

$$T_{i-1,i} = x_i$$
$$T_{i,j} = f\Big(\bigsqcup_{k=i+1}^{j-1} T_{i,k} \circ T_{k,j}\Big)$$

Theorem 2. *There is an algorithm, which, given a string $x = x_1 \ldots x_n$ of length n, computes the matrix $T(x)$ in time $O(BMM(n) \log n)$.*

In this generalized form, the algorithm can be applied to different families of grammars. For example, for context-free grammars in the binary normal form one can set $X = 2^N$, $Y = 2^{N \times N}$, $\circ = \times$, $\sqcup = \cup$, $x_i = \{ A \in N \mid A \to a_i \in P \}$ and $f(y) = \{ A \in N \mid \exists A \to BC \in P : (B, C) \in y \}$. For Boolean grammars, the only difference is in f, which has to take into account more complicated Boolean logic in the rules.

The same extended algorithm can be applied to probabilistic context-free grammars, as well as to the fuzzy generalization of Boolean grammars defined by Ésik and Kuich [4]. The next section presents one more application.

7 Application to the Well-Founded Semantics

The *well-founded semantics* of Boolean grammars was proposed by Kountouriotis, Nomikos and Rondogiannis [5]. This semantics is applicable to every syntactically valid Boolean grammar, and defines a *three-valued language* generated by each nonterminal symbol.

Three-valued languages are mappings from Σ^* to $\{0, \frac{1}{2}, 1\}$, where 1 and 0 indicate that a string definitely is or definitely is not in the language, while $\frac{1}{2}$ stands for "undefined". Equivalently, three-valued languages can be defined by pairs (L, L') with $L \subseteq L' \subseteq \Sigma^*$, where L and L' represent a lower bound and an upper bound on a language that is not known precisely. A string in both L and L' definitely is in the language, a string belonging to neither of them definitely is not, and if a string is in L' but not in L, its membership is not defined. In particular, if $L = L'$, then the language is completely defined, and a pair $(\varnothing, \Sigma^*)$ means a language about which nothing is known. The set of such pairs shall be denoted by 3^{Σ^*}.

Boolean operations and concatenation are generalized from two-valued to three-valued languages as follows:

$$(K, K') \cup (L, L') = (K \cup L, K' \cup L')$$
$$(K, K') \cap (L, L') = (K \cap L, K' \cap L')$$
$$\overline{(L, L')} = (\overline{L'}, \overline{L})$$
$$(K, K')(L, L') = (KL, K'L')$$

Two different partial orderings on three-valued languages are defined. First, they can be compared with respect to the *degree of truth*:

$$(K, K') \sqsubseteq_T (L, L') \quad \text{if} \quad K \subseteq L \text{ and } K' \subseteq L'.$$

The other ordering is with respect to the *degree of information*:

$$(K, K') \sqsubseteq_I (L, L') \quad \text{if} \quad K \subseteq L \text{ and } L' \subseteq K'.$$

It represents the fact that (K, K') and (L, L') are approximations of the same language, and that (L, L') is more precise, in the sense of having fewer uncertain strings.

Both orderings are extended to vectors of three-valued languages. The truth-ordering has a bottom element $\bot_T = \big((\varnothing, \varnothing), \ldots, (\varnothing, \varnothing)\big)$, For the information-ordering, the bottom element is $\bot_I = \big((\varnothing, \Sigma^*), \ldots, (\varnothing, \Sigma^*)\big)$.

As in the two-valued case, concatenation, union and intersection, as well as every combination thereof, are monotone and continuous with respect to the truth ordering; complementation is not monotone. With respect to the information ordering; concatenation and all Boolean operations, *including complementation*, are monotone and continuous, which extends to any combinations of these operations.

Definition 3 (Well-founded semantics [5]). *Let $G = (\Sigma, N, P, S)$ be a Boolean grammar, let $N = \{A_1, \ldots, A_n\}$. Fix any vector $K = ((K_1, K_1'), \ldots, (K_n, K_n')) \in (3^{\Sigma^*})^n$ and define a function $\Theta_K : (3^{\Sigma^*})^n \to (3^{\Sigma^*})^n$ by substituting its argument into positive conjuncts and K into negative conjuncts:*

$$[\Theta_K(L)]_A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i(L) \ \cap \ \bigcap_{j=1}^{n} \overline{\beta_j(K)} \right],$$

for each $A \in N$. Define $\Omega(K) = \bigsqcup^{T}_{\ell \geqslant 0} \Theta_K^{\ell}(\bot_T)$ and let $M = \bigsqcup^{I}_{k \geqslant 0} \Omega^k(\bot_I)$. Then, according to the well-founded semantics of Boolean grammars, $L_G(A) = [M]_A$.

The main result justifying the correctness of the well-founded semantics, is that M is a solution of the following system of equations in three-valued languages:

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \& \neg\beta_1 \& \ldots \& \neg\beta_n \in P} \left[\bigcap_{i=1}^{m} \alpha_i(L) \ \cap \ \bigcap_{j=1}^{n} \overline{\beta_j(L)} \right] \quad (\text{for } A \in N).$$

The binary normal form is generalized to the well-founded semantics:

Proposition 1 (Kountouriotis et al. [5]). *Every Boolean grammar, as in Definition 3, can be effectively transformed to a grammar in the binary normal form, in which every rule is of the form*

$$\begin{aligned} &A \to B_1C_1 \& \ldots \& B_nC_m \& \neg D_1E_1 \& \ldots \& \neg D_nE_n \& \neg\varepsilon \\ &\qquad\qquad\qquad\qquad (m \geqslant 1,\ n \geqslant 0,\ B_i, C_i, D_j, E_j \in N) \\ &A \to a \quad (a \in \Sigma) \\ &A \to a \& U \quad (a \in \Sigma) \\ &U \to \neg U \quad (\text{a special symbol generating uncertainty}) \\ &S \to \varepsilon \quad (\text{only if } S \text{ does not appear in right-hand sides of rules}) \end{aligned}$$

The transformation maintains the generated three-valued language.

Kountouriotis et al. [5] used this normal form to construct an extension of the cubic-time parsing algorithm to the well-founded semantics, which, given an input string w, computes its membership status as a value in $\{0, \frac{1}{2}, 1\}$. The data constructed in that algorithm can be computed more efficiently using matrix multiplication, which will now be demonstrated by encoding it into the abstract form of the proposed algorithm. Let $X = 3^N$, $Y = 3^{N \times N}$, $(U_1, V_1) \circ (U_2, V_2) = (U_1 \times U_2, V_1 \times V_2)$, $(Q_1, R_1) \sqcup (Q_2, R_2) = (Q_1 \cup Q_2, R_1 \cup R_2)$, $I(a) = (\{\, A \mid A \to a \in P \,\}, \{\, A \mid A \to a \in P \text{ or } A \to a \& U \in P \,\})$, and finally $f(Q, R) = \big(\{A \mid \exists A \to B_1C_1 \& \ldots \& B_mC_m \& \neg D_1E_1 \& \ldots \& \neg D_{m'}E_{m'} \& \neg \varepsilon : (B_i, C_i) \in Q \text{ and } (D_j, E_j) \notin R \text{ for all applicable } i, j\}, \{A \mid \exists A \to B_1C_1 \& \ldots \& B_mC_m \& \neg D_1E_1 \& \ldots \& \neg D_{m'}E_{m'} \& \neg \varepsilon : (B_i, C_i) \in R \text{ and } (D_j, E_j) \notin Q \text{ for all applicable } i, j\}\big)$. This establishes an analogue of Theorem 1 for the well-founded semantics, that is, the three-valued membership in $L(G)$ of a given string $w \in \Sigma^*$ can be computed in time $\Theta(BMM(n) \log n) = O(n^{2.376})$.

Thus, one more key algorithm for the context-free grammars has been extended to the general case of Boolean grammars, and its clarity has even been improved in the process. This provides further evidence for the author's long-time claim that Boolean grammars are the proper general case of the context-free grammars.

References

1. Adleman, L., Booth, K.S., Preparata, F.P., Ruzzo, W.L.: Improved time and space bounds for Boolean matrix multiplication. Acta Informatica 11(1), 61–70 (1978)
2. Arlazarov, V.L., Dinic, E.A., Kronrod, M.A., Faradzhev, I.A.: On economical construction of the transitive closure of an oriented graph. Soviet Mathematics Doklady 11, 1209–1210 (1970)
3. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation 9(3), 251–280 (1990)
4. Ésik, Z., Kuich, W.: Boolean fuzzy sets. International Journal of Foundations of Computer Science 18(6), 1197–1207 (2007)
5. Kountouriotis, V., Nomikos, C., Rondogiannis, P.: Well-founded semantics for Boolean grammars. Information and Computation 207(9), 945–967 (2009)
6. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics 6(4), 519–535 (2001)
7. Okhotin, A.: Boolean grammars. Information and Computation 194(1), 19–48 (2004)
8. Okhotin, A.: Generalized LR parsing algorithm for Boolean grammars. International Journal of Foundations of Computer Science 17(3), 629–664 (2006)
9. Okhotin, A.: Recursive descent parsing for Boolean grammars. Acta Informatica 44(3-4), 167–189 (2007)
10. Okhotin, A.: Unambiguous Boolean grammars. Information and Computation 206, 1234–1247 (2008)
11. Strassen, V.: Gaussian elimination is not optimal. Numerische Mathematik 13, 354–356 (1969)
12. Valiant, L.G.: General context-free recognition in less than cubic time. Journal of Computer and System Sciences 10(2), 308–314 (1975)

On Lexicalized Well-Behaved Restarting Automata That Are Monotone*

Friedrich Otto[1], Martin Plátek[2], and František Mráz[2]

[1] Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
otto@theory.informatik.uni-kassel.de

[2] Charles University, Faculty of Mathematics and Physics
Department of Computer Science, Malostranské nám. 25
118 00 Praha 1, Czech Republic
mraz@ksvi.ms.mff.cuni.cz, Martin.Platek@mff.cuni.cz

Abstract. We introduce lexicalized well-behaved restarting automata as a model of the gradual lexicalized syntactic disambiguation of natural languages. This model presents a non-correctness preserving counter-part to the (correctness preserving) models of analysis by reduction of natural languages. We study two types of gradual relaxations of the correctness preserving property for monotone automata of this type. They lead to two infinite hierarchies of language classes. The basic levels of these hierarchies coincide with the class LRR of left-to-right regular languages, and the hierarchies exhaust the class of context-free languages.

1 Introduction

The restarting automaton was introduced in [2] to model the so-called *analysis by reduction* of natural languages. To each sentence of the language considered, a restarting automaton associates all possible derivations that are obtained through sequences of reduction steps. These reduction steps satisfy the so-called 'error preserving property!.' This is an important property that imitates a similar property of analytical grammars. It states that any cycle of any computation of a restarting automaton M that starts from a word not belonging to the characteristic language $L_C(M)$ accepted by M necessarily yields a word that does not belong to this language, either. On the other hand, it is only deterministic restarting automata that in general also satisfy the complementary property of being *correctness preserving*, which states that any cycle of M that starts from a word belonging to the language $L_C(M)$ will again give a word from that language. A nondeterministic restarting automaton is called *correctness preserving* if it satisfies the correctness preserving property.

A restarting automaton as defined in [2] begins its computations with pure input words, while analysis by reduction is performed in a correctness preserving

* F. Mráz and M. Plátek were supported by the grants P202/10/1333 and P103/10/0783 of GAČR.

way on sentences that have already been enriched unambiguously by lexical categories (that is, nonterminal symbols). In order to bridge this gap we introduce the so-called *lexicalized well-behaved restarting automaton* to formalize the procedure of gradual disambiguation of sentences of a natural language by lexical (that is, morphological, syntactic, or semantic) categories.

As the procedure of lexical disambiguation is highly ambiguous, it cannot generally be modelled by a correctness preserving automaton. Accordingly, we consider two gradual relaxations of this property, the *phase relaxation* and the *error relaxation*. The former is a variant of the *cyclic relaxation*, which was introduced together with the error relaxation by the current authors in [6]. Both these relaxations can be interpreted as measuring the degree of nondeterminism of a restarting automaton (or a characteristic language) that is responsible for erroneous computations. In contrast to many studies of the degree of nondeterminism for various types of automata (see, e.g., [13]), we are not interested in nondeterministic steps that distinguish between different accepting computations. In [6] an infinite two-dimensional hierarchy of classes of restarting automata and their characteristic languages was derived. Here we carry these investigations over to *proper languages* of lexicalized well-behaved restarting automata. The proper language is obtained from the characteristic language by simply deleting all auxiliary symbols, and a restarting automaton is called *well-behaved*, if its proper language coincides with its input language. This requirement places serious restrictions on the way in which such a restarting automaton can use its auxiliary symbols for lexical disambiguation, restrictions that are well motivated by the way in which categories are used in European linguistics. It is expected that the categories obtained in this way correspond to those that are derived by the analysis by reduction as used in the development of the well-known 'Functional (Generative) Description (of Czech)' (see [4] and some previous work of P. Sgall). We establish two infinite hierarchies of language classes of lexicalized well-behaved restarting automata that are monotone. The basic levels of these hierarchies coincide with the class LRR of left-to-right regular languages [1], and they exhaust the class of context-free languages.

This paper is structured as follows. After giving some basic definitions in Section 2, we define the notion of lexicalized well-behaved restarting automaton in Section 3, and we prove that all context-free languages are recognized by monotone automata of this type. In Section 4 we define the notions of *phase relaxation* of degree i and of *error relaxation* of degree j, and we establish the announced hierarchies based on the degree of phase relaxation and the degree of error relaxation. In the concluding section we shortly discuss the corresponding hierarchies for lexicalized well-behaved restarting automata that are non-monotone. In this paper proofs will mostly only be outlined to meet the page limit.

2 Basic Definitions and Notation

Here we describe in short the type of restarting automaton we will be dealing with. More details on restarting automata in general can be found in [9,10].

A *two-way restarting automaton*, RLWW-automaton for short, is a nondeterministic machine $M = (Q, \Sigma, \Gamma, \cent, \$, q_0, k, \delta)$ with a finite set Q of (internal) states, a flexible tape, and a read/write window of a fixed size $k \geq 1$. The work space is limited by the left sentinel ¢ and the right sentinel \$, which cannot be removed from the tape. The tape alphabet Γ contains the input alphabet Σ and possibly a finite number of so-called auxiliary symbols. The behaviour of M is described by the transition relation δ that associates a finite set of transition steps to each pair (q, u) consisting of a state q and a possible content u of the read/write window. There are five types of transition steps: *move-right* (MVR) and *move-left steps* (MVL), which shift the window one position to the right or to the left, respectively, and change the internal state, *rewrite steps* that replace the content u of the window by a shorter word v, thereby also shortening the tape, and change the internal state, *restart steps* (Restart) that place the window over the left end of the tape, and reset the internal state to the initial state q_0, and *accept steps* that cause M to halt and accept.

If $\delta(q, u) = \emptyset$ for some pair (q, u), then M necessarily halts, and we say that M *rejects* in this situation. Further, it is required that, when ignoring move operations, rewrite and restart steps alternate in each computation of M, with a rewrite step coming first. In general, the automaton M is *nondeterministic*, that is, there can be two or more instructions with the same left-hand side (q, u). If that is not the case, the automaton is *deterministic*.

A *configuration* of M is a string $\alpha q \beta$ where q is a state, and either $\alpha = \lambda$ (the empty string) and $\beta \in \{\cent\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{\cent\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here q represents the current state, $\alpha\beta$ is the current content of the tape, and it is understood that the window contains the first k symbols of β or all of β when $|\beta| \leq k$. A *restarting configuration* is of the form $q_0 \cent w \$$.

Any finite computation of M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration. The window is moved along the tape by MVR and MVL operations and a single rewrite operation until a restart operation is performed and thus a new restarting configuration is reached. The part after the last restart operation is called a *tail*. By $u \vdash_M^c v$ we denote a cycle of M that transforms the restarting configuration $q_0 \cent u \$$ into the restarting configuration $q_0 \cent v \$$. By $\vdash_M^{c^*}$ we denote the reflexive and transitive closure of $\vdash_M^c$.

A *word* $w \in \Gamma^*$ *is accepted by* M, if there is a computation which, starting with the restarting configuration $q_0 \cent w \$$, finishes by executing an accept instruction. By $L_C(M)$ we denote the language consisting of all words accepted by M; this is the *characteristic language* accepted (or recognized) by M. When we restrict attention to input words only, then we obtain the language $L(M) = L_C(M) \cap \Sigma^*$, which is the *input language* recognized (accepted) by M. Finally, the *proper language* $L_P(M)$ is obtained from the characteristic language $L_C(M)$ by deleting all auxiliary symbols, that is, $L_P(M) = \mathsf{Pr}^\Sigma(L_C(M))$, where $\mathsf{Pr}^\Sigma : \Gamma^* \to \Sigma^*$ is the morphism defined by $a \mapsto a$ $(a \in \Sigma)$ and $A \mapsto \lambda$ $(A \in \Gamma \smallsetminus \Sigma)$. Obviously, $L(M)$ is always a subset of $L_P(M)$.

If $u \notin L_C(M)$ and $u \vdash_M^{c^*} v$, then $v \notin L_C(M)$, either. This fact is the so-called *error preserving property* of restarting automata (see, e.g, [2]). However, it is only deterministic restarting automata that in general also satisfy the following complementary property.

Definition 1. (Correctness Preserving Property)
An RLWW*-automaton* M *is* (strongly) correctness preserving *if* $u \in L_{\mathrm{C}}(M)$ *and* $u \vdash_M^{c^*} v$ *imply that* $v \in L_{\mathrm{C}}(M)$*.*

We are also interested in various restricted types of restarting automata. An RLWW-automaton is called an RRWW-*automaton*, if it does not use any move-left operations, and an RLWW- (or RRWW-) automaton is called an RL- (or RR-) *automaton*, if it has no auxiliary symbols (that is, $\Gamma = \Sigma$), and if each rewrite step is simply a deletion. For each type X of restarting automaton, we use $\mathcal{L}(\mathsf{X})$ to denote the class of input languages that are accepted by restarting automata of type X.

For an RRWW-automaton M, the relation $\vdash_M^c$ can be described more transparently by a finite sequence of *meta-instructions* of the form $(E_l, u \to v, E_r)$, where E_l, E_r are regular languages, and $u \to v$ is a rewrite step of M (see, e.g., [10]). On trying to execute this meta-instruction, M will get stuck (and so reject) starting from the configuration $q_0 ¢ w \$$, if w does not admit a factorization of the form $w = w_1 u w_2$ such that $¢ w_1 \in E_l$, and $w_2 \$ \in E_r$. On the other hand, if w does have factorizations of this form, then one such factorization is chosen nondeterministically, and $q_0 ¢ w \$$ is transformed into $q_0 ¢ w_1 v w_2 \$$. In order to describe the tails of accepting computations we use meta-instructions of the form $(¢ \cdot E \cdot \$, \mathsf{Accept})$, which accept the strings from the regular language E.

Example 1. Let $L_{pal} := \{ ww^R \mid w \in \{a,b\}^* \}$, and let M_1 be the RRWW-automaton that is given by the following meta-instructions, where $x \in \{a,b\}$:

(1) $(¢ \cdot \{a,b\}^*, xx \to C, \{a,b\}^* \cdot \$)$, (3) $(¢ \cdot \{\lambda, C\} \cdot \$, \mathsf{Accept})$.
(2) $(¢ \cdot \{a,b\}^*, xCx \to C, \{a,b\}^* \cdot \$)$,

Then $L_{\mathrm{C}}(M_1) = L_{pal} \cup \{ wCw^R \mid w \in \{a,b\}^* \}$, and $L(M_1) = L_{pal} = L_{\mathrm{P}}(M_1)$.

Each cycle C of a computation of an RLWW-automaton M contains a unique configuration of the form $\alpha q \beta$ in which a rewrite step is executed. By $D_r(C)$ we denote the *right distance* $|\beta|$ of this cycle. A sequence of cycles $C_1, C_2, \ldots, C_n$ is called *monotone* if $D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n)$ holds, and a computation of M is called *monotone* if the corresponding sequence of cycles is monotone. Observe that the tail of the computation is not taken into account here. An RLWW-automaton is called *monotone* if each of its computations is monotone. We use the prefix mon- to denote this property. Observe that the RRWW-automaton M_1 in the above example is monotone.

3 Lexicalized Well-Behaved RLWW-Automata

We are interested in restarting automata which may use auxiliary symbols only in a rather restricted manner.

Definition 2. *An* RLWW*-automaton* $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ *is called*

- lexicalized *if, for each word* $w \in L_C(M)$ *and each factorization* $w = xyv$, $|y| \geq k$ *implies that* y *contains at least one input symbol;*
- well-behaved *if* $L(M) = L_P(M)$.

We use the prefix wbl- to denote classes of restarting automata that are lexicalized and well-behaved.

Theorem 1. *For each context-free language* L*, there exists a lexicalized well-behaved* RRWW*-automaton* M *such that* M *is monotone and* $L = L(M)$.

As monotone RLWW-automata accept only context-free languages (see, e.g., [10]), this result yields the following characterization.

Corollary 1. CFL $= \mathcal{L}$(mon-wbl-RRWW) $= \mathcal{L}$(mon-wbl-RLWW).

Proof of Theorem 1. Let $L \subseteq \Sigma^*$ be a context-free language, and let $G = (V, \Sigma, S, P)$ be a context-free grammar in Chomsky normal form for L. From G we construct an RRWW-automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ with the required properties. Its window size k will be fairly large, and accordingly it will simply accept all words from L of length at most $k - 2$. Thus, we can assume that all productions of G are of the form $(A \to BC)$ or $(A \to a)$, where $A, B, C \in V$ and $a \in \Sigma$, and that G does not contain any superfluous nonterminals, that is, the language $L(G, A) = \{ w \in \Sigma^+ \mid A \Rightarrow_G^+ w \}$ is nonempty for all nonterminals A. For each $A \in V$, let $w_A \in L(G, A)$ be a shortest word from $L(G, A)$. Further, let $\hat{L}(G, A) = \{ \alpha \in (V \cup \Sigma)^+ \mid A \Rightarrow_G^* \alpha \}$.

For the window size of M we take $k := 3 \cdot 2^{3m}$, where m be the number of nonterminals of G. The idea underlying the behaviour of M is the following. If $w \in L$, $|w| > k$, then there exists a factorization $w = uxv$ and a nonterminal A such that $S \Rightarrow_G^* uAv$ and $x \in L(G, A)$ satisfying $|x| > |w_A| + 2$. Thus, M can replace the factor x of w by the word w_A, in this way shortening the word. However, it must be ensured that there really is a G-derivation of the form $S \Rightarrow_G^* uAv$. Therefore, we need to store the information on A within the new word uw_Av. This is achieved as follows.

The tape alphabet Γ of M contains the input alphabet Σ and the auxiliary letters $\Gamma_V := \{ (_A,)_A \mid A \in V \}$. The above transformation is realized by replacing the factor x by the word $(_A w_A)_A$ of length $|w_A| + 2 < |x|$. Observe that $\mathsf{Pr}^{\Sigma}(u(_A w_A)_A v) = uw_Av \in L$, if $S \Rightarrow_G^* uAv$ holds.

In the general case the tape already contains some auxiliary letters. However, it will be ensured that it always satisfies the following invariant, where F denotes the finite language $F := \{ (_A w_A)_A \mid A \in V \}$:

$$\forall w \in \Sigma^* \, \forall \alpha \in \Gamma^* : \text{ if } w \vdash_M^{c^*} \alpha, \text{ then } \alpha \in \Sigma^* \cdot (F \cdot \Sigma^*)^* .$$

We define a transformation $\iota : \Gamma^* \to (V \cup \Sigma)^*$ by taking, for all $n \geq 0$, $u_0, \ldots, u_n \in \Sigma^*$, and $A_1, \ldots, A_n \in V$,

$$\iota(u_0 (_{A_1} w_{A_1})_{A_1} u_1 \ldots u_{n-1} (_{A_n} w_{A_n})_{A_n} u_n) = u_0 A_1 u_1 \ldots u_{n-1} A_n u_n. \qquad (1)$$

The word $\iota(w)$ is the sentential form associated with w. If w is not of the form required by (1), then $\iota(w)$ is undefined. Clearly ι is a rational relation.

Given a word $\alpha \in \Gamma^*$ of length $|\alpha| > 3 \cdot 2^{3m}$, M looks for a factorization $\alpha = uxv$ such that all of the following conditions are satisfied simultaneously:

(a) $u \in \Sigma^* \cdot (F \cdot \Sigma^*)^*$,
(b) $x \in \Sigma^* \cdot (F \cdot \Sigma^*)^*$,
(c) $\iota(x) \in \hat{L}(G, A)$ and $|w_A| + 2 < |\mathsf{Pr}^{\Sigma}(x)| \leq 2^{3m}$ for some $A \in V$,
(d) $v \in \Sigma^*$.

Conditions (b) and (c) together imply that $|x| \leq 3 \cdot 2^{3m} = k$. Observe that properties (a) to (c) can be checked internally, while the read/write window of M moves across the prefix ux of α.

Whenever M finds a factor x satisfying conditions (a) to (c), then it nondeterministically chooses either to just continue moving to the right, or it replaces the factor x by the word $(_A w_A)_A$, thereby shortening the tape contents. In the latter case it then scans the remaining suffix of the tape contents to verify that also condition (d) is satisfied. In the affirmative, it restarts at the right delimiter \$, while in the negative the computation fails. It follows immediately that the new tape contents $u(_A w_A)_A v$ is of the form required in (1). As each rewrite operation creates an occurrence of a symbol $)_A$, which marks the position on the tape at which this rewrite operation was performed, property (d) ensures that no rewrite operation can be executed to the left of the place where a previous rewrite operation was performed. Thus, each computation of M is monotone.

Claim. If $\alpha \in \Sigma^* \cdot (F \cdot \Sigma^*)^*$ such that $|\alpha| > 3 \cdot 2^{3m}$ and $\iota(\alpha) \in \hat{L}(G, S)$, then there exist a nonterminal A and a factorization $\alpha = uxv$ such that conditions (a) to (c) above are satisfied.

Proof. Let $\alpha \in \Sigma^* \cdot (F \cdot \Sigma^*)^*$ such that $|\alpha| > 3 \cdot 2^{3m}$ and $\iota(\alpha) \in \hat{L}(G, S)$. Then there exists a derivation $S \Rightarrow_G^+ \iota(\alpha)$, which can be extended to a derivation $S \Rightarrow_G^+ \mathsf{Pr}^{\Sigma}(\alpha)$ by appending the derivation $A \Rightarrow_G^+ w_A$ for each nonterminal A occurring in $\iota(\alpha)$. As $|\mathsf{Pr}^{\Sigma}(\alpha)| \geq \frac{1}{3} \cdot |\alpha| > 2^{3m}$, the corresponding derivation tree T has height $h \geq 3m + 2$. Thus, T contains a path p of length h. All nodes along this path are labelled with nonterminals but the last one, which is labelled with a terminal. Hence, p contains $h \geq 3m + 2$ nodes that are labelled with nonterminals, and hence, there exists a nonterminal that occurs at least four times on this path. From among all nonterminals with this property let A be the one for which the suffix p_2 of p containing four occurrences of A is of shortest length. Then we see from the arguments above that p_2 is of length at most $3m + 1$. Now the word α can be factored as $\alpha = uxv$ such that $A \Rightarrow_G^+ \mathsf{Pr}^{\Sigma}(x)$ is the derivation that corresponds to the subtree T_A of T that is rooted at the first node of the path p_2, and $S \Rightarrow_G^* uAv$ is the derivation that corresponds to the tree T' that is obtained by removing the subtree T_A (without its root) from T. Then $\iota(x) \in \hat{L}(G, A)$, $\mathsf{Pr}^{\Sigma}(x) \in L(G, A)$, and $|\mathsf{Pr}^{\Sigma}(x)| \leq 2^{3m}$.

Further, the root of p_2 is labelled with A, and p_2 contains three additional nodes labelled with A. Thus, T_A describes a derivation of the following form:

$$A \Rightarrow_G^+ u_1 A v_1 \Rightarrow_G^+ u_1 u_2 A v_2 v_1 \Rightarrow_G^+ u_1 u_2 u_3 A v_3 v_2 v_1 \Rightarrow_G^+ u_1 u_2 u_3 y v_3 v_2 v_1 = \mathsf{Pr}^{\Sigma}(x).$$

Hence, $y \in L(G, A)$ implying that $|y| \geq |w_A|$, and $|u_1 u_2 u_3| + |v_3 v_2 v_1| \geq 3$, that is, $|\mathsf{Pr}^{\Sigma}(x)| \geq |w_A| + 3$. Thus, the above factorization satisfies conditions (a) to (c). □

If starting with a word $w \in L$ satisfying $|w| > 3 \cdot 2^{3m}$, M always chooses a nonterminal A, and the leftmost factor x such that conditions (a) to (c) are satisfied and $\iota(u)Av \in \hat{L}(G, S)$ holds, then it is guaranteed that also condition (d) is satisfied. Hence, for each word $w \in L$ of length $|w| > 3 \cdot 2^{3m}$, there exists a computation of M that reduces w to a word w' such that $\mathsf{Pr}^{\Sigma}(w') \in L$ and $|w'| \leq k$, and conversely, if M accepts a word $w \in \Sigma^*$, then w belongs to the language L. It is easily seen that M is well-behaved and lexicalized. □

4 Relaxations of the Correctness Preserving Property

For each correctness preserving RLWW-automaton M, there exists a deterministic RLWW-automaton M' satisfying $L(M') = L(M)$ [5]. Actually, M' just executes some of the possible computations of M. Accordingly, we have the following consequence.

Corollary 2. *For each correctness preserving* wbl-RLWW-*automaton M, there exists a deterministic* wbl-RLWW-*automaton M' satisfying $L(M') = L(M)$.*

While $\mathcal{L}$(det-mon-RR) = $\mathcal{L}$(det-mon-RRWW) = DCFL (see [10]), it has been shown in [11] that $\mathcal{L}$(det-mon-RL) = $\mathcal{L}$(det-mon-RLWW) = LRR, where LRR denotes the class of *left-to-right regular languages* [1]. As RR- and RL-automata are lexicalized and well-behaved, we have the following consequence, where cp- denotes the property of being correctness preserving.

Corollary 3. (a) DCFL = $\mathcal{L}$(det-mon-wbl-RRWW).
(b) LRR = $\mathcal{L}$(det-mon-wbl-RLWW) = $\mathcal{L}$(cp-mon-wbl-RRWW).

Thus, for monotone wbl-RRWW-automata, by relaxing the requirement of being deterministic to that of being correctness preserving, we enlarge the class of accepted languages from DCFL to LRR. Next we introduce two notions that relax the strong requirements of the correctness preserving property.

Definition 3. *Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an* RLWW-*automaton.*

(a) *We say that M is* correctness preserving *on a word $w \in L_{\mathrm{C}}(M)$, if, for all $z \in \Gamma^*$, $w \vdash_M^{c^*} z$ implies that $z \in L_{\mathrm{C}}(M)$, too.*
(b) *Let $j \in \mathbb{N}$. A word $w \in L_{\mathrm{C}}(M)$ has* error relaxation *of degree j for M, if $m \leq j$ holds, whenever $w \vdash_M^{c} w_i$, $1 \leq i \leq m$, such that $w_1, \ldots, w_m \notin L_{\mathrm{C}}(M)$ and $w_r \neq w_s$ for all $1 \leq r < s \leq m$. We say that M has* error relaxation *of degree j, if each $w \in L_{\mathrm{C}}(M)$ has* error relaxation *of degree j for M.*

(c) *Let $i \in \mathbb{N}$. We say that M has* phase relaxation *of degree i, if, for all words $w \in L_C(M)$, all $m \geq i$, and each sequence of cycles of the form $w = w_0 \vdash_M^c w_1 \vdash_M^c \cdots \vdash_M^c w_m$, $w_m \in L_C(M)$ implies that at most i words from $\{w_0, \cdots, w_m\}$ have error relaxation of degree larger than 0 for M.*

Obviously, the property of being correctness preserving corresponds to error relaxation of degree 0 and to phase relaxation of degree 0. The error relaxation was already introduced in [6], while the phase relaxation is a variant of the cyclic relaxation of [6].

Let A be a class of RLWW-automata. By p(i)-A we denote the subclass of automata of A with phase relaxation of degree i, and by e(j)-A we denote the subclass of automata of A with error relaxation of degree j. The corresponding classes of input languages are denoted by $\mathcal{L}$(p(i)-A) and by $\mathcal{L}$(e(j)-A), respectively.

It has been observed in [12] that, for each RLWW-automaton M_L, there exists an RRWW-automaton M_R that executes exactly the same cycles as M_L. Thus, we have the following result.

Proposition 1. *Let $i, j \geq 0$, and let* X $\in$ {p(i),e(j)}. *For each* X-wbl-RLWW-*automaton M, there exists an* X-wbl-RRWW-*automaton M' that executes exactly the same cycles as M, and so $L_C(M') = L_C(M)$ and $L(M') = L(M)$.*

Hence, in the following we can restrict our attention to RRWW-automata.

Example 1 (*cont.*) M_1 is lexicalized, well-behaved, and monotone. For all $w \in \{a,b\}^*$ and all $d \in \{a,b\}$, the only accepting computation of M_1 that begins with the restarting configuration $q_0 ¢ wd(wd)^R \$$ is $wddw^R \vdash_{M_1}^c wCw^R \vdash_{M_1}^{c^*} C$. As M_1 is correctness preserving on all words from the sublanguage $\{\, wCw^R \mid w \in \{a,b\}^* \,\}$, it follows that M_1 has phase relaxation of degree 1.

However, M_1 does not have error relaxation of bounded degree. Let $w_m := (aabb)^m a$, where m is sufficiently large. Then $w_m w_m^R = (aabb)^{2m} aa \in L_{pal}$. However, starting from the restarting configuration $q_0 ¢ w_m w_m^R \$$, M_1 has $4m+1$ options to apply meta-instruction (1), that is, there are $4m+1$ possible cycles $w_m w_m^R \vdash_{M_1}^c y$, but only one of them yields a word from $L_C(M_1)$.

The language L_{pal} is not accepted by any deterministic RRWW-automaton, as $\mathcal{L}$(det-RRWW) coincides with the class CRL of Church-Rosser languages (see [10]), while L_{pal} is not Church-Rosser [3]. For proving that L_{pal} is not accepted by *any* lexicalized well-behaved RRWW-automaton with error relaxation of bounded degree we will use the following modification of a pumping lemma for restarting automata (see, e.g., [7]).

Proposition 2. *For any* RRWW-*automaton M, there exists a constant p such that the following holds. Assume that $uvw \vdash_M^c uv'w$, where $u = u_1 u_2 \cdots u_n$ for some non-empty words $u_1, \ldots, u_n$ and a constant $n > p$. Then there exist $r, s \in \mathbb{N}$, $1 \leq r < s \leq p$, such that*

$$u_1 \cdots u_{r-1} (u_r \cdots u_{s-1})^i u_s \cdots u_n vw \vdash_M^c u_1 \cdots u_{r-1} (u_r \cdots u_{s-1})^i u_s \cdots u_n v'w$$

holds for all $i \geq 0$, that is, $u_r \cdots u_{s-1}$ is a 'pumping factor' in the above cycle. Similarly, such a pumping factor can be found in any factorization of length *greater than p of w. Such a pumping factor can also be found in any factorization of length greater than p of a word accepted in a tail computation.*

Lemma 1. *The language L_{pal} is not accepted by any lexicalized well-behaved* RRWW*-automaton with error relaxation of finite degree.*

Proof. Let $\Sigma = \{a, b\}$, and assume that $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is a well-behaved RRWW-automaton such that $L(M) = L_{\mathrm{P}}(M) = L_{pal}$.

Let $n_0 > 0$ be an integer. We will show that there exist a word $z \in L_{pal}$ and cycles (reductions) $z \vdash^c_M z_i$, $1 \leq i \leq n_0$, such that $z_1, \ldots, z_{n_0} \notin L_{\mathrm{C}}(M)$ and $z_j \neq z_r$ for all $1 \leq j < r \leq n_0$. That is, for the first cycle on some word from $L(M)$, there are at least n_0 different mistakes that M can make.

For $m \geq 1$, let $A_m := \{\, (a^{2m}b^{2m})^n a^{2m} (b^{2m}a^{2m})^n \mid n > 0 \,\}$. Obviously, $A_m \subset L_{pal}$. Further, the following property is easily verified:

(∗) Let uyw be a factorization of a word $x \in A_m$ such that $|y| < m$. If there exists a word y' satisfying $|y'| < |y|$ such that $\mathsf{Pr}^{\Sigma}(uy'w) \in L_{pal}$, then $|u| > \frac{1}{2}|x| - 2m$ and $|w| > \frac{1}{2}|x| - 2m$.

Let p be the constant for M from Proposition 2, and let $m > \max\{k, p\}$, $n > p$, and $v := (a^{2m}b^{2m})^n a^{2m} (b^{2m}a^{2m})^n \in L(M)$. We consider an accepting computation of M on input v. From Proposition 2 it follows easily that v cannot be accepted by a tail computation of M. Thus, the accepting computation considered begins with a cycle of the form $v \vdash^c_M \omega$ for some word $\omega \in L_{\mathrm{C}}(M)$, which implies that $w := \mathsf{Pr}^{\Sigma}(\omega) \in L_{pal}$. From (∗) it follows that the corresponding rewrite operation occurs at a position in the range $[\frac{1}{2}|v| - 2m, \frac{1}{2}|v| + 2m]$, that is, a factor of the middle part $b^m a^{2m} b^m$ is being rewritten. The word v has a factorization of the form $v = \alpha_1\alpha_2 \cdots \alpha_n y \beta_1 \cdots \beta_n$ such that $\alpha_t = a^{2m}b^{2m}$ for all $t = 1, \ldots, n-1$, $\alpha_n = a^{2m}b^m$, $y = b^m a^{2m} b^m$, $\beta_1 = b^m a^{2m}$, and $\beta_t = b^{2m}a^{2m}$ for all $t = 2, \ldots, n$. Thus, according to Proposition 2 there exist indices r, s, $1 \leq r < s \leq p < n$ such that

$$\begin{aligned}\alpha_1 \cdots \alpha_{r-1}(\alpha_r \cdots \alpha_{s-1})^i \alpha_s \cdots \alpha_n y \beta_1 \cdots \beta_n \quad &\vdash^c_M \\ &\alpha_1 \cdots \alpha_{r-1}(\alpha_r \cdots \alpha_{s-1})^i \alpha_s \cdots \alpha_n y' \beta_1 \cdots \beta_n,\end{aligned}$$

for some word y' and all exponents $i \geq 0$. Obviously, the length of the pumping factor $\alpha_r \cdots \alpha_{s-1}$ divides the number $4m \cdot p!$. Analogously a pumping factor with corresponding properties can be found in the suffix $\beta_2 \cdots \beta_n$ of v. Hence, there exist words $v_{i,j}$ and $\omega_{i,j}$ obtained by pumping the cycle $v \vdash^c_M \omega$ independently on the left and on the right of the rewritten factor y such that

$$\begin{aligned}v_{i,j} &= (a^{2m}b^{2m})^{n+ip!} a^{2m} b^m y b^m a^{2m} (b^{2m}a^{2m})^{n+jp!},\\ \omega_{i,j} &= (a^{2m}b^{2m})^{n+ip!} a^{2m} b^m y' b^m a^{2m} (b^{2m}a^{2m})^{n+jp!},\end{aligned}$$

and $v_{i,j} \vdash^c_M \omega_{i,j}$ for all $i, j \in \mathbb{N}$. Consider the words $v_\ell := v_{\ell, n_0 - \ell} = (a^{2m}b^{2m})^{2n + n_0 p! + 2} a^{2m}$ and $\omega_\ell := \omega_{\ell, n_0 - \ell}$ for $\ell = 0, \ldots, n_0$. For any ℓ_1, ℓ_2, $0 \leq \ell_1 < \ell_2 \leq n_0$, $\omega_{\ell_1} \neq \omega_{\ell_2}$, as these words arise by rewriting v_ℓ at different places. Since at most one of the words $\mathsf{Pr}^{\Sigma}(\omega_0), \ldots, \mathsf{Pr}^{\Sigma}(\omega_{n_0})$ is a palindrome, M can make at least n_0 mistakes in the first cycle on the word v_ℓ from $L(M)$. □

This lemma yields the following consequences.

Corollary 4. $L_{pal} \in \mathcal{L}(\mathsf{p(1)\text{-}mon\text{-}wbl\text{-}RRWW}) \setminus \bigcup_{j\geq 0} \mathcal{L}(\mathsf{e}(j)\text{-}\mathsf{wbl\text{-}RRWW})$.

Actually we obtain an infinite hierarchy of language classes based on the degree of phase relaxation of lexicalized well-behaved restarting automata.

Theorem 2. *For all* $i \geq 1$ *and all* $\mathsf{X} \in \{\mathsf{wbl}, \mathsf{mon\text{-}wbl}\}$, $\mathcal{L}(\mathsf{p}(i-1)\text{-}\mathsf{X\text{-}RLWW}) \subsetneq \mathcal{L}(\mathsf{p}(i)\text{-}\mathsf{X\text{-}RLWW})$.

Proof. It remains to show that these inclusions are proper. By Proposition 1 we only need to consider RRWW-automata. Let

$$L_{pal}^{(i)} := \{dww^R dw_1 w_1^R dw_2 w_2^R d ... dw_{i-1} w_{i-1}^R \mid w, w_1, \ldots, w_{i-1} \in \{a,b\}^*\}.$$

In the first cycle a monotone RRWW-automaton M can guess the middle of the first syllable ww^R, which is a non-deterministic step of unbounded degree of error relaxation. Then it deletes the syllable ww^R completely in the next cycles, which is a sequence of deterministic steps. Thereafter it processes the next syllables in the same manner. Thus, M is also lexicalized and well-behaved, and it has phase relaxation of degree i. On the other hand, even a non-monotone wbl-RRWW-automaton has to guess the middle of all i syllables, that is, it will also have phase-relaxation of degree at least i (see the proof of Lemma 1). ☐

Corresponding hierarchy results also hold for the degree of error relaxation.

Theorem 3. *For all* $j \geq 1$ *and all* $\mathsf{X} \in \{\mathsf{wbl}, \mathsf{mon\text{-}wbl}\}$, $\mathcal{L}(\mathsf{e}(j-1)\text{-}\mathsf{X\text{-}RLWW}) \subsetneq \mathcal{L}(\mathsf{e}(j)\text{-}\mathsf{X\text{-}RLWW})$.

Proof. For proving that these inclusions are proper, we consider another family of example languages. For $m \geq 1$, let $\Sigma := \{a,b\}$, $\Gamma_m := \{a, b, C_0, \ldots, C_m\}$, and $Le_{1,m} := \{\, a^n b^{2^i \cdot n} \mid 0 \leq i \leq m,\ n \geq 1 \,\}$. Then $Le_{1,m}$ is accepted by the monotone RRWW-automaton $M_{1,m}$ that is specified as follows, where $0 \leq i \leq m$:

$(1.i)$ $(¢ \cdot a^+, ab^{2^i} \to C_i, (b^{2^i})^+ \cdot \$)$, $(3.i)$ $(¢ \cdot a \cdot \{\lambda, C_i\} \cdot b^{2^i} \cdot \$, \mathsf{Accept})$.
$(2.i)$ $(¢ \cdot a^+, aC_i b^{2^i} \to C_i, (b^{2^i})^+ \cdot \$)$,

Then $L_C(M_{1,m}) = Le_{1,m} \cup L'_{1,m}$, where $L'_{1,m} := \{\, a^n C_i b^{2^i \cdot n} \mid 0 \leq i \leq m,\ n \geq 1 \,\}$. Thus, $M_{1,m}$ is lexicalized and well-behaved.

Starting from an initial configuration $q_0 ¢ a^n b^{2^m \cdot n}\$$, there are $m+1$ possible cycles that $M_{1,m}$ can execute, but only one of them yields a word from $L_C(M_{1,m})$, and this word actually belongs to the subset $L'_{1,m}$. As $M_{1,m}$ is correctness preserving on $L'_{1,m}$, it follows that $M_{1,m}$ has phase relaxation of degree 1 and error relaxation of degree m.

However, it can be shown that the language $Le_{1,m}$ is not accepted by any well-behaved RRWW-automaton with error relaxation of degree lower than m. Assume that M is a well-behaved RRWW-automaton for $Le_{1,m}$. Given an input of the form $a^n b^{2^m \cdot r} \in Le_{1,m}$, where n is sufficiently large, M has an accepting computation. However this cannot simply be a tail computation. Thus, it begins with a cycle of the form $a^n b^{2^m \cdot r} \vdash^c_M a^{n-s}\alpha b^{2^m \cdot r - t}$, where $1 \leq s, t < k$ and $|\alpha| < s + t \leq k$. Here k denotes the window size of M. As the computation considered is accepting, we have $a^{n-s}\alpha b^{2^m \cdot r - t} \in L_{\mathrm{C}}(M)$, which in turn implies that $\mathsf{Pr}^{\Sigma}(a^{n-s}\alpha b^{2^m \cdot r - t}) \in Le_{1,m}$. Hence, we see that $\mathsf{Pr}^{\Sigma}(a^{n-s}\alpha b^{2^m \cdot r - t}) = a^{n-s'} b^{2^m \cdot r - 2^i \cdot s'}$ for some number $i \in \{0, \ldots, m\}$ and a small number $s' \geq 1$. By a technique similar to the one used in the proof of Lemma 1 we can show that there are $m + 1$ possible values for i, each of which could be the correct one, while all others are wrong. Therefore, in this situation M has at least m options, and so it has error relaxation of degree at least m. □

In analogy to the RRWW-automaton $M_{1,1}$ from the proof of Theorem 3, an e(1)-mon-wbl-RRWW-automaton can be designed for the language $(Le_{1,1})^+$, but it can be shown that there is no well-behaved RRWW-automaton with bounded phase relaxation for it. In addition, the language $L_{pal,+} := \{\, dww^R \mid w \in \{a,b\}^* \,\}^+$, which is easily seen to be accepted by a monotone wbl-RRWW-automaton with phase relaxation of unbounded degree, is not accepted by any non-monotone wbl-RRWW-automaton with phase relaxation of bounded degree, either. Together with Lemma 1 and Corollary 4 this yields the following results.

Corollary 5. *For all* $\mathsf{X} \in \{\mathsf{wbl}, \mathsf{mon\text{-}wbl}\}$*:*

(a) $\bigcup_{i \geq 0} \mathcal{L}(\mathsf{p}(i)\text{-}\mathsf{X}\text{-}\mathsf{RLWW}) \subsetneq \mathcal{L}(\mathsf{X}\text{-}\mathsf{RLWW})$.
(b) $\bigcup_{j \geq 0} \mathcal{L}(\mathsf{e}(j)\text{-}\mathsf{X}\text{-}\mathsf{RLWW}) \subsetneq \mathcal{L}(\mathsf{X}\text{-}\mathsf{RLWW})$.
(c) $\mathcal{L}(\mathsf{p}(i)\text{-}\mathsf{X}\text{-}\mathsf{RLWW})$ *and* $\mathcal{L}(\mathsf{e}(j)\text{-}\mathsf{X}\text{-}\mathsf{RLWW})$ *are incomparable under inclusion for all* $i, j \geq 1$.

5 Conclusion

We have seen that the two relaxations of the notion of correctness preservation for lexicalized well-behaved restarting automata yield infinite hierarchies of language classes, both in the monotone and in the non-monotone case. The hierarchies for the monotone classes are depicted in Figure 1.

$$\begin{tikzcd}
 & \mathsf{p}(0) \arrow[r] \arrow[d, equal] & \mathsf{p}(1) \arrow[r] & \mathsf{p}(2) \arrow[r] & \cdots \arrow[r] & \bigcup_{i \geq 0} \mathsf{p}(i) \arrow[d] & \\
\mathsf{DCFL} \arrow[r] & \mathsf{LRR} \arrow[d, equal] & & & & \mathcal{L}(\mathsf{mon\text{-}wbl\text{-}RLWW}) \arrow[r, equal] & \mathsf{CFL} \\
 & \mathsf{e}(0) \arrow[r] & \mathsf{e}(1) \arrow[r] & \mathsf{e}(2) \arrow[r] & \cdots \arrow[r] & \bigcup_{j \geq 0} \mathsf{e}(j) \arrow[u] &
\end{tikzcd}$$

Fig. 1. Infinite hierarchies of classes of input languages of monotone lexicalized well-behaved restarting automata based on the degrees of phase relaxation and error relaxation. Here $\mathsf{p}(i)$ stands for the language class $\mathcal{L}(\mathsf{p}(i)\text{-}\mathsf{mon\text{-}wbl\text{-}RLWW})$, and analogously for $\mathsf{e}(j)$. Each arrow denotes a proper inclusion. If there is no oriented path between two language classes in the picture, then these classes are incomparable under inclusion.

All example languages considered so far are context-free. Thus, the question arises of whether the language classes defined above do only contain context-free languages. This, however, is not the case.

In fact, in [10] it is shown that already the class $\mathcal{L}(\mathsf{det\text{-}RL}) \subseteq \mathcal{L}(\mathsf{cp\text{-}wbl\text{-}RLWW})$ contains an encoding of the copy language $\{\, w\#w \mid w \in \{a,b\}^* \,\}$, which is not even growing context-sensitive. On the other hand, the Church-Rosser language $\{\, a^{2^n} \mid n \geq 0 \,\}$ is not the proper language of any lexicalized RRWW-automaton [8], and hence, it is not contained in $\mathcal{L}(\mathsf{wbl\text{-}RLWW})$. Thus, all language classes $\mathcal{L}(\mathsf{p}(i)\text{-}\mathsf{wbl\text{-}RLWW})$ $(i \geq 0)$ and $\mathcal{L}(\mathsf{e}(j)\text{-}\mathsf{wbl\text{-}RLWW})$ $(j \geq 0)$ are incomparable under inclusion to the class of growing context-sensitive languages. However, it remains to classify $\mathcal{L}(\mathsf{det\text{-}wbl\text{-}RLWW})$ and $\mathcal{L}(\mathsf{wbl\text{-}RLWW})$, which form the bottom and the top of the non-monotone hierarchies.

References

1. Čulik II, K., Cohen, R.: LR-regular grammars - an extension of LR(k) grammars. J. Comput. System Sci. 7, 66–96 (1973)
2. 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)
3. Jurdziński, T., Loryś, K.: Church-Rosser languages vs. UCFL. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 147–158. Springer, Heidelberg (2002)
4. Lopatková, M., Plátek, M., Sgall, P.: Towards a formal model for functional generative description: Analysis by reduction and restarting automata. The Prague Bull. of Math. Linguistics 87, 7–26 (2007)
5. Messerschmidt, H., Otto, F.: On determinism versus nondeterminism for restarting automata. Inform. and Comput. 206, 1204–1218 (2008)
6. Mráz, F., Otto, F., Plátek, M.: Hierarchical relaxations of the correctness preserving property for restarting automata. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 230–241. Springer, Heidelberg (2007)
7. Mráz, F., Plátek, M., Jurdziński, T.: Ambiguity by restarting automata. Int. Journ. of Found. of Comp. Sci. 18, 1343–1352 (2007)
8. Mráz, F., Otto, F., Plátek, M.: The degree of word-expansion of lexicalized RRWW-automata - A new measure for the degree of nondeterminism of (context-free) languages. Theor. Comput. Sci. 410, 1343–1352 (2007)
9. Otto, F.: Restarting automata and their relations to the Chomsky hierarchy. In: Ésik, Z., Fülöp, Z. (eds.) DLT 2003. LNCS, vol. 2710, pp. 55–74. Springer, Heidelberg (2003)
10. 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)
11. Otto, F.: Left-to-right regular languages and two-way restarting automata. RAIRO Theor. Inf. Appl. 43, 653–665 (2009)
12. Plátek, M.: Two-way restarting automata and j-monotonicity. In: Pacholski, L., Ružička, P. (eds.) SOFSEM 2001. LNCS, vol. 2234, pp. 316–325. Springer, Heidelberg (2001)
13. Salomaa, K., Yu, S.: Nondeterminism degrees for context-free languages. In: Dassow, J., Rozenberg, G., Salomaa, A. (eds.) Developments in Language Theory II, Proc., pp. 154–165. World Scientific, Singapore (1996)

On a Powerful Class of Non-universal P Systems with Active Membranes

Antonio E. Porreca, Alberto Leporati, and Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{porreca,leporati,zandron}@disco.unimib.it

Abstract. We prove that uniform and semi-uniform families of P systems with active membranes using only communication and nonelementary division rules are not computationally universal. However, they are powerful enough to solve exactly the problems solvable by Turing machines operating in time and space that are "tetrational" (i.e., bounded by a stack of exponentials of polynomial height) with respect to the size of the input.

1 Introduction

Membrane systems, usually called *P systems*, have been introduced in [4] as computing devices inspired by the internal working of biological cells. The main feature of P systems is a structure of membranes dividing the space into regions, inside which multisets of objects describe the molecular environment. A set of rules describe how the molecules (and often the membranes themselves) evolve during the computation; usually, the rules are applied in a maximally parallel way, i.e., each component of the P system *must* be subject to a rule during each computation step, if a suitable rule exists. When multiple rules may be applied to an object or membrane, one of them is nondeterministically chosen. The computation, starting from an initial configuration, proceeds until no further rule can be applied. For an introduction to membrane computing, we refer the reader to [7,8], and for the latest information to the P Systems Webpage [14], where an extensive bibliography on the topic can be found.

Families of P systems can be used as language recognizers, by associating with each input string (or with each input length) a P system; this association is subject to a uniformity condition (i.e., it must be computed by a Turing machine operating in polynomial time). The constructed P systems can then accept or reject, thus deciding the membership of strings to the language. The computational complexity of recognizer P systems with *active membranes* [5], where the membranes themselves play an important role during the computation, has been subject to extensive investigation, due to their ability of solving **NP**-complete [5] and even **PSPACE**-complete problems [13] in polynomial time: this efficiency is due to the possibility of creating in polynomial time an exponential number of membranes, which then evolve in parallel, using *membrane division* (a process found in nature, e.g., in cell mitosis).

From the computability standpoint, this class of P systems is known to be equivalent to Turing machines [6]: this result, however, is proved by using *object evolution* rules, which can increase the number of objects an arbitrary number of times. In this paper, we analyze the case in which only *communication* rules (which move an object from a region to another) and *nonelementary division* rules (which only apply to membranes containing other membranes) are allowed, and prove that the resulting P systems are not computationally universal; nonetheless, they are very powerful, as they characterize the class of languages decidable by Turing machines using time (or, equivalently, space) bounded by an exponential function iterated polynomially many times (known as *tetration*).

The paper is organized as follows. In Section 2 we define P systems with active membranes, their use as language recognizers and their space complexity; we also recall the notion of tetration and introduce a class of languages having that kind of complexity. In Section 3 we prove that P systems with active membranes using only communication and nonelementary division rules are not universal, by showing how they can be simulated by tetrational-space bounded Turing machines. In Section 4 we introduce a P system module which will then be used in Section 5 in order to prove that the reverse simulation is also possible, thus characterizing exactly the power of our class of P systems. Section 6 concludes the paper and describes possible future research.

2 Definitions

Let us start by recalling the definition of the model of P systems we will use in this paper.

Definition 1. *A* P system with active membranes *with polarizations (using only communication and nonelementary division rules) of degree $m \geq 1$, is a tuple*

$$\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_m, R)$$

where:

- *Γ is a finite alphabet of symbols, also called* objects*;*
- *Λ is a finite set of labels for the membranes;*
- *μ is a membrane structure (i.e., a rooted unordered tree) consisting of m membranes enumerated by $1, \ldots, m$. Furthermore, each membrane is labeled by an element of Λ, non necessarily in an injective way;*
- *$w_1, \ldots, w_m$ are strings over Γ, describing the multisets of objects placed in the m initial regions of μ;*
- *R is a finite set of rules.*

The membrane corresponding to the root of μ is called the skin*, the ones corresponding to the leaves* elementary membranes*, whereas the others are called* nonelementary. *Each membrane possesses a further attribute, named* polarization *or* electrical charge, *which is either neutral (represented by* 0*), positive* $(+)$ *or negative* $(-)$*.*

The rules are of the following kinds:

- Send-in communication rules, *of the form* $a\,[\,]_h^\alpha \to [b]_h^\beta$
  They can be applied to a membrane labeled by h*, having polarization* α *and such that the external region contains an occurrence of the object* a*; the object* a *is sent into* h *becoming* b *and, simultaneously, the polarization of* h *is changed to* β*.*
- Send-out communication rules, *of the form* $[a]_h^\alpha \to [\,]_h^\beta\, b$
  They can be applied to a membrane labeled by h*, having polarization* α *and containing an occurrence of the object* a*; the object* a *is sent out from* h *to the outside region becoming* b *and, simultaneously, the polarization of* h *is changed to* β*.*
- Non-elementary division rules, *of the form*

$$[[\,]_{h_1}^{+} \cdots [\,]_{h_k}^{+} [\,]_{h_{k+1}}^{-} \cdots [\,]_{h_n}^{-}]_h^\alpha \to [[\,]_{h_1}^{\delta} \cdots [\,]_{h_k}^{\delta}]_h^\beta [[\,]_{h_{k+1}}^{\varepsilon} \cdots [\,]_{h_n}^{\varepsilon}]_h^\gamma$$

  They can be applied to a membrane labeled by h*, having polarization* α*, containing the positively charged membranes* $h_1, \ldots, h_k$*, the negatively charged membranes* $h_{k+1}, \ldots, h_n$*, and possibly some neutral membranes. The membrane* h *is divided into two copies having polarization* β *and* γ*, respectively; the positive children are placed inside the former, their polarizations changed to* δ*, while the negative ones are placed inside the latter, their polarizations changed to* ε*. Any neutral membrane inside* h *is duplicated (together with the objects and membranes it contains) and placed inside both copies.*

A *configuration* in a P system with active membranes is described by its membrane structure, together with its polarizations and the multisets of objects contained in its regions. The initial configuration is given by μ, all membranes having polarization 0 and the initial contents of the membranes being $w_1, \ldots, w_m$. A *computation step* changes the current configuration according to the following principles:

- Each object and membrane can be subject to at most one rule per computation step.
- The rules are applied in a *maximally parallel way*: each object which appears on the left-hand side of applicable communication rules must be subject to exactly one of them; the same holds for each membrane which can be involved in a communication or nonelementary division rule. The only objects and membranes which remain unchanged are those associated with no rule, or with unapplicable rules.
- When more than one rule can be applied to an object or membrane, the actual rule to be applied is nondeterministically chosen; hence, in general, multiple configurations can be reached from the current one.
- Every object which is sent out from the skin membrane cannot be brought in again.

A halting computation $\boldsymbol{\mathcal{C}}$ of a P system Π is a finite sequence of configurations $(\mathcal{C}_0, \ldots, \mathcal{C}_k)$, where $\mathcal{C}_0$ is the initial configuration of Π, every $\mathcal{C}_{i+1}$ can be reached from $\mathcal{C}_i$ according to the principles just described, and no further configuration can be reached from $\mathcal{C}_k$ (i.e., no rule can be applied). P systems might also perform non-halting computations; in this case, we have infinite sequences $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$ of successive configurations.

We can use families of P systems with active membranes as language recognizers, thus allowing us to solve decision problems [9].

Definition 2. *A* recognizer P system with active membranes Π *has an alphabet containing two distinguished objects yes and no, used to signal acceptance and rejection respectively; every computation of Π is halting and exactly one object among yes, no is sent out from the skin membrane during the last step of each computation.*

In what follows we will only consider *confluent* recognizer P systems with active membranes, in which all computations starting from the initial configuration agree on the result.

Definition 3. *Let $L \subseteq \Sigma^\star$ be a language and let $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a family of recognizer P systems. We say that $\boldsymbol{\Pi}$* decides *L, in symbols $L(\boldsymbol{\Pi}) = L$, when for each $x \in \Sigma^\star$, the result of Π_x is acceptance iff $x \in L$.*

Usually some uniformity condition, inspired by those applied to families of Boolean circuits, is imposed on families of P systems. Two different definitions are used:

Definition 4. *A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be* polynomially semi-uniform *when the mapping $x \mapsto \Pi_x$ can be computed in polynomial time, with respect to $|x|$, by a deterministic Turing machine.*

Definition 5. *A family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ is said to be* polynomially uniform *when there exist two polynomial-time Turing machines M_1 and M_2 such that, for each $n \in \mathbb{N}$ and each $x \in \Sigma^n$*

- *M_1, on input 1^n (the unary representation of the length of x), outputs the description of a P system Π_n with a distinguished input membrane;*
- *M_2, on input x, outputs a multiset w_x (an encoding of x);*
- *Π_x is Π_n with w_x added to the multiset located inside its input membrane.*

In other words, the P system Π_x associated with string x consists of two parts; one of them, Π_n, is common for all strings of length $|x| = n$ (in particular, the membrane structure and the set of rules fall into this category), and the other (the input multiset w_x for Π_n) is specific to x. The two parts are constructed independently and, only as the last step, w_x is inserted in Π_n.

In this paper we denote by $\mathbf{MC}_{\mathcal{D}}$ (resp., $\mathbf{MC}^\star_{\mathcal{D}}$) the class of languages that can be decided by uniform (resp., semi-uniform) families of confluent P systems of type $\mathcal{D}$ (e.g., $\mathcal{AM}$ denotes P systems with active membranes) without any restriction on time complexity. The space complexity of P systems [10] is defined as follows:

Definition 6. *Let $\mathcal{C}$ be a configuration of a P system Π. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is defined as the sum of the number of membranes in the current membrane structure and the total number of objects they contain. If $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_k)$ is a halting computation of Π, then the* space required by $\boldsymbol{\mathcal{C}}$ *is defined as*

$$|\boldsymbol{\mathcal{C}}| = \max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_k|\}$$

or, in the case of a non-halting computation $\boldsymbol{\mathcal{C}} = (\mathcal{C}_i : i \in \mathbb{N})$,

$$|\boldsymbol{\mathcal{C}}| = \sup\{|\mathcal{C}_i| : i \in \mathbb{N}\}.$$

Indeed, non-halting computations might require an infinite amount of space (in symbols $|\boldsymbol{\mathcal{C}}| = \infty$): for example, if the number of objects strictly increases at each computation step.

The space required by Π *itself is then*

$$|\Pi| = \sup\{|\boldsymbol{\mathcal{C}}| : \boldsymbol{\mathcal{C}} \textit{ is a computation of } \Pi\}.$$

Notice that $|\Pi| = \infty$ might occur if either Π has a non-halting computation requiring infinite space (as described above), or Π has an infinite set of halting computations, such that for each bound $b \in \mathbb{N}$ there exists a computation requiring space larger than b.

Finally, let $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ be a family of recognizer P systems; also let $f\colon \mathbb{N} \to \mathbb{N}$. We say that $\boldsymbol{\Pi}$ operates within space bound f *iff $|\Pi_x| \leq f(|x|)$ for each $x \in \Sigma^\star$.*

Next, we recall the definition of (deterministic) register machines.

Definition 7. *A* deterministic n-register machine *is a construct $R = (n, I, m)$, where $n > 0$ is the number of registers, I is a finite sequence of instructions (program) bijectively labeled with the elements of the set $\{1, 2, \ldots, m\}$, 1 is the label of the first instruction to be executed, and m is the label of the last instruction of I. Registers contain non-negative integer values. The instructions of I have the following forms:*

- *"$i\colon \textsc{inc}(r), j$" with $i \in \{1, 2, \ldots, m\}$, $j \in \{1, 2, \ldots, m+1\}$ and $r \in \{1, 2, \ldots, n\}$. This instruction, labeled with i, increments the value contained in register r, and then jumps to instruction j.*
- *"$i\colon \textsc{dec}(r), j, k$" with $i \in \{1, 2, \ldots, m\}$, $j, k \in \{1, 2, \ldots, m+1\}$ and $r \in \{1, 2, \ldots, n\}$. If the value contained in register r is positive then decrement it and jump to instruction j. If the value of r is zero then jump to instruction k, without altering the contents of the register.*

Computations start by executing the first instruction of I (labeled with 1), and halt when the instruction currently executed tries to jump to label $m + 1$.

Formally, a *configuration* is a $(n+1)$-tuple whose components are the contents of the n registers, and the label of the next instruction of I to be executed. In the *initial* configuration this label is set to 1, whereas it is equal to $m+1$ in any *final* configuration. A *computation* starts in the initial configuration and proceeds by performing computation steps, i.e., executing the current instruction, modifying accordingly the contents of the affected register and the pointer to the next instruction. A computation halts if it reaches a final configuration. The contents of the registers in the initial configuration are regarded as the input, and those in the final one as the output of the computation. A non-halting computation does not produce a result.

We will also use the operation of *tetration* (iterated exponentiation) to describe upper bounds on time and space complexity:

Definition 8. *For $n \in \mathbb{N}$, let tetration be defined by*

$$^{n}b = \begin{cases} 1 & \text{if } n = 0 \\ b^{(^{n-1}b)} & \text{if } n > 0 \end{cases}, \qquad \text{that is,} \qquad ^{n}b = \underbrace{b^{b^{\cdot^{\cdot^{\cdot^{b}}}}}}_{n \text{ times}}$$

We denote by **PTETRA** *the class of languages decidable by deterministic Turing machines in time $^{p(n)}2$ for some polynomial p.*

This class is trivially closed under union, intersection, complement and polynomial time reductions; furthermore, since a TM operating in space $f(n)$ can be simulated in time $2^{O(f(n))}$, **PTETRA** is also the class of languages decidable in *space* $^{p(n)}2$ for some polynomial p (we will make use of this characterization later). It is important to notice that **PTETRA**, despite being a very large class, does not contain all recursively enumerable (or even recursive) languages, as a consequence of the hierarchy theorems [3] for Turing machines.

3 Proof of Non-universality

We begin by providing an upper bound on the size that can be reached by the membrane structure of a P system not using elementary division. If we ignore labels and polarizations, any membrane structure of size m is a subtree of the complete m-ary tree of depth $m-1$.[1] For the sake of finding the upper bound, we can thus assume that the membrane structure we are considering has exactly this shape. A distinctive feature of this tree is *uniformity* [1], that is, the number of descendants of each node (hence the size and shape of the subtree rooted in it) only depends on its level. Since we only deal with uniform trees in this section, we denote them by $T(n_1, \ldots, n_{k-1})$ as in [1], where n_i is the number of nodes on level i (the 0-th level implicitly contains only the root node). The complete tree we just described is then denoted by $T(m, m^2, \ldots, m^{m-2}, m^{m-1})$.

Since nonelementary division rules separate the positively and negatively charged children membranes (explicitly listed on the left-hand side of the rules

[1] Smaller trees containing all subtrees of m nodes are known to exist, but none of them is substantially (i.e., exponentially) smaller than this one [1].

themselves) while duplicating the neutral ones, it should be clear that the number of newly created membranes can be maximized by using only one positive and one negative membrane, as follows:

$$[[\,]_{h_1}^{+}[\,]_{h_2}^{-}]_h^{\alpha} \to [[\,]_{h_1}^{\delta}]_h^{\beta}\ [[\,]_{h_2}^{\epsilon}]_h^{\gamma}$$

Suppose this rule is applied to a membrane h having $k \geq 2$ children (among these, a positively charged instance of h_1 and a negatively charged instance of h_2, while all the other children membranes are neutral). As a result, membrane h is doubled, and each of the two resulting instances of h possesses $k-1$ children (one of the two charged ones, and a copy of all the neutral ones). For the rest of this section, we ignore polarizations and labels, and assume that nonelementary division rules (all of the above form) are always applicable when needed, as this is the worst-case scenario. Under these assumptions, a membrane can divide as long as it has at least two children. Notice that communication rules play no role here, as they cannot increase the number of membranes.

The number of membranes created during the computation is maximized when division rules are applied in a bottom-up fashion, that is, by first applying all of them to membranes on the lowest possible level of the tree, until no further division is possible, and only then proceeding to the upper level. Indeed, so doing the membranes on the upper level will have more children, and so will be, in turn, susceptible to more division rules. Notice that membranes on the same level are, instead, independent as far as nonelementary division is concerned: we can thus apply division rules to all membranes of the current level simultaneously, in a maximally parallel way, in order to simplify the calculations.

According to the principles described thus far, each node on level $m-2$ of the tree

$$T(m, m^2, \ldots, m^{m-2}, m^{m-1}),$$

which is the lowest where nonelementary division can be applied, is doubled $m-1$ times by applying every possible division rules in a maximally parallel way; the first few steps are

$$T\big(m, m^2, \ldots, 2 \cdot m^{m-2}, 2 \cdot m^{m-2}(m-1)\big)$$
$$T\big(m, m^2, \ldots, 2^2 \cdot m^{m-2}, 2^2 \cdot m^{m-2}(m-2)\big)$$
$$T\big(m, m^2, \ldots, 2^3 \cdot m^{m-2}, 2^3 \cdot m^{m-2}(m-3)\big)$$

and the resulting tree is

$$T\big(m, m^2, \ldots, m^{m-3}, 2^{m-1} \cdot m^{m-2}, 2^{m-1} \cdot m^{m-2}\big).$$

Repeating the whole process at level $m-3$ yields

$$T\big(m, m^2, \ldots, m^{m-4}, 2^{2^{m-1} \cdot m-1} \cdot m^{m-3}, 2^{2^{m-1} \cdot m-1} \cdot m^{m-3}, 2^{2^{m-1} \cdot m-1} \cdot m^{m-3}\big).$$

In general, the number of nodes of level $m-1-k$ of tree $T(m, m^2, \ldots, m^{m-1})$ after applying all division rules on the lowest k levels (excluding the leaves) is $m^{m-1-k} \cdot N(k)$ where

$$N(k) = \begin{cases} 1 & \text{if } k = 0 \\ 2^{N(k-1)m-1} & \text{if } m-2 \geq k > 0. \end{cases}$$

After all possible division rules have been applied on level k, the number of nodes is also $m^{m-1-k} \cdot N(k)$ in all levels below it. Hence the final tree, obtained by applying all division rules on all $m-2$ levels below the root (in a bottom-up fashion), has a total of $1 + (m-1) \cdot m \cdot N(m-2)$ nodes, and it can be proved that this amount is ${}^{O(m^2)}2$. As a consequence, the following statement holds.

Lemma 1. *Let Π be a P system that does not use elementary division rules, with an initial membrane structure of size m. The maximum number of membranes of Π, in any of its computations, is bounded by ${}^{O(m^2)}2$.* □

Now suppose that only communication and nonelementary division rules are allowed. The only way to create new objects is, then, dividing a membrane, thus duplicating all objects inside it. An upper bound on the maximum number of objects which can be generated by a P system of this kind can be computed as follows.

Lemma 2. *Let Π be a P system using only communication and nonelementary division rules, with an initial membrane structure of size m, and k objects in its initial configuration. The maximum number of objects of Π, in any of its computations, is bounded by ${}^{O(m^2)}2 \cdot k$.*

Proof. Suppose that, before *each* application of a nonelementary division rule to a membrane, all objects are moved inside that membrane, so that they are doubled when it divides: this hypothetical situation is the worst case. Clearly, the number of applications of division rules which may occur during the whole computation is bounded by the final number of membranes, which is, in turn, bounded by ${}^{O(m^2)}2$ (Lemma 1). Then, the number of objects is bounded by $2^{({}^{O(m^2)}2)} \cdot k$, that is, by ${}^{O(m^2)}2 \cdot k$. □

Lemmata 1 and 2 allow us to prove that recognizer P systems using only communication and nonelementary division rules are not universal, as they can be simulated by Turing machines operating in tetrational space.

Theorem 1. *Let $\mathcal{D}$ be the class of recognizer P systems using only communication and nonelementary division rules. Then $\mathbf{MC}^{\star}_{\mathcal{D}} \subseteq \mathbf{PTETRA}$.*

Proof. Let $L \in \mathbf{MC}^{\star}_{\mathcal{D}}$. Then L is decided by a semi-uniform family $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ of P systems of type $\mathcal{D}$. Since there exists a polynomial-time Turing machine constructing each Π_x, there is a polynomial p such that each P system Π_x has a description of size at most $p(n)$, where $n = |x|$. But then both the size of the initial membrane structure and the initial number of objects in each P system of $\boldsymbol{\Pi}$ are bounded by $p(n)$. Hence, by Lemmata 1 and 2, the family $\boldsymbol{\Pi}$ requires ${}^{O(p(n)^2)}2 \cdot p(n)$ space; since a family of P systems can be simulated by a Turing machine using asymptotically the same space [12], the statement of the theorem follows. □

4 A Useful Gadget

The absence of object evolution rules (i.e., rules such as $[a \to w]_h^\alpha$ which transform an object into a multiset of objects without using the membrane structure) from the class of P systems we are considering makes the process of designing membrane algorithms a daunting task. In this section we introduce a gadget that can be adapted for several uses, in particular as a timer (to generate a specific object after a certain number of time steps) and as a generator of exponentially or even tetrationally many copies of an object (which is useful to increase the polynomial amount of objects available in the initial configuration), thus partially fulfilling the role of evolution rules. These features will be needed in Section 5 in order to simulate a register machine using tetrational space.

The gadget is a P system "module" $\Pi_{d,w}$ with a membrane structure having a tree representation with $w + d - 1$ nodes; $d - 1$ of them are arranged in a chain, while the remaining w are children of the lowest node in the chain, thus they correspond to the innermost membranes. The membranes are labeled by $h_1, \dots, h_d$ according to the level of the tree they belong to. We give here just an informal description of the inner working of the gadget; a complete description involves a number of other purely technical details (for instance, a constant number of extra membranes per level must be added in order to provide a way of delaying the action of objects, without using object evolution rules).

Two objects named ℓ and r, initially located inside h_{d-1}, enter two of the membranes labeled by h_d and set their charge to positive and negative respectively; at this point, a nonelementary division rule can be applied to h_{d-1}, which splits in two copies containing all the neutral membranes h_d and one of the two charged membranes; the objects ℓ and r are also duplicated. Both copies of h_{d-1} are now in a configuration analogous to the initial one (only with one less child), and the process is repeated in parallel inside both of them. The procedure stops when each of the 2^{w-1} resulting copies of h_{d-1} has a unique child, at which point an object c is sent to h_{d-2} to signal that the operations on level $d-1$ have been completed; object c activates the objects ℓ and r of the level above, and the whole process is repeated there. When an object c finally reaches h_1 (the root of the membrane structure), another object e is produced, thus signaling the completion of the division procedure on all the levels.

Choosing $w = 3$ is sufficient to produce at least ${}^{d-2}2$ membranes on the level immediately below the root (and on every level below it). This requires at least ${}^{d-3}2$ steps of nonelementary divisions on that level: as a consequence, the object e is only produced after this amount of time steps have passed. By choosing an appropriate value of d, we can use $\Pi_{d,3}$ as a clock, when we need a certain object e to appear only after some other operations have been carried out.

Similarly, by adding a child membrane containing a copy of the object a to each membrane labeled by h_d in $\Pi_{d,3}$ we can obtain at least ${}^{d-2}2$ copies of a, located inside the elementary membranes. When the object e is produced, it can be used to release the copies of a (by changing the polarization of the membranes that contain them), which can then travel upwards and be sent out from h_1. Notice that, by modifying $\Pi_{3,w}$ this way instead, we obtain *exactly* 2^{w-1} copies of object a.

5 Solving PTETRA Problems

We defined the complexity class **PTETRA** in terms of Turing machines, operating in time or space ${}^{p(n)}2$, but the class remains the same when register machines are used. Indeed, register machines can simulate TMs by using asymptotically the same space (understood as the number of bits needed to represent the values of the registers) by interpreting the tape of the TM as two binary integers corresponding to the portions of the tape on the left and on the right of the head, respectively [2]. A register machine recognizes a set of natural numbers by computing its characteristic function. Since strings may have meaningful leading 0s, before feeding them as input to the register machines we prefix them by 1 (this operation is, obviously, a bijection between $\{0,1\}^\star$ and $1\{0,1\}^\star$). By simulating these register machines via P systems, using only communication and nonelementary division rules, we can prove the reverse inclusion of Theorem 1.

It is known [11] that a register machine can be simulated by a uniform family of P systems using only communication rules as long as the values stored in the registers can be represented in unary using a polynomial number of symbols. Each register r is represented by a membrane having label r and containing a number of occurrences of object a equal to the value of the register; increment and decrement operations are performed by sending in or out another occurrence of a when the polarization of r is changed via a program-counter object (with a subscript denoting the current instruction).

In our case, however, the unary encoding of an input string $x \in 1\{0,1\}^n$ requires exponentially many objects; furthermore, tetrationally many auxiliary objects are required during the computation. We can solve both these problems by using the gadget described in the previous section, only requiring the addition of nonelementary division rules.

Let $L \in$ **PTETRA**, and let R be a register machine accepting all integers with binary encoding $1x$ for $x \in L$, and rejecting integers with binary encoding $1x$ for $x \notin L$; assume that R works in space ${}^{p(n)}2$ for some polynomial p. We simulate R via a uniform family of P systems $\boldsymbol{\Pi} = \{\Pi_x : x \in \{0,1\}^\star\}$.

The membrane structure of all systems Π_x with $|x| = n$ is the following one:

$$\mu_n = \big[[\mu'_0 \cdots \mu'_n]_1^0 [\,]_2^0 \cdots [\,]_k^0 \mu'' \mu''' [\,]_z^0\big]_0^0$$

where k is the number of registers of R, 1 is the input register, μ'' is the membrane structure of a P system module sending out ${}^{p(n)}2$ copies of object a (thus providing the auxiliary objects needed to carry out the computation), μ''' is the membrane structure of a module generating object p_1 after $O(^{p(n)}2)$ steps, and z is an auxiliary membrane (used to simulate "waiting" without using object evolution rules [11]). Membrane μ'_i (for $0 \leq i \leq n$) sends out 2^i copies of object a inside the input membrane 1 if and only if is activated by an object y_i entering its outermost membrane; this process is used to convert the binary input into unary notation. The membrane structure μ_n, together with the associated set of rules and initial objects, can be constructed from 1^n by a Turing machine operating in polynomial time.

The encoding of $x = x_{n-1} \cdots x_0$, to be placed inside the input membrane 1, is constructed as follows: for all $i \in \{0, \ldots, n-1\}$, object y_i is part of the input multiset iff $x_i = 1$; furthermore, y_n is also part of the input multiset. Clearly, this encoding can be computed from x by a Turing machine working in polynomial time; hence, the family of P systems $\boldsymbol{\Pi}$ is uniform.

Each P system Π_x of $\boldsymbol{\Pi}$ operates as follows. As described above, the object y_i enters the membrane structure μ'_i which activates and sends out 2^i copies of object a to membrane 1. When the process is completed for all y_i, the multiplicity of a inside membrane 1 is exactly the same as the number having binary representation $1x$. Simultaneously, inside μ'' we produce enough copies of a to carry out the simulation of the register machine, which are then sent out to the skin membrane one at a time. Finally, we use μ''' to delay the appearance of the program counter object p_1 until all the instances of object a (both those encoding the input and the auxiliary ones) have been generated and moved to their initial position (membrane 1 and the skin, respectively).

When p_1 is sent out to the skin membrane, the real simulation (which is described in detail in [11]) begins. The object p_i, where i is the currently simulated instruction, moves between regions and changes the polarizations of the membranes, thus activating rules that bring in or send out a copy of object a to one of the register membranes, i.e., performing increment and decrement operations. For instance, the following rules simulate the instruction "$i : \textsc{inc}(r), j$":

$$p_i\ [\ ]_r^0 \to [p'_i]_r^+ \qquad\qquad a\ [\ ]_r^+ \to [a]_r^0 \qquad\qquad [p'_i]_r^0 \to [\ ]_r^0\ p_j$$

Decrement instructions "$i : \textsc{dec}(r), j, k$" are implemented similarly, only with the difference that the program counter object p_i must wait at least one step in order to let a copy of object a (that possibly occurs inside r) be sent out, thus changing the charge of membrane r; now p_i is sent out of r as p_j or p_k depending on whether an a was actually sent out (i.e., the value of register r was nonzero).

The computation halts when the object *yes* (resp., *no*) is sent out from the skin membrane of Π_x if the simulated register machines accepts (resp., rejects), that is, if $x \in L$ (resp., $x \notin L$). Thus, the family $\boldsymbol{\Pi}$ recognizes L.

Since L is an arbitrary **PTETRA** language, and given the space requirements of $\boldsymbol{\Pi}$ and the result of Theorem 1, we proved the following statement:

Theorem 2. *Let $\mathcal{D}$ be the class of P systems with active membranes using only communication and elementary division rules. Then* $\mathbf{PTETRA} = \mathbf{MC}_{\mathcal{D}} = \mathbf{MC}^{\star}_{\mathcal{D}}$. □

6 Conclusions

We proved that P systems with active membranes using only communication and nonelementary division rules are computationally weaker than Turing machines. In particular, they characterize the class **PTETRA** of languages decided by Turing machines whose resources are bounded by ${}^{p(n)}2$ for some polynomial p. This statement is proved by extending a recently published simulation of register machines [11].

The fact that this kind of P systems, despite their non-universality, are able to decide such a large class of languages suggests some topics for future research. Are there any other classes of P systems (and, more generally, of classic or bio-inspired computing devices) that exhibit a similar behavior? If this is not the case, how can universal computing devices be restricted (in a different way than simply imposing a resource bound) or enriched in order to achieve it?

References

1. Chung, F.R.K., Graham, R.L., Coppersmith, D.: On trees containing all small trees. In: Chartrand, G. (ed.) The Theory of Applications of Graphs, pp. 265–272. Wiley, Chichester (1981)
2. Fischer, P.C., Meyer, A.R., Rosenberg, A.L.: Counter machines and counter languages. Mathematical Systems Theory 2(3), 265–283 (1968)
3. Hartmanis, J., Stearns, R.E.: On the computational complexity of algorithms. Transactions of the American Mathematical Society 117, 285–306 (1965)
4. Păun, G.: Computing with Membranes. Journal of Computer and System Sciences 1(61), 108–143 (2000)
5. Păun, G.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
6. Păun, G.: Active membranes. In: Păun, G., Rozenberg, G., Salomaa, A. (eds.) The Oxford Handbook of Membrane Computing, pp. 282–301. Oxford University Press, Oxford (2010)
7. Păun, G., Rozenberg, G.: A guide to membrane computing. Theoretical Computer Science 287, 73–100 (2002)
8. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
9. Pérez-Jiménez, M.J., Romero Jiménez, A., Sancho Caparrini, F.: Complexity classes in models of cellular computing with membranes. Natural Computing 2, 265–285 (2003)
10. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: Introducing a space complexity measure for P systems. International Journal of Computers, Communications & Control 4(3), 301–310 (2009)
11. Porreca, A.E., Leporati, A., Mauri, G., Zandron, C.: P systems with active membranes: Trading time for space. Natural Computing (to appear)
12. Porreca, A.E., Mauri, G., Zandron, C.: Complexity classes for membrane systems. RAIRO Theoretical Informatics and Applications 40(2), 141–162 (2006)
13. Sosík, P.: The computational power of cell division in P systems: Beating down parallel computers? Natural Computing 2(3), 287–298 (2003)
14. The P Systems Webpage, `http://ppage.psystems.eu`

State Complexity of Prefix, Suffix, Bifix and Infix Operators on Regular Languages

Elena V. Pribavkina[1] and Emanuele Rodaro[2]

[1] Ural State University, 620083, Ekaterinburg, Russia
[2] Departemento de Matematica, Facultade de Ciências
Universidade do Porto, 4169-007 Porto, Portugal
`elena.pribavkina@usu.ru, emanuele.rodaro@fc.up.pt`

Abstract. We consider four operators on a regular language. Each of them is a tool for constructing a code (respectively *prefix*, *suffix*, *bifix* and *infix*) out of a given regular language. We give the precise values of the (deterministic) state complexity of each of these operators.

1 Background and Motivation

Let Σ be a finite alphabet and Σ^* denote the set of all *words* over the alphabet Σ. By Σ^+ we denote the set of all *non-empty* words in Σ^*. A word $u \in \Sigma^+$ is a *factor* (*prefix* or *suffix* respectively) of a word w if w can be decomposed as $w = xuy$ ($w = uy$ or $w = xu$ respectively) for some $x, y \in \Sigma^*$. A *language* over Σ is any subset of Σ^*. A subset X of Σ^* is said to be a *code*, if any word $w \in X^*$ can be written *uniquely* as a concatenation of words in X, namely $w = x_1 x_2 \cdots x_n$ with $x_1, x_2, \dots x_n \in X$. A language L is said to be *prefix-free* (*suffix-free*, *infix-free* respectively) if no word in L is a prefix (suffix, factor respectively) of any other word in L. Obviously a prefix-free (respectively suffix-free, infix-free) language consisting of non-empty words is a code, and such a code is called *prefix* (respectively *suffix*, *infix*). A code which is both prefix and suffix is called *bifix*.

Codes play an important role in many areas such as information processing, data compression and cryptography. In the applications different types of codes are used. For example, widely-used Huffman codes are prefix-free languages, and their advantage is that a given encoded word can be decoded deterministically. From the formal language theory point of view codes are languages, but not vice versa. In this paper we consider four operators on a regular language, each of them is a tool for constructing a code (prefix, suffix, bifix or infix respectively) out of a given regular language L. The obtained codes are maximal with respect to inclusion among all the codes contained in L with a given property. Moreover they contain all the words of minimal length in L. The *prefix operator* deletes all the words w in L having a prefix (different from w) which is an element of L. The language obtained is denoted by L^p and obviously is a prefix code. Formally we define $L^p = L \setminus L\Sigma^+$. Analogously the *suffix operator* deletes all the words w in L having a suffix (different from w) belonging to L. The corresponding language is denoted by L^s and it is a suffix code. Formally $L^s = L \setminus \Sigma^+ L$. The language L^b obtained by the *bifix operator* is the intersection $L^b = L^s \cap L^p$, and clearly it is a bifix code. The *infix operator* builds the language L^ι of all the words of L such that none of their inner factors is in L. The formal definition is $L^\iota = L \setminus (\Sigma^+ L \Sigma^* \cup \Sigma^* L \Sigma^+)$. By the closure properties of the class of regular languages it is clear that if L is a regular language, all the languages L^p, L^s, L^b and L^ι are also regular.

Recall that a *deterministic finite automaton* (DFA for short) is a tuple $\mathscr{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is the set of states, Σ is the alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states. Note that we assume that δ is defined in each state for all letters (such automata are usually called *complete*). The language $L(\mathscr{A})$ *recognized* by a DFA $\mathscr{A}$ is the set of words corresponding to the labels of paths starting at the initial state and ending in some final state. If $\mathscr{A}$ is a DFA recognizing the language L, the words in L^b represent efficient tasks of the automaton $\mathscr{A}$ since they are accepted without useless first and last computation, while the language L^ι represents optimal computations. Thus it makes sense studying the state complexity of each of the operators $\cdot^p, \cdot^s, \cdot^b$ and $\cdot^\iota$. Recall that the *(deterministic) state complexity* of a regular language L (denoted by $sc(L)$) is the number of states of its minimal DFA. *The state complexity of an operator* $\cdot^x$ *on a regular language* is the number of states that are necessary and sufficient in the worst-case for the minimal DFA that accepts the language obtained after applying the operator. Formally $sc_x(n) = \max\{sc(L^x) \mid sc(L) = n\}$. The state complexity of different operations on regular languages has been widely studied in the recent years. For instance Câmpeanu et al. [4] studied the state complexity in the case of finite languages, Pighizzini and Shallit [12,13] examined the state complexity of unary language operations, Jirásková [10] investigated the state complexity of some operations on binary regular languages. For a comprehensive survey of the state complexity of regular languages see [17]. A number of recent papers were dedicated to the study of the state complexity of different operations on prefix-free and suffix-free languages. In [6] Han and Salomaa studied the state complexity of basic operations on suffix-free regular languages. Results concerning the deterministic and nondeterministic state complexity of some operations on prefix-free regular languages were obtained by Han et al. [7,8,9]. In [11] Kao, Rampersad and Shallit et al. study a complementary problem, that is the state complexity of prefix-closed, suffix-closed, or factor closed languages.

Note, that in particular case when L is a two-sided ideal in Σ^*, we have $L^b = L^\iota$ and this language consists of the generators of L, i.e. $L = \Sigma^* L^b \Sigma^*$. Moreover in this case we have simply $L^s = L \setminus \Sigma L$, $L^p = L \setminus L\Sigma$ and $L^b = L^\iota = L \setminus (\Sigma L \cup L \Sigma)$. Such a case arises for instance when we deal with the language of *synchronizing* words of a *synchronizing* automaton. See [15] for more details and the surveys [14,16] on synchronizing automata. The four operators and their state complexities in the case of languages which are ideals were recently studied by Brzozowski, Jiraskova and Li [3].

2 State Complexity of the Prefix Operator

Let $L = L(\mathscr{A})$ be a regular language recognized by the minimal DFA $\mathscr{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$. Some combinatorial properties of the language L^p were studied in [1, Chaper 3], where it was introduced as a tool for constructing prefix codes. The following statement is rather trivial. We put it here for the sake of completeness.

Proposition 1. *$sc_p(n) = n + 1$.*

Proof. Clearly if $q_0 \in F$, then L^p consists of just the empty word, so we may assume that $q_0 \notin F$. Then we modify the automaton $\mathscr{A}$ to accept the language L^p in the following way. We redirect all the outgoing edges from all the final states to the sink state s not belonging to F (if such a state exists, otherwise it is added). Formally, we obtain the automaton $\mathscr{A}^p = \langle Q \cup \{s\}, \Sigma, \delta', q_0, F\rangle$ where for all $a \in \Sigma$ we have $\delta'(q, a) = \delta(q, a)$ if $q \notin F$, and for all $q \in F$ we put $\delta'(q, a) = s$, $\delta'(s, a) = s$. It is straightforward to check that $L(\mathscr{A}^p) = L^p$. Therefore $n + 1$ states are sufficient. They are also necessary. Indeed, consider the language $L = \Sigma^{\geq n-1}$ of all the words over the alphabet Σ of length at least $n-1$ for $n \geq 2$. Clearly the minimal DFA recognizing this language has n states. Then using the previous construction we get that $L^p = \Sigma^{n-1}$ is accepted by the minimal DFA with $n + 1$ states. □

3 State Complexity of the Suffix Operator

Since $L^s = L \setminus \Sigma^+ L$ a naive construction from a DFA with n states accepting L, leads to a DFA accepting L^s with at most $n2^{n+1}$ states. In this section we improve this bound and show that the new bound is tight.

Proposition 2. *Let $\mathscr{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ be the n-state minimal DFA. The language $L(\mathscr{A})^s$ is recognized by a DFA with at most $(n - 1)2^{n-2} + 2$ states.*

Proof. We can suppose without loss of generality that $q_0 \notin F$ otherwise $L(\mathscr{A})^s$ would consist of just the empty word. Let $\hat{\mathscr{A}}$ be the automaton obtained from $\mathscr{A}$ by erasing all the ingoing edges of q_0. Then clearly $L(\hat{\mathscr{A}})$ consists of the set of words in $L(\mathscr{A})$ which label paths in the automaton $\mathscr{A}$ passing through the state q_0 only once. Moreover $L(\mathscr{A})^s = L(\hat{\mathscr{A}})^s$. Thus to simplify the notations we assume from the beginning that there are no cycles on the state q_0 in $\mathscr{A}$ (for the construction itself it does not matter whether the initial automaton is complete).

Consider the following automaton $\mathscr{P} = \langle P, \Sigma, \delta_P, i_P, F_P\rangle$ with the state set $P = \{s, (q_0, \varnothing)\} \cup \{(q, H) \mid q \in Q \setminus \{q_0\}, H \subseteq Q\}$, the initial state $i_P = (q_0, \varnothing)$, the set of final states is $F_P = \{(q, H) \mid q \in F, H \cap F = \varnothing\}$ and the transition function for all $a \in \Sigma$ is defined in the following way:

$$\delta_P((q, H), a) = \begin{cases} (\delta(q, a), \delta(H, a) \cup \{q_0\}) & \text{if } \delta(q, a) \text{ is defined,} \\ s \text{ otherwise;} \end{cases}$$
$$\delta_P(s, a) = s$$

(here $\delta(H,a) = \bigcup_{h\in H} \delta(h,a)$). Note that the automaton $\mathscr{P}$ is complete and s is its sink state. Now we prove that $L(\mathscr{P}) = L(\mathscr{A})^s$. Let $w = a_1a_2\cdots a_n \in L(\mathscr{A})^s$. Since $w \in L(\mathscr{A})$, we have $\delta_P(i_P, w) \neq s$. Then it is easy to see by induction that $\delta_P(i_P, w) = (q, H)$ where $q \in F$ and $H = \bigcup_{i=2}^{n} \{\delta(q_0, a_i \cdots a_n)\} \cup \{q_0\}$. Since $q_0 \notin F$ and none of the suffixes $a_i \cdots a_n$ $i > 1$ is in $L(\mathscr{A})$, we have $H \cap F = \varnothing$. Hence $w \in L(\mathscr{P})$.

Conversely, let $w = a_1a_2\cdots a_n \in L(\mathscr{P})$. This means that $\delta_P(i_P, w) = (q, H)$ with $q \in F$ and $H \cap F = \varnothing$. As we have seen $H = \bigcup_{i=2}^{n} \{\delta(q_0, a_i \cdots a_n)\} \cup \{q_0\}$, hence $w \in L(\mathscr{A})$ and none of its suffixes different from w is in $L(\mathscr{A})$, thus $w \in L(\mathscr{A})^s$.

Clearly the automaton $\mathscr{P}$ has $(n-1)2^n+2$ states. However a simple inspection shows that if a state (q, H) is reachable from i_P then $q_0 \in H$. Thus the trimmed automaton has at most $(n-1)2^{n-1} + 2$ states. Moreover the right language of a state (q, H) with $q \in H$ is empty since if for some word $w \in \Sigma^*$ $\delta_P((q, H), w) = (q', H')$ with $q' \in F$, then clearly $q' \in H'$ and $H' \cap F \neq \emptyset$, so $(q', H') \notin F_P$. Thus in the minimal automaton recognizing $L(\mathscr{P})$ all the states (q, H) with $q \in H$ are identified with the sink state s. There are $(n-1)2^{n-2}$ such states (for each $q \neq q_0$ we have all the subsets H of Q containing q_0 and q, and there are 2^{n-2} such subsets), and so we get that the minimal DFA $\mathscr{A}^s$ accepting $L(\mathscr{A})^s$ has at most $(n-1)2^{n-1} + 2 - (n-1)2^{n-2} = (n-1)2^{n-2} + 2$ states. □

Proposition 3. *For every $n \geq 4$ there exists an n-state DFA $\mathscr{A}_n$ over a 4-letter alphabet such that the minimal DFA recognizing the language $L(\mathscr{A}_n)^s$ has exactly $(n-1)2^{n-2} + 2$ states.*

Proof. For $n \geq 3$ consider the DFA $\mathscr{A}_{n+1} = \langle Q_{n+1}, \Sigma, \delta, 0, \{1\}\rangle$ where $Q_{n+1} = \{0, 1, \ldots, n\}$, the alphabet consists of four elements $\Sigma = \{a, b, c, d\}$, the transition function is defined by the following rules (see Fig. 1):
$\delta(i, a) = i + 1$ for $0 \leq i \leq n-1$, $\delta(n, a) = 1$;
$\delta(0, b) = 0$, $\delta(i, b) = i + 1$ for $1 \leq i \leq n-1$, $\delta(n, b) = 1$;
$\delta(0, c) = 0$, $\delta(1, c) = 1$, $\delta(n, c) = 2$ and $\delta(i, c) = i + 1$ for $2 \leq i \leq n-1$;
$\delta(i, d) = i$ for $i \neq 2$, $\delta(2, d) = 1$.

Consider the automaton $\mathscr{P}_{n+1}$ built from $\mathscr{A}_{n+1}$ as in Proposition 2. We prove that the minimal DFA $\mathscr{A}^s_{n+1}$ accepting the language $L(\mathscr{A}_{n+1})^s$ has $n2^{n-1} + 2$ states. First we show that in the automaton $\mathscr{A}^s_{n+1}$ all the pairs (q, H) with $q \neq 0$, $q \notin H$ and $0 \in H$ are reachable. Indeed consider the state (q, H) with $H = \{0, h_1, \ldots, h_r\}$, $h_1 > h_2 > \cdots > h_r$. Then applying the word

$$w_{(q,H)} = ab^{(q-h_1-1) \bmod n}ab^{h_1-h_2-1}ab^{h_2-h_3-1} \cdots ab^{h_{r-1}-h_r-1}ab^{h_r-1}$$

to the initial state $(0, \varnothing)$ we get exactly the state (q, H). Indeed, we have

$$\delta_P((0, \varnothing), a) = (1, \{0\});$$
$$\delta_P((1, \{0\}), b^{(q-h_1-1) \bmod n}) = ((q - h_1) \bmod n, \{0\}).$$

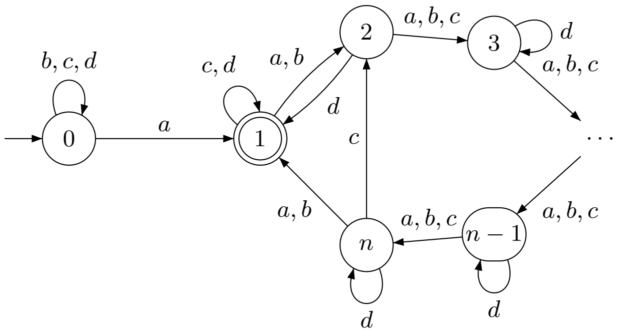


Fig. 1. Automaton $\mathscr{A}_{n+1}$

Denote $(q_1, H_1) = ((q-h_1) \bmod n, \{0\})$ and let $(q_i, H_i) = ((q-h_i) \bmod n, \{0, h_1 - h_i, h_2 - h_i, \dots, h_{i-1} - h_i\})$ for $2 \leq i \leq r-1$. Then it is easy to check by induction that

$$\delta_P((q_i, H_i), ab^{h_i - h_{i+1} - 1}) = (q_{i+1}, H_{i+1}).$$

Finally we have $(q_r, H_r) = ((q - h_r) \bmod n, \{0, h_1 - h_r, \dots, h_{r-1} - h_r\})$, thus $\delta_P((q_r, H_r), ab^{h_r - 1}) = (q, H)$.

Now let us prove that the right languages of two different reachable pairs (q, H), (q', H') are different. Let us denote these languages respectively by $L_{(q,H)}$ and $L_{(q',H')}$. If $q \neq q'$ suppose without loss of generality that $q < q'$, then it easy to see that the word $b^{n-q'+1}$ belongs to $L_{(q',H')} \setminus L_{(q,H)}$, since $\delta(q', b^{n-q'+1}) = 1 \neq \delta(q, b^{n-q'+1})$ and in $\mathscr{A}^s_{n+1}$ any state of the form $(1, K)$ with $1 \notin K$, $0 \in K$ is final. Therefore we can suppose $q = q'$ and so $H \neq H'$. After applying the word b^{n-q+1} the states (q, H), (q, H') move to the states $(1, K)$, $(1, K')$ respectively, where $K \neq K'$ and both K, K' do not contain 1. We can suppose without loss of generality that there is an element $i \in K \setminus K'$ (note that $i \neq 1$). Thus after applying the word c^{n-i+1} to the states $(1, K)$ and $(1, K')$ we obtain the states $(1, K_1)$, $(1, K_1')$, and since $\delta(i, c^{n-i+1}) = 2$ we have $2 \in K_1 \setminus K_1'$. Therefore after applying the letter d to both states $(1, K_1)$, $(1, K_1')$ we obtain the states $(1, K_2)$ and $(1, K_2')$, where $1 \in K_2$ (it means that in the minimal automaton $\mathscr{A}^s_{n+1}$ this state is the sink s), while $1 \notin K_2'$. Thus the word $u = b^{n-q+1}c^{n-i+1}d$ belongs to $L_{(q,H')} \setminus L_{(q,H)}$. □

As a corollary of Propositions 2 and 3 we have the following result:

Theorem 1. $sc_s(n) = (n-1)2^{n-2} + 2$.

4 State Complexity of the Bifix Operator

In this section we consider the bifix operator $L^b = L^s \cap L^p = L \setminus (\Sigma^+ L \cup L \Sigma^+)$.

Proposition 4. *Let $\mathscr{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ be the n-state minimal DFA. The language $L(\mathscr{A})^b$ is recognized by a DFA with at most $(n-1-|F|)2^{n-2} + 3$ states.*

Proof. The automaton $\mathscr{P}$ from Proposition 2 can be easily modified to accept the language $L(\mathscr{A})^b$. Indeed consider $\mathscr{P}' = \langle P, \Sigma, \delta'_P, i_P, F_P \rangle$ obtained from $\mathscr{P}$ by redirecting all the transitions outgoing from the states (q, H) with $q \in F$ to the sink state s, i.e. for all states (q, H) with $q \in F$ and for all $a \in \Sigma$ we put $\delta'_P((q, H), a) = s$. Let us prove that $L(\mathscr{P}') = L(\mathscr{A})^b$. Indeed let $w = a_1 a_2 \cdots a_n \in L(\mathscr{A})^b$. Since $\delta(q_0, a_1 \cdots a_i) \notin F$ for all $1 \le i < n$ and $\delta(q_0, w) \in F$, we have $\delta'_P(i_P, w) \neq s$ in the modified automaton $\mathscr{P}'$. Hence as in the proof of Proposition 2 we have $\delta'_P(i_P, w) = (q, H)$ with $q \in F$, $H = \bigcup_{i=2}^{n} \{\delta(q_0, a_i \cdots a_n)\} \cup \{q_0\}$ and $H \cap F = \varnothing$. Thus $w \in L(\mathscr{P}')$. Conversely, let $w = a_1 a_2 \cdots a_n \in L(\mathscr{P}')$. Then $w \in L(\mathscr{A})^s$. Moreover by the definition of the modified transition function if there were $1 \le i < n$ such that $\delta(q, a_1 \cdots a_i) \in F$, then we would have $\delta'_P(i_P, w) = s$ which is not final. Hence none of the prefixes of w is in $L(\mathscr{A})$, therefore $w \in L(\mathscr{A})^s \cap L(\mathscr{A})^p = L(\mathscr{A})^b$.

Note that in the new automaton $\mathscr{P}'$ all the states (q, H) with $q \in F$ and $H \cap F = \varnothing$ are equivalent to the unique final state, whereas all the states (q, H) with $q \in F$ and $H \cap F \neq \varnothing$ are equivalent to the sink state s. Since there are $|F| 2^{n-2}$ states (q, H) with $q \in F$, the minimal automaton accepting $L(\mathscr{P}')$ has at most $(n - 1 - |F|)2^{n-2} + 3$ states. □

Obviously the maximum of the function $(n - 1 - |F|)2^{n-2} + 3$ is obtained in case $|F| = 1$. Let us prove that the bound $(n-2)2^{n-2} + 3$ is tight.

Proposition 5. *For every $n \ge 4$ there exists an n-state DFA $\mathscr{B}_n$ over a 4-letter alphabet such that the minimal DFA recognizing $L(\mathscr{B}_n)^b$ has $(n-2)2^{n-2} + 3$ states.*

Proof. Consider the following automaton $\mathscr{B}_{n+1} = \langle Q_{n+1}, \Sigma, \delta, 0, \{n\} \rangle$ with $Q_{n+1} = \{0, 1, \ldots, n\}$, $\Sigma = \{a, b, c, d\}$ and the transition function defined as follows (see Fig. 2):
$\delta(i, a) = i + 1$ for $0 \le i < n - 1$, $\delta(n-1, a) = 1$, $\delta(n, a) = n$;
$\delta(0, b) = 0$, $\delta(i, b) = i + 1$ for $1 \le i < n - 1$, $\delta(n-1, b) = 1$, $\delta(n, b) = n$;
$\delta(i, c) = i$ for $i \neq n - 1$, $\delta(n-1, c) = n$;
$\delta(i, d) = i$ for $i \neq n$, $\delta(n, d) = 0$.

We apply the construction from Proposition 4. First we show that all the states of the form (q, H) with $q \notin \{0, n\}$, $0 \in H$ and $q \notin H$ are reachable from the initial state $(0, \varnothing)$ in the minimal automaton $\mathscr{B}^b_{n+1}$. Let $H = \{0, h_1, h_2, \ldots, h_r\}$ with $h_1 > h_2 > \cdots > h_r$. Suppose first that $n \notin H$. Then it is not hard to see in the same way as in the proof of Proposition 3 that the word

$$w_{(q,H)} = ab^{(q-h_1-1) \bmod (n-1)} ab^{h_1-h_2-1} a \cdots ab^{h_{r-1}-h_r-1} ab^{h_r-1}$$

brings the initial state $(0, \varnothing)$ to the state (q, H). Now we assume that $n \in H$: $H = \{0, n, h_1, \ldots, h_r\}$ with $h_1 > h_2 > \cdots > h_r$. Consider the word

$$w'_{(q,H)} = a^2 b^{n-2} cab^{(q-h_1-1) \bmod (n-1)} ab^{h_1-h_2-1} a \cdots ab^{h_{r-1}-h_r-1} ab^{h_r-1}.$$

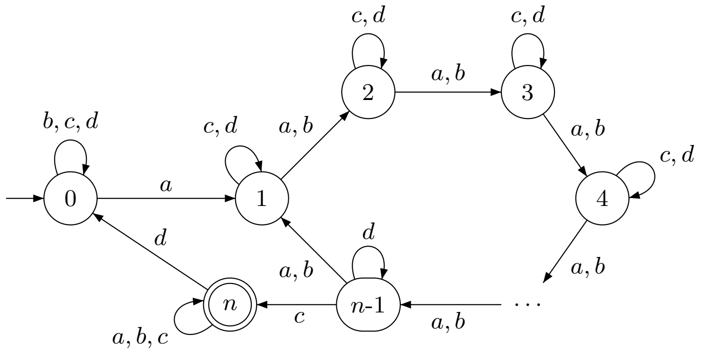


Fig. 2. Automaton $\mathscr{B}_{n+1}$

Its prefix $a^2b^{n-2}c$ brings the initial state $(0, \varnothing)$ to the state $(1, \{0, n\})$. Further since n is fixed by the letters a, b and c, applying the rest of the word $w'_{(q,H)}$ we obtain the state (q, H).

Note that in the minimal automaton $\mathscr{B}^b_{n+1}$ there is only one final state f which is a Myhill-Nerode equivalence class containing all the states of the form (n, H) with $0 \in H$ and $n \notin H$. We prove that all the pairs of non-final reachable states have different right languages. Let us consider the two states (q_1, H_1), (q_2, H_2) such that $q_1, q_2 \neq n$, $0 \in H_1 \cap H_2$, $q_1 \notin H_1$ and $q_2 \notin H_2$. After applying the letter d to both states we obtain the states (q_1, H'_1) and (q_2, H'_2) such that $H'_1 = H_1 \setminus \{n\}$ and $H'_2 = H_2 \setminus \{n\}$.

Case 1. Assume first $q_1 \neq q_2$. Without loss of generality assume $q_2 > q_1$. After applying the word $b^{n-q_1-1}c$ to both states (q_1, H'_1) and (q_2, H'_2) we obtain respectively the states (n, H''_1) and $(q_2 - q_1, H''_2)$. Since $q_1 \notin H'_1$ and $n \notin H'_1$ we have $n \notin H''_1$. Hence (n, H''_1) is equivalent to the final state f, while $(q_2 - q_1, H''_2)$ is not. Thus $db^{n-q_1-1}c \in L_{(q_1,H_1)} \setminus L_{(q_2,H_2)}$.

Case 2. Let $q_1 = q_2 = q \neq n$. Then obviously $H'_1 \neq H'_2$. Let $i \in H'_1 \setminus H'_2$ (note that $i \notin \{0, n\}$). Apply the word $b^{n-i-1}c$. Since $\delta(i, b^{n-i-1}c) = n$, we will obtain the two states (q', H''_1) and (q', H''_2) such that $n \in H''_1 \setminus H''_2$. After applying the word $b^{n-q'-1}c$ we will have the states (n, H'''_1) and (n, H'''_2) such that $n \in H'''_1 \setminus H'''_2$. Thus the state (n, H'''_2) is equivalent to the final state f, while the state (n, H'''_1) is equivalent to the sink state s. Thus $db^{n-i-1}cb^{n-\delta(q,b^{n-i-1}c)-1}c \in L_{(q,H_2)} \setminus L_{(q,H_1)}$. □

From Propositions 4 and 5 we deduce the following

Theorem 2. $sc_b(n) = (n-2)2^{n-2} + 3$.

5 State Complexity of the Infix Operator

Lemma 1. *For any language L the following equality holds: $L^\iota = (L^p \Sigma^*)^b$.*

Proof. It is easy to see that $L^p\Sigma^* = L\Sigma^*$. Thus $(L^p\Sigma^*)^b = (L\Sigma^*)^b$. By the definition we have

$$(L^p\Sigma^*)^b = (L\Sigma^*)^b = L\Sigma^* \setminus (\Sigma^+L\Sigma^* \cup L\Sigma^*\Sigma^+) = L\Sigma^* \setminus (\Sigma^+L\Sigma^* \cup L\Sigma^+). \quad (1)$$

On the other hand, by definition we have $L^\iota = L \setminus (\Sigma^+L\Sigma^* \cup \Sigma^*L\Sigma^+)$. Since $\Sigma^*L\Sigma^+ = \Sigma^+L\Sigma^+ \cup L\Sigma^+$ and $\Sigma^+L\Sigma^+ \subseteq \Sigma^+L\Sigma^*$ we get

$$L^\iota = L \setminus (\Sigma^+L\Sigma^* \cup L\Sigma^+). \quad (2)$$

Comparing (1) and (2) we get $L^\iota \subseteq (L^p\Sigma^*)^b$. Conversely, let $w \in (L^p\Sigma^*)^b$. By (1) we have in particular $w \in L\Sigma^* \setminus L\Sigma^+$, hence $w \in L$ and $w \notin \Sigma^+L\Sigma^* \cup L\Sigma^+$, so $w \in L^\iota$. □

Proposition 6. *Let $\mathscr{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ be the n-state minimal DFA. The language $L(\mathscr{A})^\iota$ is recognized by a DFA with at most $(n - |F| - 1)2^{n-|F|-2} + 3$ states.*

Proof. Since $L^\iota \subseteq L^b \subseteq L^s$, we can assume, as in Proposition 2, that there are no cycles on the state q_0 in $\mathscr{A}$ and $q_0 \notin F$. We modify the automaton $\mathscr{A}$ to obtain the automaton $\mathscr{A}'$ recognizing the language $L^p\Sigma^*$. To this aim we erase all the outgoing edges from the final states as in Proposition 1 and add loops labelled by all the elements of Σ on each final state. Obviously the right languages of all the final states are now equal, thus the minimized automaton $\mathscr{A}'$ recognizing the language $L^p\Sigma^*$ will have at most $n - |F| + 1$ states and only one of them (denote it by f) is final. Now we apply the construction of Proposition 4 to the automaton $\mathscr{A}'$. By this Proposition the resulting automaton will have at most $(n - |F| - 1)2^{n-|F|-1} + 3$ states. Moreover since f is fixed by all the letters, the right languages of all the states (q, H) (such that $q \neq q_0$, $q \neq f$, $q_0 \in H$, $q \notin H$) with $f \in H$ are empty. Thus in the minimal DFA all of them are identified with the sink state s. There are $(n-|F|-1)2^{n-|F|-2}$ such states. Thus in the minimal DFA recognizing the language L^ι there are at most $(n - |F| - 1)2^{n-|F|-2} + 3$ states. □

It is easy to see that the maximum of the function $(n - |F| - 1)2^{n-|F|-2} + 3$ is achieved in the case $|F| = 1$.

Proposition 7. *For every $n \geq 4$ there exists an n-state DFA $\mathscr{C}_n$ over a 3-letter alphabet such that the minimal DFA recognizing $L(\mathscr{C}_n)^\iota$ has exactly $(n-2)2^{n-3}+3$ states.*

Proof. For each $n \geq 3$ consider the automaton $\mathscr{C}_{n+1} = \langle Q_{n+1}, \Sigma, \delta, 0, \{n\}\rangle$ with $Q_{n+1} = \{0, 1, \ldots, n\}$, $\Sigma = \{a, b, c\}$ and the transition function defined as follows (see Fig. 3):
$\delta(i, a) = i + 1$ for $0 \leq i \leq n - 2$, $\delta(n - 1, a) = 1$, $\delta(n, a) = n$;
$\delta(0, b) = 0$, $\delta(i, b) = i + 1$ for $1 \leq i \leq n - 2$, $\delta(n - 1, b) = 1$, $\delta(n, b) = n$;
$\delta(i, c) = i$ for $0 \leq i \leq n$, $i \neq n - 1$, $\delta(n - 1, c) = n$.

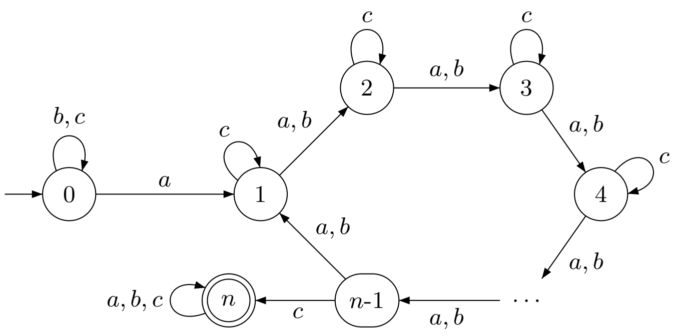


Fig. 3. Automaton $\mathscr{C}_{n+1}$

The automaton $\mathscr{C}_{n+1}$ is obtained from the automaton $\mathscr{B}_{n+1}$ from Proposition 5 by deleting the letter d. Note that the language L recognized by the automaton $\mathscr{C}_{n+1}$ is a right ideal of Σ^*, i.e. $L = L\Sigma^* = L^p\Sigma^*$. Thus we can immediately apply the construction from Proposition 4. Let $\mathscr{C}^{\iota}_{n+1}$ be the minimal DFA obtained by this construction. In Proposition 5 it is shown that all the non-final states (q, H) are reachable from the initial state $(0, \varnothing)$. By Proposition 5 the right languages of two different non-final reachable states (q_1, H_1) and (q_2, H_2) are different in case $n \notin H_1 \cup H_2$ (we do not have to apply the letter d first to guarantee that n belongs to neither H_1 nor H_2). Thus we have $(n-2)2^{n-3}$ inequivalent states of the form (q, H) with $q \notin \{0, n\}$, $0 \in H$, $q \notin H$ and $n \notin H$. There are also three additional states: the initial state, the unique final state f, and the sink state s. Thus in the minimal DFA $\mathscr{C}^{\iota}_{n+1}$ there are $(n-2)2^{n-3}+3$ states. □

By Propositions 6 and 7 we have the following result:

Theorem 3. $sc_{\iota}(n) = (n-2)2^{n-3}+3$.

6 Open Problems

In our examples showing that the bounds for the state complexity are attained, we consider automata over a 4- or 3-lettered alphabet. We are not aware if there exist series of DFA's over smaller alphabets giving same lower bounds.

Since the state complexities of suffix, bifix and infix operators are exponential, it is natural to ask whether there exist classes of languages for which the state complexities of these operators are polynomial. Some partial results on this question have been obtained recently in [3].

Acknowledgments

The first-named author acknowledges support from the Federal Education Agency of Russia, grant 2.1.1/3537, and from the Russian Foundation for Basic Research, grants 09-01-12142 and 10-01-00793. The second-named author acknowledges the support of the Centro de Matemâtica da Universidade do Porto financed by FCT through the programmes POCTI and POSI, with Portuguese and European Community structural funds, as well as the support of the FCT project SFRH/BPD/65428/2009. We are very grateful to Mikhail Volkov for having suggested the problem. We thank Giovanni Pighizzini for a fruitful discussion during LATA 2009 conference and Inna Mikhaylova for pointing out some inaccuracies in the first version of the paper. We gratefully acknowledge the precious remarks by anonymous referees, which certainly helped to improve the quality of the paper.

References

1. Berstel, J., Perrin, D., Reutenauer, C.: Codes and Automata. In: Encyclopedia of Mathematics and its Applications, vol. 129. Cambridge University Press, Cambridge (2009)
2. Birget, J.-C.: Intersection and union of regular languages and state complexity. Information Processing Letters 43, 185–190 (1992)
3. Brzozowski, J., Jiraskova, G., Li, B.: Quotient complexity of ideal languages. In: Proceedings LATIN 2010 (to appear 2010), Available at CoRR, ArXiv:0908.2083 (2009)
4. Câmpeanu, C., Culik, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 60–70. Springer, Heidelberg (2001)
5. Černý, J.: Poznámka k homogénnym eksperimentom s konečnými automatami Mat.-Fyz. Mat.-Fyz. Čas. Slovensk. Akad. Vied. 14, 208–216 (1964)
6. Han, Y.-S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. Theoret. Comput. Sci. 410(27-29), 2537–2548 (2009)
7. Han, Y.-S., Salomaa, K., Wood, D.: Nondeterministic state complexity of basic operations for prefix-free regular languages. Fundamenta Informaticae 90, 93–106 (2009)
8. Han, Y.-S., Salomaa, K., Wood, D.: Operational state complexity of prefix-free regular languages. In: Ésik, Z., Fülöp, Z. (eds.) Automata, Formal Languages and Related Topics, Institute of Informatics, pp. 99–115. University of Szeged (2009)
9. Han, Y.-S., Salomaa, K., Yu, S.: State complexity of combined operations for prefix-free regular languages. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 398–409. Springer, Heidelberg (2009)
10. Jirásková, G.: State complexity of some operations on binary regular languages. Theoret. Comput. Sci. 330, 287–298 (2005)
11. Kao, J.-Y., Rampersad, N., Shallit, J.: On NFAs where all states are final, initial, or both. Theoret. Comput. Sci. 410, 5010–5021 (2009)
12. Pighizzini, G.: Unary language concatenation and its state complexity. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 252–262. Springer, Heidelberg (2001)
13. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthals function. Internat. J. Found. Comput. Sci. 13, 145–159 (2002)
14. Mateescu, A., Salomaa, A.: Many-valued truth functions, Černý's conjecture and road coloring. EATCS Bull. 68, 134–150 (1999)

15. Pribavkina, E.V., Rodaro, E.: Finitely generated synchronizing automata. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 672–683. Springer, Heidelberg (2009)
16. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)
17. Yu, S.: State complexity of regular languages. Journal of Automata, Languages and Combinatorics 6(2), 221–234 (2001)

Restricted Ambiguity of Erasing Morphisms

Daniel Reidenbach[1] and Johannes C. Schneider[2,⋆]

[1] Department of Computer Science, Loughborough University, Loughborough, LE11 3TU, United Kingdom
D.Reidenbach@lboro.ac.uk

[2] Fachbereich Informatik, Technische Universität Kaiserslautern, Postfach 3049, 67653 Kaiserslautern, Germany
jschneider@informatik.uni-kl.de

Abstract. A morphism h is called ambiguous for a string s if there is another morphism that maps s to the same image as h; otherwise, it is called unambiguous. In this paper, we examine some fundamental problems on the ambiguity of erasing morphisms. We provide a detailed analysis of so-called ambiguity partitions, and our main result uses this concept to characterise those strings that have a morphism of strongly restricted ambiguity. Furthermore, we demonstrate that there are strings for which the set of unambiguous morphisms, depending on the size of the target alphabet of these morphisms, is empty, finite or infinite. Finally, we show that the problem of the existence of unambiguous erasing morphisms is equivalent to some basic decision problems for nonerasing multi-pattern languages.

1 Introduction

The research on the ambiguity of morphisms is based on the following, elementary questions: Given a string s and a morphism h, do there exist morphisms g with $g(s) = h(s)$, but $g(x) \neq h(x)$ for a symbol x in s? If so, what properties do these morphisms g have? For example, let $s := \mathtt{AABBCC}$, and let the morphism $h : \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}^* \to \{\mathtt{a}, \mathtt{b}\}^*$ be given by $h(\mathtt{A}) := h(\mathtt{C}) := \mathtt{a}$ and $h(\mathtt{B}) := \mathtt{b}$. Then it can be easily verified that there is no morphism g satisfying $g(s) = \mathtt{aabbaa} = h(s)$ and $g(x) \neq h(x)$ for an $x \in \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$. Therefore, we call h *unambiguous* for s. On the other, if we consider the morphism $h' : \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}^* \to \{\mathtt{a}, \mathtt{b}\}^*$, defined by $h'(\mathtt{A}) := h'(\mathtt{B}) := h'(\mathtt{C}) := (\mathtt{ab})^{10}$, then there are various other morphisms g that map s to $h'(s) = (\mathtt{ab})^{60}$. Hence, h' is *ambiguous* for s. Furthermore, for every n with $0 \leq n \leq 30$ and for every symbol $x \in \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$, there exists at least one morphism g satisfying $g(s) = h'(s)$ and $g(x) = (\mathtt{ab})^n$. Thus, the ambiguity of h' for s is largely unrestricted. In the present paper, we wish to investigate this phenomenon, and we shall mainly focus on the question of whether, for any string, there *exists* a morphism with a restricted ambiguity. To this end, we distinguish between two types of restrictions: maximally restricted ambiguity (i. e., unambiguity) and so-called *moderate ambiguity*, a sophisticated yet natural concept to be introduced below.

⋆ Corresponding Author.

The existence of unambiguous and moderately ambiguous *nonerasing* morphisms has already been intensively studied (see, e. g., Freydenberger et al. [1], Reidenbach [9]), and characteristic criteria have been provided. These criteria reveal that the existence of such morphisms is alphabet-independent, i. e., for any string s over some alphabet $\mathcal{A}$ and for any alphabets Σ, Σ' with at least two letters each, s has an unambiguous or moderately ambiguous nonerasing morphism $h : \mathcal{A}^* \to \Sigma^*$ if and only if there is a morphism $h' : \mathcal{A}^* \to \Sigma'^*$ with the equivalent property. In the present work, we study the ambiguity of all morphisms, including *erasing morphisms*, which map a symbol in s to the empty string. As pointed out by Schneider [13], here the existence of unambiguous erasing morphism does not only depend on the structure of the string, but also on the size of the target alphabet of the morphism, which turns the search for characteristic conditions into a rather intricate problem.

The examination of the ambiguity of morphisms is not only of intrinsic interest, but, due to the simplicity of the concept, also shows various connections to other topics in theoretical computer science and discrete mathematics. This primarily holds for those approaches where several morphisms are applied to one finite string, including pattern languages (see, e. g., Mateescu and Salomaa [8]) as well as equality sets (and, thus, the Post Correspondence Problem, cf. Harju and Karhumäki [2]). Particularly well understood are the relations to pattern languages, where several prominent problems have been solved using insights into the ambiguity of morphisms (see, e. g., Reidenbach [10]). Moreover, there are further connections of the ambiguity of morphisms to various concepts that involve morphisms such as fixed points of morphisms, avoidable patterns and word equations.

Our work is organised as follows: After giving some definitions and basic results, we provide a detailed analysis of *ambiguity partitions* (as introduced by Schneider [13]), which are a vital concept when investigating the ambiguity of erasing morphisms. In Section 4, we introduce and study moderate ambiguity, i. e., an important type of strongly restricted ambiguity. We characterise those strings for which there exist moderately ambiguous erasing morphisms, and this is the main result of our paper. In Section 5, we deal with unambiguous morphisms, and we study the number of such morphisms for certain strings. Finally, in Section 6, we reveal that the existence of unambiguous erasing morphisms can be characterised through basic decision problems for so-called nonerasing multi-pattern languages. This insight might be a worthwhile starting point for future research. Note that, due to space constraints, all proofs are omitted from this paper.

2 Definitions and Basic Notes

In the present section we give some basic definitions and results. For notations not explained explicitly, we refer the reader to Rozenberg and Salomaa [12].

Let $\mathbb{N} := \{1, 2, \ldots\}$ be the set of natural numbers. The *power set* of a set S is denoted by $\mathcal{P}(S)$. An *alphabet* $\mathcal{A}$ is an enumerable set of symbols. A *string (over*

$\mathcal{A}$) is a finite sequence of symbols taken from $\mathcal{A}$. By $|X|$ we denote the cardinality of a set X or the length of a string X. The *empty* string ε is the unique sequence of symbols of length 0. For the *concatenation* of strings s, t we write $s \cdot t$ (or st for short). The string that results from the n-fold concatenation of a string s is denoted by s^n. The notation $\mathcal{A}^*$ refers to the set of all strings over $\mathcal{A}$, i. e., more precisely, the *free monoid* generated by $\mathcal{A}$; furthermore, $\mathcal{A}^+ := \mathcal{A}^* \setminus \{\varepsilon\}$. The number of occurrences of a symbol $x \in \mathcal{A}$ in a string $s \in \mathcal{A}^*$ is written as $|s|_x$. With regard to arbitrary strings $s, t \in \mathcal{A}^*$, we write $s = t \ldots$ if there exists an $u \in \mathcal{A}^*$ such that $s = tu$, we write $s = \ldots t$ if there exists an $u \in \mathcal{A}^*$ such that $s = ut$, and, finally, $s = \ldots t \ldots$ if there exist $u, v \in \mathcal{A}^*$ such that $s = utv$. We call t a *prefix*, *suffix* and *factor* of s, respectively. In contrast to this notation, if we omit some parts of a canonically given string, then we henceforth use the symbol $[\ldots]$; e. g., $s = \ldots a\, b\, [\ldots]\, f$ means that s ends with the string $a\,b\,c\,d\,e\,f$.

We often use $\mathbb{N}$ as an infinite alphabet of symbols. In order to distinguish between strings over $\mathbb{N}$ and strings over a (possibly finite) alphabet Σ, we call the former *patterns*. Given a pattern $\alpha \in \mathbb{N}^*$, we call symbols occurring in α *variables* and denote the set of variables in α with $\operatorname{var}(\alpha)$. Hence, $\operatorname{var}(\alpha) \subseteq \mathbb{N}$. We use the symbol $\cdot$ to separate the variables in a pattern, so that, for instance, $1 \cdot 1 \cdot 2$ is not confused with $11 \cdot 2$.

Given arbitrary alphabets $\mathcal{A}, \mathcal{B}$, a *morphism* is a mapping $h : \mathcal{A}^* \to \mathcal{B}^*$ that is compatible with the concatenation, i. e., for all $v, w \in \mathcal{A}^*$, $h(vw) = h(v)h(w)$. Hence, h is fully defined for all $v \in \mathcal{A}^*$ as soon as it is defined for all symbols in $\mathcal{A}$. We call h *erasing* if and only if $h(a) = \varepsilon$ for an $a \in \mathcal{A}$; otherwise, h is called *nonerasing.* If we call a morphism h (non)erasing with a certain input string s in mind, we only demand h to be (non)erasing for the symbols occurring in s.

A pattern $\alpha \in \mathbb{N}^+$ is called a *fixed point (of a morphism h)* if $h(\alpha) = \alpha$. A morphism $h : \mathbb{N}^* \to \mathbb{N}^*$ is said to be *nontrivial* if $h(x) \neq x$ for an $x \in \mathbb{N}$. Let $V \subseteq \mathbb{N}$. We call $h : \mathbb{N}^* \to \mathbb{N}^*$ *nontrivial for V* if $h(x) \neq x$ for an $x \in V$. The morphism $\pi_V : \mathbb{N}^* \to \mathbb{N}^*$ is given by $\pi_V(x) := x$ if $x \in V$ and $\pi_V(x) := \varepsilon$ if $x \notin V$.

For any alphabet Σ, for any morphism $\sigma : \mathbb{N}^* \to \Sigma^*$ and for any pattern $\alpha \in \mathbb{N}^+$ with $\sigma(\alpha) \neq \varepsilon$, we call σ *unambiguous (for α)* if and only if there is no morphism $\tau : \mathbb{N}^* \to \Sigma^*$ satisfying $\tau(\alpha) = \sigma(\alpha)$ and, for some $x \in \operatorname{var}(\alpha)$, $\tau(x) \neq \sigma(x)$. If σ is not unambiguous for α, it is called *ambiguous (for α).* We extend this definition to any word $w \in \Sigma^*$ in the natural way, i. e., w is said to be *unambiguous (for α)* if there is an unambiguous morphism $\sigma : \mathbb{N}^* \to \Sigma^*$ with $\sigma(\alpha) = w$, and w is called *ambiguous (for α)* if there is an ambiguous morphism $\sigma : \mathbb{N}^* \to \Sigma^*$ satisfying $\sigma(\alpha) = w$. Furthermore, with regard to the E-pattern language of α to be introduced in the subsequent paragraph, we say that a word $w \in L_{\mathrm{E},\Sigma}(\alpha)$ is (un-)ambiguous if w is (un-)ambiguous for α.

Basically, the set of all images of a pattern $\alpha \in \mathbb{N}^+$ under morphisms $\sigma : \mathbb{N}^* \to \Sigma^*$, where Σ is an arbitrary alphabet of so-called *terminal-symbols*, is called the *pattern language (generated by α).* Formally, two main types of pattern languages of α are considered: its *E-pattern language* $L_{\mathrm{E},\Sigma}(\alpha) := \{\sigma(\alpha) \mid \sigma : \mathbb{N}^* \to \Sigma^*$ is a morphism$\}$ and its *NE-pattern language* $L_{\mathrm{NE},\Sigma}(\alpha) := \{\sigma(\alpha) \mid$

$\sigma : \mathbb{N}^* \to \Sigma^*$ is a nonerasing morphism$\}$. Note that, in literature, pattern languages as defined above are usually called *terminal-free*, since, in a more general understanding of the concept, a pattern may additionally contain terminal symbols. The morphisms $\sigma : (\mathbb{N} \cup \Sigma)^* \to \Sigma^*$ applied to such a pattern $\alpha \in (\mathbb{N} \cup \Sigma)^+$ when generating its pattern language must then be *terminal-preserving*, i. e., for any $a \in \Sigma$, $\sigma(a) = a$ must be satisfied.

We conclude the definitions in this section with a partition of the set of all patterns subject to the following criterion:

Definition 1. *Let $\alpha \in \mathbb{N}^+$. We call α* prolix *if and only if there exists a factorisation $\alpha = \beta_0\, \gamma_1\, \beta_1\, \gamma_2\, \beta_2 \dots \gamma_n\, \beta_n$ with $n \geq 1$, $\beta_i \in \mathbb{N}^*$, $0 \leq i \leq n$, and $\gamma_i \in \mathbb{N}^+$, $1 \leq i \leq n$, such that*

1. *for every $i \in \{1, 2, \dots, n\}$, $|\gamma_i| \geq 2$,*
2. *for every $i \in \{0, 1, \dots, n\}$, for every $j \in \{1, 2, \dots, n\}$, $\mathrm{var}(\beta_i) \cap \mathrm{var}(\gamma_j) = \emptyset$,*
3. *for every $i \in \{1, 2, \dots, n\}$, there exists an $y_i \in \mathrm{var}(\gamma_i)$ such that y_i occurs exactly once in γ_i and, for every $i' \in \{1, 2, \dots, n\}$, if $y_i \in \gamma_{i'}$ then $\gamma_i = \gamma_{i'}$.*

We call $\alpha \in \mathbb{N}^+$ succinct *if and only if it is not prolix.*

A succinct pattern is the shortest generator of its respective E-pattern language, i. e., for any Σ, $|\Sigma| \geq 2$, and any succinct pattern α, there is no pattern β with $|\beta| < |\alpha|$ and $L_{\mathrm{E},\Sigma}(\beta) = L_{\mathrm{E},\Sigma}(\alpha)$. Furthermore, the set of prolix patterns exactly corresponds to the class of finite fixed points of nontrivial morphisms (cf. Head [3]). Note that set of succinct patterns is also equivalent to the set of *morphically primitive* words (as introduced by Reidenbach and Schneider [11]).

Regarding the unambiguity of *nonerasing* morphisms, the classification of patterns into succinct and prolix patterns is vital:

Theorem 1 (Freydenberger, Reidenbach, and Schneider [1]). *Let $\alpha \in \mathbb{N}^+$, let Σ be an alphabet, $|\Sigma| \geq 2$. There exists an unambiguous nonerasing morphism $\sigma : \mathbb{N}^* \to \Sigma^*$ for α if and only if α is succinct.*

According to this result, for any prolix pattern α, every nonerasing morphism is ambiguous. In contrast to this negative insight, there are prolix patterns that have unambiguous *erasing* morphisms (as pointed out by Schneider [13]). However, this is not a universal property of prolix patterns; thus, certain prolix patterns do not have any unambiguous morphism at all. This phenomenon is the main topic of our paper.

3 Ambiguity Partitions

Previous results show that *ambiguity partitions* as introduced by Schneider [13] are a crucial notion when investigating the ambiguity of erasing morphisms, and the main result of our paper, given in Section 4, further illustrates their importance. In the present section, we therefore study some fundamental properties of this concept.

Definition 2. *We inductively define an* ambiguity partition (for any $\alpha \in \mathbb{N}^+$)*:*

(i) $(\emptyset, \mathrm{var}(\alpha))$ *is an ambiguity partition for* α.
(ii) *If* (E, N) *is an ambiguity partition for* α *and there exists a morphism* $h : \mathbb{N}^* \to \mathbb{N}^*$ *that is nontrivial for* N *and satisfies* $h(\alpha) = \pi_N(\alpha)$, *then* (E', N') *is an ambiguity partition with* $E' := E \cup \{x \in N \mid h(x) = \varepsilon\}$, $N' := \{x \in N \mid h(x) \neq \varepsilon\}$.

According to [13], Definition 2 permits a number of fundamental insights into the ambiguity of erasing morphisms to be established. They directly or indirectly result from the following, slightly technical fact:

Theorem 2 (Schneider [13]). *Let* Σ *be an alphabet. Let* $\alpha \in \mathbb{N}^+$ *and let* (E, N) *be an ambiguity partition for* α. *Then every morphism* $\sigma : \mathbb{N}^* \to \Sigma^*$ *satisfying* $\sigma(x) \neq \varepsilon$ *for an* $x \in E$ *is ambiguous for* α.

Consequently, for any pattern α, an ambiguity partition (E, N) for α gives us valuable information on the set S of variables in α which must be erased by unambiguous morphisms, since $S \supseteq E$. Thus, the larger the set E becomes, the more information we get. Therefore, we name ambiguity partitions with a set E of maximal size in the following definition:

Definition 3. *Let* $\alpha \in \mathbb{N}^+$. *An ambiguity partition* (E, N) *for* α *is called* maximal *if and only if every ambiguity partition* (E', N') *for* α *satisfies* $|E'| \leq |E|$ *and* $|N'| \geq |N|$.

This definition supports some of our proofs, and we can use it to express vital statements on the (non-)existence of morphisms with a restricted ambiguity.

From Definition 2, it is not obvious whether or not a maximal ambiguity partition for a pattern α is unique. However, it can be shown that, for any pattern, there is exactly one maximal ambiguity partition:

Theorem 3. *Let* $\alpha \in \mathbb{N}^+$ *and* (E, N) *be a maximal ambiguity partition for* α. *Then* (E, N) *is unique.*

Evidently, the uniqueness of the maximal ambiguity partition (E, N) of a pattern α is a nontrivial property only if $E \neq \mathrm{var}(\alpha)$. On the other hand, if $(\mathrm{var}(\alpha), \emptyset)$ is the maximal ambiguity partition of α, then it is known that the following statement on the existence of unambiguous morphisms holds true:

Corollary 1 (Schneider [13]). *Let* Σ *be an alphabet, and let* $\alpha \in \mathbb{N}^+$. *If* $(\mathrm{var}(\alpha), \emptyset)$ *is an ambiguity partition for* α, *then every morphism* $\sigma : \mathbb{N}^* \to \Sigma^*$ *is ambiguous for* α.

While Corollary 1, in the case of arbitrary alphabets Σ, uses ambiguity partitions $(\mathrm{var}(\alpha), \emptyset)$ to establishes a *sufficient* criterion on the nonexistence of unambiguous morphisms (note that [13] gives examples demonstrating that this criterion is not characteristic), an even stronger result is known for infinite Σ:

Theorem 4 (Schneider [13]). *Let* Σ *be an infinite alphabet, and let* $\alpha \in \mathbb{N}^+$. *Then* $(\mathrm{var}(\alpha), \emptyset)$ *is an ambiguity partition for* α *if and only if every morphism* $\sigma : \mathbb{N}^* \to \Sigma^*$ *is ambiguous for* α.

Thus, when investigating the existence of unambiguous erasing morphisms, the question of whether or not $(\mathrm{var}(\alpha), \emptyset)$ is an ambiguity partition for α leads to an important (and sometimes even characteristic) partition of $\mathbb{N}^+$. Therefore, we now introduce a new terminology reflecting this question:

Definition 4. *Let $\alpha \in \mathbb{N}^+$. We call α* morphically erasable *if and only if $(\mathrm{var}(\alpha), \emptyset)$ is an ambiguity partition for α. Otherwise, α is called* morphically unerasable*.*

Thus, referring to Definition 4, Corollary 1 demonstrates that, for finite alphabets Σ, the search for patterns with unambiguous morphisms can be narrowed down to the morphically unerasable ones. Therefore, and since our main result in Section 4 again is based on this property, we now give a nontrivial characterisation of such patterns. To this end, we use a condition that is based on the inclusion of E-pattern languages, which is a well-investigated problem (see Jiang et al. [6]).

Condition 1. A pattern $\alpha \in \mathbb{N}^+$ satisfies Condition 1 if and only if there exists a set $N \subseteq \mathrm{var}(\alpha)$ such that, for every $M \subseteq \mathrm{var}(\alpha)$ with $M \not\supseteq N$ and for any alphabet Σ with $|\Sigma| \geq 2$, $L_{\mathrm{E},\Sigma}(\pi_M(\alpha)) \not\supseteq L_{\mathrm{E},\Sigma}(\pi_N(\alpha))$.

Lemma 1. *A pattern $\alpha \in \mathbb{N}^+$ satisfies Condition 1 if and only if α is morphically unerasable.*

Summarising the above statements, we can note the following sufficient condition on the nonexistence of unambiguous erasing morphisms, that is equivalent to Corollary 1:

Theorem 5. *Let Σ be an alphabet. If an $\alpha \in \mathbb{N}^+$ does not satisfy Condition 1, then every morphism $\sigma : \mathbb{N}^* \rightarrow \Sigma^*$ with $\sigma(\alpha) \neq \varepsilon$ is ambiguous for α.*

The original motivation for investigating the ambiguity of morphisms is derived from inductive inference of E-pattern languages – i. e., the problem of computing a pattern from the words in its pattern languages –, which strongly depends on the inclusion relation between E-pattern languages. In this context, certain morphisms with a restricted ambiguity are known to generate words that contain reliable and algorithmically usable information about their generating pattern (cf. Reidenbach [10]) and, thus, are a vital input to any inference procedure. Theorem 5 further illustrates this close connection between the two topics.

The techniques used in [10] are based on the notion of an ambiguity of specific *nonerasing* morphisms that is restricted in a particular manner. We now introduce and study an equivalent concept for *erasing* morphisms.

4 Moderate Ambiguity

Theorem 4 shows that, in case of an infinite alphabet Σ, the property of a pattern α being morphically unerasable is characteristic for the existence of a morphism $\sigma : \mathbb{N}^* \rightarrow \Sigma^*$ that is unambiguous for α. However, concerning finite target alphabets Σ, there are morphically unerasable patterns for which there exists no unambiguous morphism (see the examples given by Schneider [13]). Although we are, hence, not able to achieve unambiguity for every morphically unerasable pattern, we shall demonstrate below that a certain restricted ambiguity is possible, which can be interpreted as unambiguity of a morphism with regard to particular factors of $\sigma(\alpha)$. As briefly mentioned above, a similar property of nonerasing morphisms is used for many fundamental results on inductive inference of E-pattern languages, and an extensive analysis of this phenomenon is provided by Reidenbach [9].

In accordance with [9], we call the said type of ambiguity *moderate ambiguity.* Intuitively, it can be understood as follows: A morphism $\sigma : \mathbb{N}^* \to \Sigma^*$ is called *moderately ambiguous* for a pattern α if, for every variable position j of a variable x in α with $\sigma(x) \neq \varepsilon$, there exists a certain factor w_j of $\sigma(\alpha)$ at a certain position (between the l_jth and r_jth letter in $\sigma(\alpha)$) such that *every* morphism $\tau : \mathbb{N}^* \to \Sigma^*$ with $\tau(\alpha) = \sigma(\alpha)$ maps the variable x at position j to a word which covers at least the factor w_j at this particular position. We illustrate this type of ambiguity in the following example:

Example 1. Let $\Sigma := \{\mathtt{a}, \mathtt{b}\}$ and $\alpha := i_1 \cdot i_2 \cdot i_3 \cdot i_4 \cdot i_5 \cdot i_6 \cdot i_7 \cdot i_8 \cdot i_9 \cdot i_{10} := 1 \cdot 2 \cdot 1 \cdot 1 \cdot 2 \cdot 1 \cdot 1 \cdot 3 \cdot 1 \cdot 3$. Let $\sigma : \mathbb{N}^* \to \Sigma^*$ be a morphism defined by $\sigma(1) := \varepsilon$, $\sigma(2) := \mathtt{aba}$, $\sigma(3) := \mathtt{abb}$. The morphism σ is ambiguous for α since $\tau : \mathbb{N}^* \to \Sigma^*$, defined by $\tau(1) := \mathtt{a}$, $\tau(2) := \mathtt{b}$, $\tau(3) := \mathtt{bb}$, satisfies $\tau(\alpha) = \sigma(\alpha)$. Hence, the situation looks as follows:

$$\sigma(\alpha) = \overbrace{\underbrace{\mathtt{a}}_{\tau(1)}\underbrace{\mathtt{b}}_{\tau(2)}\underbrace{\mathtt{a}}_{\tau(1)}}^{\sigma(2)}\overbrace{\underbrace{\mathtt{a}}_{\tau(1)}\underbrace{\mathtt{b}}_{\tau(2)}\underbrace{\mathtt{a}}_{\tau(1)}}^{\sigma(2)}\overbrace{\underbrace{\mathtt{a}}_{\tau(1)}\underbrace{\mathtt{bb}}_{\tau(3)}}^{\sigma(3)}\overbrace{\underbrace{\mathtt{a}}_{\tau(1)}\underbrace{\mathtt{bb}}_{\tau(3)}}^{\sigma(3)} = \tau(\alpha).$$

However, we call σ *moderately ambiguous* since all morphism τ' with $\tau'(\alpha) = \sigma(\alpha)$ map every variable i_k with $\sigma(i_k) \neq \varepsilon$ to a certain factor w_k of $\sigma(i_k)$ at a particular position. In this example, we have $w_2 = w_5 = \mathtt{b}$ and $w_8 = w_{10} = \mathtt{bb}$ – and the only morphisms τ' with $\tau'(\alpha) = \sigma(\alpha)$ are σ itself and τ which satisfy $\sigma(i_k) = \ldots w_k \ldots = \tau(i_k)$ for $k = 2, 5, 8, 10$.

We now formalise moderate ambiguity. As explained above, we consider this a very natural way of slightly relaxing the requirement of unambiguity, and the importance of this approach has been demonstrated in the context of inductive inference of pattern languages. Nevertheless, our definition is quite involved, since we do not only postulate that, for a given pattern α and for every $x \in \mathrm{var}(\alpha)$, there exists a string w_x such that, for every morphism τ with $\tau(\alpha) = \sigma(\alpha)$, $\tau(x)$ contains w_x as a factor (which could be called *factor-preserving* ambiguity), but we also demand that these factors are located at fixed positions for all τ. This means that we need to identify and mark the positions of the factors.

Definition 5. *Let Σ be an alphabet, let $\alpha = i_1 \cdot i_2 \cdot [\ldots] \cdot i_n$ with $n, i_1, i_2, \ldots, i_n \in \mathbb{N}$, and let $\sigma : \mathbb{N}^* \to \Sigma^*$ be a morphism satisfying $\sigma(\alpha) \neq \varepsilon$. Then σ is called* moderately ambiguous (for α) *provided that there exist $l_2, l_3, \ldots, l_n, r_1, r_2, \ldots, r_{n-1} \in \mathbb{N} \cup \{0\}$ such that, for every morphism $\tau : \mathbb{N}^* \to \Sigma^*$ with $\tau(\alpha) = \sigma(\alpha)$,*

(i) *if* $\sigma(i_1) \neq \varepsilon$ *then* $r_1 \geq 1$,
(ii) *if* $\sigma(i_n) \neq \varepsilon$ *then* $l_n \leq |\sigma(\alpha)|$,
(iii) *for every* $k \in \{2, 3, \ldots, n-1\}$ *with* $\sigma(i_k) \neq \varepsilon$, $l_k \leq r_k$,
(iv) *for every* k *with* $1 \leq k \leq n-1$, $|\tau(i_1 \cdot i_2 \cdot [\ldots] \cdot i_k)| < l_{k+1}$, *and*
(v) *for every* k *with* $1 \leq k \leq n-1$, $|\tau(i_1 \cdot i_2 \cdot [\ldots] \cdot i_k)| \geq r_k$.

We call σ strongly ambiguous (for α) *if and only if it is not moderately ambiguous (for* α*).*

In the definition, for any pattern α and any moderately ambiguous morphism σ for α, a pair (l_k, r_k) for some $i_k \in \text{var}(\alpha)$ with $\sigma(i_k) \neq \varepsilon$ "marks" the factor w_k from position l_k to r_k in $\sigma(\alpha)$. This factor must be covered by the image of i_k under every morphism τ with $\tau(\alpha) = \sigma(\alpha)$ – this is guaranteed by the conditions (iv) and (v). Considering Example 1, we choose the following markers l_i, r_k: Let $r_1 := 0$, $(l_2, r_2) := (2, 2)$, $(l_5, r_5) := (5, 5)$, $(l_8, r_8) := (8, 9)$, $l_{10} := 11$ and finally $(l_k, r_k) := (|\sigma(\alpha)| + 1, 0)$ for $k \in \{1, 3, 4, 6, 7, 9\}$ since, for these k, $\sigma(i_k) = \varepsilon$, and, thus, no factor has to be marked. It can be verified that these values of l_j, $2 \leq j \leq n$, and r_k, $1 \leq k \leq n-1$ meet the requirements (i)–(v) of Definition 5.

The following lemma is useful when studying moderate ambiguity since, in certain cases, it circumvents a check of the minutiae of Definition 5.

Lemma 2. *Let* Σ *be an alphabet,* $\alpha \in \mathbb{N}^+$ *and* $\sigma : \mathbb{N}^* \to \Sigma^*$ *be a morphism. If there exists a morphism* $\tau : \mathbb{N}^* \to \Sigma^*$ *such that* $\tau(\alpha) = \sigma(\alpha)$*, but* $\tau(x) = \varepsilon \neq \sigma(x)$ *for an* $x \in \text{var}(\alpha)$*, then* σ *is not moderately ambiguous for* α*.*

As suggested by the definitions and further substantiated by Example 1, for any given morphism, the requirement of being moderately ambiguous is less strict than that of being unambiguous:

Proposition 1. *Let* Σ *be an alphabet, let* $\sigma : \mathbb{N}^* \to \Sigma^*$ *be a morphism, and let* $\alpha \in \mathbb{N}^+$*. If* σ *is unambiguous for* α*, then* σ *is moderately ambiguous for* α*. In general, the converse does not hold.*

This directly implies that if there exists no moderately ambiguous morphism for a pattern α, then there exists no unambiguous morphism for α and, thus, every morphism is strongly ambiguous for α.

With these new terms of ambiguity, we can give a stronger version of Theorem 2:

Theorem 6. *Let* Σ *be an alphabet. Let* $\alpha \in \mathbb{N}^+$ *and let* (E, N) *be an ambiguity partition for* α*. Then every morphism* $\sigma : \mathbb{N}^* \to \Sigma^*$ *satisfying* $\sigma(x) \neq \varepsilon$ *for an* $x \in E$ *is strongly ambiguous for* α*.*

The main result of our paper characterises those patterns that have a moderately ambiguous morphism. More precisely, it states that moderate ambiguity can be achieved if and only if the pattern is morphically unerasable:

Theorem 7. *Let* Σ *be an alphabet,* $|\Sigma| \geq 2$*, let* $\alpha \in \mathbb{N}^+$*. There exists a morphism* $\sigma : \mathbb{N}^* \to \Sigma^*$ *that is moderately ambiguous for* α *if and only if* α *is morphically unerasable.*

In addition to the facts that Theorem 7 provides an algorithmically verifiable characteristic condition on a vital problem regarding the existence of morphisms with a restricted ambiguity and, furthermore, implies the equivalent result for the weaker requirement of factor-preserving ambiguity, we consider two other aspects of it quite remarkable. Firstly, it confirms that ambiguity partitions are indeed a crucial tool when investigating the ambiguity of erasing morphisms, since they cannot only be used to give sufficient criteria on the subject (cf. Corollary 1) and characteristic criteria for special cases (cf. Theorem 4), but are also capable of expressing a key phenomenon in this field of study.

Secondly, it establishes a quite remarkable and counter-intuitive difference between the ambiguity of erasing and nonerasing morphisms. As demonstrated by Freydenberger et al. [1], the existence of a moderately ambiguous nonerasing morphism σ for a pattern implies the existence of an unambiguous nonerasing morphism σ'. More technically, it can be shown that σ can be turned into σ' by applying some minor yet sophisticated changes that depend on the structure of the pattern in question (see Reidenbach [9] for a detailed discussion of this topic). It is also important to note that the morphism σ and σ' both use a binary target alphabet; hence, the existence of such morphisms – which characterises the succinct patterns, cf. Theorem 1 – exclusively depends on the pattern and not on the size of Σ (provided that Σ contains at least two letters). In contrast to these observations, Theorem 7 demonstrates that the existence of moderately ambiguous erasing morphisms does not imply the existence of unambiguous erasing morphisms:

Corollary 2. *Let Σ be an alphabet. There exists an $\alpha \in \mathbb{N}^+$ and a morphism $\sigma : \mathbb{N}^* \to \Sigma^*$ such that σ is moderately ambiguous for α, but no morphism is unambiguous for α.*

Hence, the main result of our paper also shows that the technical concepts used by Freydenberger et al. [1] to turn a moderately ambiguous morphism into an unambiguous one necessarily fail for erasing morphisms. Since this insight is rather unexpected, it is also surprising that Theorem 7 is alphabet-independent, whereas any characterisation of the set of those patterns that have an unambiguous erasing morphism must depend on the size of Σ (as shown by Schneider and to be further addressed by Section 5).

We wish to conclude this section with an insight into the complexity of the problem of deciding on the existence of moderately ambiguous morphisms:

Corollary 3. *Let Σ be an alphabet, $|\Sigma| \geq 2$. The problem of deciding, for any given $\alpha \in \mathbb{N}^+$, on whether there is an erasing morphism $\sigma : \mathbb{N}^* \to \Sigma^*$ that is moderately ambiguous for α, is NP-complete.*

This nicely contrasts with the recent result by Holub [4], which implies that there is a polynomial-time procedure deciding on the existence of unambiguous nonerasing morphisms.

As briefly mentioned above, we now study another fundamental property of those patterns that can be used to prove Corollary 2.

5 Patterns with Finitely Many Unambiguous Morphisms

Once the existence of morphisms with a restricted ambiguity has been established for a given pattern, it is a natural problem to investigate the number of such morphisms. Since the existence of one moderately ambiguous morphism for a given pattern immediately implies an infinite number of such morphisms (the morphism used to prove Theorem 7 can easily be generalised), we now study the above-mentioned topic with regard to a maximal restriction of ambiguity, i. e. unambiguity. To this end, we introduce the following notation:

Definition 6. *Let Σ be an alphabet and $\alpha \in \mathbb{N}^+$. Then* $\mathrm{UNAMB}_{\Sigma}(\alpha)$ *is the set of all $\sigma(\alpha)$, where $\sigma : \mathbb{N}^* \to \Sigma^*$ is a morphism that is unambiguous for α, and* $\mathrm{UNAMB}_{\mathrm{NE},\Sigma}(\alpha)$ *is the set of all $\sigma(\alpha)$, where $\sigma : \mathbb{N}^* \to \Sigma^*$ is a morphism that is nonerasing and unambiguous for α.*

We wish to point out that the sets $\mathrm{UNAMB}_{\Sigma}(\alpha)$ and $\mathrm{UNAMB}_{\mathrm{NE},\Sigma}(\alpha)$ do not consist of morphisms, but of morphic images. This makes sure that all unambiguous morphisms indirectly collected by these sets necessarily differ on variables that are contained in $\mathrm{var}(\alpha)$.

We first consider the case of *nonerasing* morphisms.

Theorem 8. *Let $\alpha \in \mathbb{N}^+$. Then either, for all alphabets Σ with $|\Sigma| \geq 2$,* $\mathrm{UNAMB}_{\mathrm{NE},\Sigma}(\alpha)$ *is empty or, for all alphabets Σ with $|\Sigma| \geq 2$,* $\mathrm{UNAMB}_{\mathrm{NE},\Sigma}(\alpha)$ *is infinite.*

If we study the equivalent question for the ambiguity of *erasing* morphisms, we can observe a novel phenomenon that establishes a further difference to the case of nonerasing morphisms. More precisely, for certain patterns α, the cardinality of $\mathrm{UNAMB}_{\Sigma}(\alpha)$ can be finite, and this essentially depends on the size of Σ:

Theorem 9. *Let $k \in \mathbb{N}$. Let $\Sigma_k, \Sigma_{k+1}, \Sigma_{k+2}$ be alphabets with $k, k+1, k+2$ letters, respectively. There exists an $\alpha_k \in \mathbb{N}^+$ such that* $|\mathrm{UNAMB}_{\Sigma_k}(\alpha_k)| = 0$, $|\mathrm{UNAMB}_{\Sigma_{k+1}}(\alpha_k)| = m$ *for an $m \in \mathbb{N}$, and* $\mathrm{UNAMB}_{\Sigma_{k+2}}(\alpha_k)$ *is an infinite set.*

6 Connections to NE-Pattern Languages

In this final main section of our paper we wish to study a topic that, after the particularly strong result in Theorem 7, remains as the most fundamental open problem on erasing morphisms with a restricted ambiguity, namely a characterisation of those patterns that have an unambiguous erasing morphism. As a matter of fact, the main result of the present section can be understood as such a characterisation, but the immediate usefulness of the result is limited. Nevertheless, our examinations reveal some enlightening and rather counter-intuitive insights that might be useful for further investigations.

While the existence of a relation between the ambiguity of erasing morphisms and certain properties of E-pattern languages (as, e. g., demonstrated by Condition 1 and Theorem 5) is by no means surprising, our characterisation shall demonstrate likewise deep connections between the main subject of our paper and vital properties of *NE*-pattern languages. It reads as follows:

Theorem 10. *Let Σ be an alphabet, and let $\alpha \in \mathbb{N}^+$. For any partition (U, V) of $\mathcal{P}(\mathrm{var}(\alpha)) \setminus \{\emptyset\}$, let $L_{\alpha,U,V} := \bigcup_{u \in U} L_{\mathrm{NE},\Sigma}(\pi_u(\alpha)) \cap \bigcup_{v \in V} L_{\mathrm{NE},\Sigma}(\pi_v(\alpha))$. There is no unambiguous word in $L_{\mathrm{E},\Sigma}(\alpha) \setminus \{\varepsilon\}$ if and only if there is no unambiguous word in $L_{\mathrm{E},\Sigma}(\alpha) \setminus (\{\varepsilon\} \cup L_{\alpha,U,V})$.*

It is a noteworthy property of Theorem 10 that it covers the ambiguity of both erasing and nonerasing morphisms and, hence, allows a unified view on both topics. However, for the latter case, Theorem 1 already gives a definite answer, indirectly stating that, for every *succinct* pattern α, there is *no* partition (U, V) of $\mathcal{P}(\mathrm{var}(\alpha)) \setminus \{\emptyset\}$ such that every word in $L_{\mathrm{E},\Sigma}(\alpha) \setminus (\{\varepsilon\} \cup L_{\alpha,U,V})$ is ambiguous for α. Thus, we can completely concentrate on prolix patterns when investigating applicability and consequences of Theorem 10.

From a practical point of view, Theorem 10 is not too helpful yet, as it merely reduces the number of words that need to be examined with regard to their ambiguity. Thus, it cannot be seen as an applicable characterisation of those patterns that have an unambiguous erasing morphism. On the other hand, it constitutes a promising starting point for further research on that topic, asking how U and V have to be be chosen such that $L_{\alpha,U,V}$ has maximal size and what a maximal $L_{\alpha,U,V}$ looks like for a given α. In this regard, it is worth mentioning that example patterns α and sets $U, V \subseteq \mathcal{P}(\mathrm{var}(\alpha)) \setminus \{\emptyset\}$ can be given where $L_{\alpha,U,V}$ is a nonempty subset of $L_{\mathrm{E},\Sigma}(\alpha)$ or even equals $L_{\mathrm{E},\Sigma}(\alpha) \setminus \{\varepsilon\}$.

Since, for any pattern α, $L_{\mathrm{E},\Sigma}(\alpha)$ is equivalent to a finite union of NE-pattern languages (see Theorem 2.1 by Jiang et al. [5]), Theorem 10 shows that the existence of unambiguous *erasing* morphisms strongly depends on equivalence and inclusion of certain finite unions of *NE*-pattern languages (or *nonerasing multi-pattern languages*, as they are called by Kari et al. [7]). This is not only a rather counter-intuitive insight, but it also gives an idea of how difficult the problem of the existence of unambiguous erasing morphisms might be. More precisely, even the decidability of the inclusion problem for ordinary terminal-free NE-pattern languages is open and includes some prominent open problems on pattern avoidability (cf. [5]). The inclusion of terminal-free NE-pattern languages is also known to depend on the size of the target alphabet, which fits very well with what is known for the subject of our paper (see, e. g., Theorem 9).

The following sufficient condition illustrates how Theorem 10 can be used to find criteria on the nonexistence of unambiguous erasing morphisms:

Corollary 4. *Let Σ be an alphabet, and let $\alpha \in \mathbb{N}^+$. If there exists a partition (U, V) of $\mathcal{P}(\mathrm{var}(\alpha)) \setminus \{\emptyset\}$ with $L_{\mathrm{E},\Sigma}(\alpha) \setminus \{\varepsilon\} = \bigcup_{u \in U} L_{\mathrm{NE},\Sigma}(\pi_u(\alpha)) = \bigcup_{v \in V} L_{\mathrm{NE},\Sigma}(\pi_v(\alpha))$, then there is no unambiguous word in $L_{\mathrm{E},\Sigma}(\alpha) \setminus \{\varepsilon\}$.*

We finally wish to mention that Theorem 10 and Corollary 4 do not need to be based on a *partition* (U, V) of $\mathcal{P}(\mathrm{var}(\alpha)) \setminus \{\emptyset\}$. Alternatively, they could refer to *arbitrary* disjoint subsets U and V of $\mathcal{P}(\mathrm{var}(\alpha)) \setminus \{\emptyset\}$.

7 Conclusion and Open Problems

Concerning the ambiguity of erasing morphisms, the partition of patterns into morphically unerasable and erasable patterns (introduced and studied in

Section 3) has similar importance as the partition into succinct and prolix patterns regarding the ambiguity of nonerasing morphisms: Both partitions characterise a vital property of strings, namely the (non-)existence of moderately ambiguous morphisms (cf. Theorem 7 and Reidenbach [9]). While, in the case of nonerasing morphisms, this restricted ambiguity can additionally be turned into unambiguity, this does not hold for erasing morphisms since their ambiguity essentially depends on the size of the target alphabet (cf. Corollary 2 and, featuring a rather unexpected insight, Theorem 9).

A characterisation of those patterns that have an unambiguous erasing morphism is the main remaining open problem on the subject of the present paper, and even its mere decidability is still unresolved. Due to the insights summarised above, it seems evident that any solution to it requires concepts that significantly differ from the techniques used regarding moderate ambiguity. Section 6 reveals fundamental and quite surprising connections between the ambiguity of erasing morphisms and decision problems for nonerasing multi-pattern languages. An examination of these topics might be a helpful starting point for future studies.

References

[1] Freydenberger, D.D., Reidenbach, D., Schneider, J.C.: Unambiguous morphic images of strings. International Journal of Foundations of Computer Science 17, 601–628 (2006)

[2] Harju, T., Karhumäki, J.: Morphisms. In: [12], ch. 7, pp. 439–510

[3] Head, T.: Fixed languages and the adult languages of 0L schemes. International Journal of Computer Mathematics 10, 103–107 (1981)

[4] Holub, Š.: Polynomial-time algorithm for fixed points of nontrivial morphisms. Discrete Mathematics 309, 5069–5076 (2009)

[5] Jiang, T., Kinber, E., Salomaa, A., Salomaa, K., Yu, S.: Pattern languages with and without erasing. International Journal of Computer Mathematics 50, 147–163 (1994)

[6] Jiang, T., Salomaa, A., Salomaa, K., Yu, S.: Decision problems for patterns. Journal of Computer and System Sciences 50, 53–63 (1995)

[7] Kari, L., Mateescu, A., Păun, G., Salomaa, A.: Multi-pattern languages. Theoretical Computer Science 141, 253–268 (1995)

[8] Mateescu, A., Salomaa, A.: Patterns. In: [12], ch. 4.6, pp. 230–242

[9] Reidenbach, D.: The Ambiguity of Morphisms in Free Monoids and its Impact on Algorithmic Properties of Pattern Languages. Logos Verlag, Berlin (2006)

[10] Reidenbach, D.: Discontinuities in pattern inference. Theoretical Computer Science 397, 166–193 (2008)

[11] Reidenbach, D., Schneider, J.C.: Morphically primitive words. Theoretical Computer Science 410, 2148–2161 (2009)

[12] Rozenberg, G., Salomaa, A.: Handbook of Formal Languages, vol. 1. Springer, Berlin (1997)

[13] Schneider, J.C.: Unambiguous erasing morphisms in free monoids. Theoretical Informatics and Applications (RAIRO) 44, 193–208 (2010)

Automata with Extremal Minimality Conditions

Antonio Restivo and Roberto Vaglica

Dipartimento di Matematica e Informatica,
Università degli Studi di Palermo
Via Archirafi 34, 90123 Palermo, Italy
{restivo,vaglica}@math.unipa.it

Abstract. It is well known that the minimality of a deterministic finite automaton (DFA) depends on the set of final states. In this paper we study the minimality of a strongly connected DFA by varying the set of final states. We consider, in particular, some extremal cases. A strongly connected DFA is called *uniformly minimal* if it is minimal, for any choice of the set of final states. It is called *never-minimal* if it is not minimal, for any choice of the set of final states. We show that there exists an infinite family of uniformly minimal automata and that there exists an infinite family of never-minimal automata. Some properties of these automata are investigated and, in particular, we consider the complexity of the problem to decide whether an automaton is uniformly minimal or never-minimal.

1 Introduction

It is well known that the minimization problem of deterministic finite automata (*DFAs*) is related to the indistinguishability notion of states (cf. [12]). Indeed, a well known technique to minimize a DFA, essentially, consists in finding pairs of states that are equivalent (or *indistinguishable*), namely pairs of states (p, q) such that it is impossible to assert the difference between p and q only by starting in each of the two states and asking whether or not a given input string leads to a final state. Since, in the testing states equivalence, the notion of initial state is irrelevant, some of the main techniques for minimization of automata, such as Moore's algorithm [15] and Hopcroft's algorithm [11], do not care what is the initial state of the automaton provided that all states can be reached from the initial state. Therefore a natural question that arises is, for these automata, on what does minimality depend? Obviously, it depends on both the automata transitions and the set of final states. In this paper, our main focus is to investigate to what extent minimality depends on the choice of the subset of final states. So we consider DFAs of the form $\mathcal{A} = (Q, \Sigma, \delta)$, where initial and final states are not specified, and such that the state graphs are strongly connected. This last condition is required since we are interested to study the minimality of such automata by varying the subset of final states: by the hypothesis of strong connectivity we avoid that there are states not accessible or not coaccessible. Note that such automata occur also in the study of the synchronization problem and the Černý's conjecture (cf. [4,17]), or in Symbolic Dynamics (cf. [14,1]).

In order to investigate the dependence of the minimality of the automaton $\mathcal{A}$ on the choice of final states, we introduce the state-pair graph $G(\mathcal{A})$. The choice of a set F of final states defines a coloring γ_F of $G(\mathcal{A})$, and we prove that the minimality of $\mathcal{A}$ corresponds, essentially, to a property of the colored graph. In this way, in order to check whether $\mathcal{A}$ is minimal with respect to various sets of final states, we need to compute the graph $G(\mathcal{A})$ only once, and then analyze the various cases by considering the colorings on $G(\mathcal{A})$ corresponding to the various choices of final states.

We next consider some extremal cases. We introduce the family of *uniformly minimal* automata, i.e. (non-necessary complete) automata which are minimal for any choice of the set of final states. We provide a characterization of such a family of automata in term of the state-pair graph, from which one derives a polynomial algorithm to decide whether a given DFA is uniformly minimal. Another characterization shows interesting relations between uniformly minimal automata, *multi-entry automata* (cf. [10]) and Symbolic Dynamics (cf. [14,1]). In the case the automaton is complete, the notion of uniform minimality turns out to be trivial. So we introduce, for complete automata, the weaker notion of almost uniform minimality and we show that there exists an infinite family of almost uniformly minimal automata. However, contrary to the previous case, we do not know, at present, a polynomial algorithm to check whether a complete automaton is almost uniformly minimal. Later we consider the opposite extremal case, i.e. automata that are *never-minimal*, for any choice of the set of final states. Also in this case we prove that there exists an infinite family of never-minimal automata. We show a sufficient condition for an automaton to be never-minimal, and we leave as an open problem whether such a condition is necessary too. An affirmative answer could provide a polynomial algorithm for checking whether an automaton is never-minimal.

Finally we consider the case of unary alphabet. We show that there do not exist never-minimal automata and we give a complete characterization of almost uniformly minimal automata.

As a concluding remark, we show some relations between the problem to decide whether an automaton is never-minimal and the "syntactic monoid problem" (cf. [8]).

2 Preliminaries

In this section we give some basic notations and terminology concerning finite automata and refer the reader to the literature for more details (cf. [5,12]). A deterministic finite automaton, or DFA, $\mathcal{A} = (Q, \Sigma, \delta)$ is defined by specifying a finite *state set* Q, a finite input alphabet Σ, and a (partial) *transition function* $\delta : Q \times \Sigma \rightarrow Q$. The action of the letters in Σ on the states in Q can be extended in a natural way to Σ^*, where Σ^* is the free monoid over the alphabet Σ; this extension is here denoted by δ^*. An automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is said *strongly connected* if for every ordered pair of states $q, q' \in Q$ there exists $w \in \Sigma^*$ such that $\delta^*(q, w) = q'$. An automaton is *complete* when its transition function is total. A DFA with a partial transition function can be transformed into a complete one by adding a single new state, the *sink* state usually denoted by s, and transitions from all other states to that sink for all symbols for which they have no transitions. The sink itself is made complete by adding loop transitions for all symbols. The automaton obtained in this way is called the *completion* of $\mathcal{A}$ and is denoted by $\hat{\mathcal{A}} = (\hat{Q}, \Sigma, \hat{\delta})$. It is clear that an automaton with a sink state is not strongly connected.

Let $\mathcal{A} = (Q, \Sigma, \delta)$ a DFA. If we designate a certain state $i \in Q$ as *initial* state, and a non-empty subset $F \subseteq Q$ as set of *final* (or *accepting*) states, then we say that the DFA, that we now denote by $\mathcal{A}_{i,F}$, *recognizes* a language. The language recognized by $\mathcal{A}_{i,F}$ is the set $L(\mathcal{A}_{i,F}) = \{w \in \Sigma^* : \delta^*(i, w) \in F\}$. Two automata that recognize the same language are called *equivalent*. Finally a DFA is *minimal* if it has the minimum number of states among all its equivalent DFAs. For any finite deterministic automaton $\mathcal{A}_{i,F}$ there is a unique (up to labeling of the states) minimal automaton that recognizes the same language as the automaton $\mathcal{A}_{i,F}$. As already mentioned in the introduction, the minimal automaton equivalent to a given DFA, can be computed essentially by using the *indistinguishable equivalence I*. More precisely, we say that two states p and q are indistinguishable if, for all input strings w, $\delta^*(p, w) \in F$ iff $\delta^*(q, w) \in F$. One can prove that this is an equivalence relation compatible with the transitions of the automaton, i.e. for any $a \in \Sigma$, pIq implies $\delta(p, a)I\delta(q, a)$. A state is *accessible* (resp. *coaccessible*) if there is a path from the initial state to this state (resp. from this state to a final state). Hence, a schematic description of the minimization algorithm consists of two steps: first, eliminate states that are not accessible, then merge states that are equivalent (cf. [12]). With the sole object of answering the question if a given DFA is minimal, free from the choice of the initial state, in this paper we consider strongly connected DFAs in which the initial state is not specified. On the other hand, we want to investigate the connections between minimality and the particular choice of the set of final states. Thus, in our objects of study, also the set of final states is not specified.

3 State-Pair Graph

In this section we introduce the *state-pair graph* of an automaton, that is a tool that turned out to be useful in our investigations.

Definition 1. *Given a deterministic finite automaton $\mathcal{A} = (Q, \Sigma, \delta)$, the state-pair graph of $\mathcal{A}$ is the graph $\mathcal{G}(\mathcal{A}) = (V_G, E_G)$ defined as follows:*

i. the set V_G of vertices consists of all unordered pairs of distinct states of $\mathcal{A}$;
ii. $E_G = \{((p, q), (p', q')) \mid \delta(p, a) = p', \delta(q, a) = q'$ and $a \in \Sigma\}$.

Figure 1 illustrates a DFA and the corresponding state-pair graph.

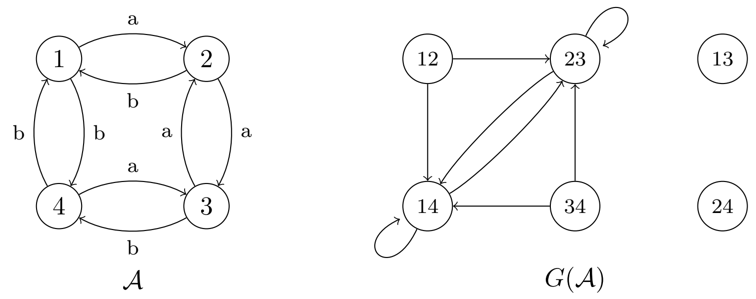


Fig. 1. A DFA $\mathcal{A}$ and the corresponding state-pair graph $G(\mathcal{A})$

As regards the complexity of the state-pair graph $\mathcal{G}(\mathcal{A})$, one can verify that: if $\mathcal{A}$ has n states, then

$$| V_G |= \binom{n}{2}$$

and

$$| E_G |\leq| \Sigma | \cdot | V_G | \, .$$

A similar tool has already been used in the literature (cf. [2,3,6,9]). However, besides being used in a different topic, it has here some remarkable differences, especially in the definition of the set of vertices V_G.
Given a set $F \subseteq Q$ of final states for the automaton $\mathcal{A}$, we associate with F a map

$$\gamma_F : V_G \to \{B, W\}$$

from the set of vertices V_G of the state-pair graph $G(\mathcal{A})$ into the set of colors $\{B, W\}$, defined as follows: for any $(p, q) \in V_G$,

$$\gamma_F(p,q) = \begin{cases} B & \text{if } p \in F \text{ and } q \notin F\text{, or vice versa;} \\ W & \text{otherwise.} \end{cases}$$

Thus, we have a non bijective mapping that assigns to every set $F \subseteq Q$ of final states a coloring of $G(\mathcal{A})$. Figure 2 illustrates two colorings, for the state-pair graph of the automaton depicted in Fig. 1, related to two different sets of final states. An automaton $\mathcal{A}_{i,F}$ is said to be *trim* if all its states are both accessible and coaccessible. If we call *distinguishable* a pair of states that are not equivalent with respect to the indistinguishability relation, we have that a trim DFA $\mathcal{A}_{i,F}$ is minimal iff every pair of its states is distinguishable. In other words, $\mathcal{A}_{i,F}$ is minimal if and only if for every two states p and q there is at least one string w such that one of $\delta^*(p, w)$ and $\delta^*(q, w)$ is final and the other is not. This fact together with the crucial property of the state-pair graph that, if there is an input string w such that $\delta^*(q, w) = q'$ and $\delta^*(p, w) = p'$ where $q, q', p, p' \in Q$ and $p' \neq q'$, then there is a path from (p, q) to (p', q') in $G(\mathcal{A})$, leads to the following result.

Theorem 1. *Let $\mathcal{A} = (Q, \Sigma, \delta)$, $i \in Q$ and $F \subseteq Q$ such that $\mathcal{A}_{i,F}$ is a trim DFA. Then $\mathcal{A}_{i,F}$ is minimal iff for every vertex v of $G(\hat{\mathcal{A}})$ with $\gamma_F(v) = W$, there is a path, in $G(\hat{\mathcal{A}})$, from v to a vertex v' such that $\gamma_F(v') = B$.*

The remarkable thing about the use of the state-pair graph is that to check whether a DFA $\mathcal{A}$ is minimal for some initial state i and various sets F of final states, we need to compute $G(\hat{\mathcal{A}})$ only once. Then, for any $F \subseteq Q$, we have to check if $\mathcal{A}_{i,F}$ is trim and consider the various colorings on $G(\hat{\mathcal{A}})$ corresponding to the sets of final states.

In particular, since the automaton $\mathcal{A}$ depicted in Fig. 1 is complete, then $G(\mathcal{A}) = G(\hat{\mathcal{A}})$. Thus from Fig. 2 we note that $\mathcal{A}_{i,F}$, for any $i \in \{1, ..., 4\}$, is minimal if we choose $F = \{1, 2\}$ as set of final states, but it isn't with respect to $F' = \{1, 3\}$.

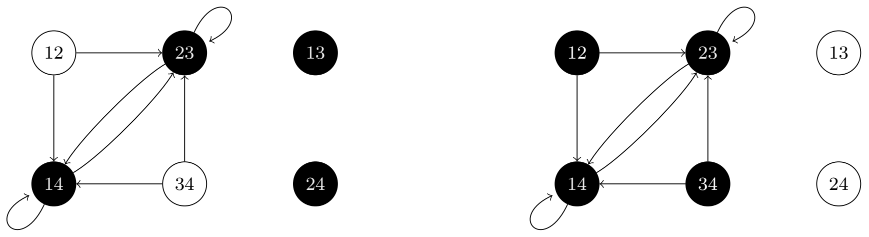


Fig. 2. Two colorings of $G(\mathcal{A})$ relate to the sets $F = \{1, 2\}$ and $F' = \{1, 3\}$, respectively. Vertices that are mapped by γ_F to the element B are drawn in black.

4 Uniformly Minimal Automata

In the previous section we have observed that, in general, an automaton can be minimal with respect to some set of final states but not with respect to others. At this point, the following question arises: do there exist minimal automata whose minimality is not affected by the choice of the final states?

Remark 1. It is easy to see that, given a DFA $\mathcal{A} = (Q, \Sigma, \delta)$, $A_{i,F}$ is trim for some $i \in Q$ and for all $F \subseteq Q$ if and only if $\mathcal{A}$ is strongly connected. Thus, the above question makes sense only if we consider strongly connected automata.

A strongly connected automaton $\mathcal{A} = (Q, \Sigma, \delta)$ is called *uniformly minimal* if, for all $F \subseteq Q$, it is minimal. Remark that, if $\mathcal{A}$ is complete and $F = Q$, then $\mathcal{A}$ is minimal only if it corresponds to the trivial automaton with only one state. So a nontrivial uniformly minimal automaton is not complete. The following lemma provides a characterization of a uniformly minimal DFA $\mathcal{A}$ in terms of the state-pair graph $G(\hat{\mathcal{A}})$ of the completion of $\mathcal{A}$.

Lemma 1. *A strongly connected (incomplete) automaton $\mathcal{A}$ is uniformly minimal if and only if, for any vertex v in the state-pair graph $G(\hat{\mathcal{A}})$, there is a path from v to a vertex of the form (q, s), where s is the sink state.*

Remark 2. As a consequence of the lemma, one obtains a polynomial algorithm to decide whether a given DFA is uniformly minimal.

The next theorem provides another characterization of uniformly minimal automata. In order to state this theorem, it is useful to introduce the following notation. Let $\mathcal{A} = (Q, \Sigma, \delta)$ a deterministic finite automaton. If $I, F \subseteq Q$, denote by $\mathcal{A}(I, F)$ the (non deterministic) automaton

$$\mathcal{A}(I, F) = (Q, \Sigma, \delta, I, F),$$

where I is a set of initial states and F is a set of final states. If $\mid I \mid \leq k$, $\mathcal{A}(I, F)$ is called a *k-entry DFA* (cf. [10]). The language recognized by a k-entry DFA $\mathcal{A}(I, F)$ is the set of words w such that w corresponds to a path in $\mathcal{A}$ from some state of I to some state of F. Denote by $L(\mathcal{A}(I, F))$ such a language. A k-entry DFA $\mathcal{A}$ is minimal if it has a minimum number of states among all k-entry DFAs recognizing $L(\mathcal{A}(I, F))$. Remark that, given a regular language L, the minimal k-entry DFA recognizing L is not, in general, unique. Moreover the minimization problem for k-entry DFAs is PSpace-complete (cf. [10]). By using previous notation, one has that a strongly connected DFA $\mathcal{A} = (Q, \Sigma, \delta)$ is uniformly minimal if $\mathcal{A}(\{q\}, F)$ is minimal for some $q \in Q$ and for all $F \subseteq Q$.

Theorem 2. *Let $\mathcal{A} = (Q, \Sigma, \delta)$ a strongly connected DFA. The following conditions are equivalent:*

1. *$\mathcal{A}(\{q\}, F)$ is minimal for some $q \in Q$ and for all $F \subseteq Q$, i.e. $\mathcal{A}$ is uniformly minimal.*
2. *$\mathcal{A}(\{q\}, F)$ is minimal for all $q \in Q$ and for all $F \subseteq Q$.*
3. *$\mathcal{A}(\{q\}, Q)$ is minimal for some $q \in Q$.*
4. *$\mathcal{A}(\{q\}, Q)$ is minimal for all $q \in Q$.*
5. *$\mathcal{A}(I, F)$ is minimal for all $I \subseteq Q$ and for all $F \subseteq Q$.*
6. *$\mathcal{A}(Q, Q)$ is minimal.*

Proof. The equivalence between 1) and 2), and the equivalence between 3) and 4), are trivial consequences of the strongly connectedness of $\mathcal{A}$ and of the notion of minimal automaton.

The implication 1) $\Rightarrow$ 3) is trivial, whereas the implication 3) $\Rightarrow$ 1) is a consequence of Lemma 1. Actually, note that the vertices of the form (q, s), where s is the sink state, form a strongly connected component of $G(\hat{\mathcal{A}})$.

The implication 3) $\Rightarrow$ 6) is a consequence of a result in Symbolic Dynamics. More precisely, it is well known that, given a language L recognized by a strongly connected automaton $\mathcal{A}(Q, Q)$, the minimal deterministic automaton, recognizing L, among the automata in which all states are both initial and final, is unique up to the labeling of the states. Moreover, this automaton (called *Fisher cover* in Symbolic Dynamics) can be obtained from $\mathcal{A}$ by merging the indistinguishable states, like in the minimization algorithm for DFA (cf. Prop. 3.3.9 and Theorem 3.3.18 of [14]). Now, suppose there exists a trim automaton

$\mathcal{A}'(Q', \Sigma, \delta')$, with $\mid Q' \mid < \mid Q \mid$, such that $L(\mathcal{A}'(I', F')) = L(\mathcal{A}(Q, Q))$. Observe that, for any language L recognized by a k-entry trim automaton $\mathcal{A}(I, F)$, $L(\mathcal{A}(Q, Q)) = F(L)$, where $F(L)$ denotes the set of the factors of all words of L. It follows that $L(\mathcal{A}'(I', F')) = F(L(\mathcal{A}'(I', F'))) = L(\mathcal{A}'(Q', Q'))$, hence $\mathcal{A}' = \mathcal{A}$.

The implication 5) $\Rightarrow$ 1) is trivial.

Finally, for the implication 6) $\Rightarrow$ 5), assume by contradiction that there exists $\mathcal{A}' = (Q', \Sigma, \delta')$ such that $L(\mathcal{A}'(I', F')) = L(\mathcal{A}(I, F))$ with $\mid I' \mid \leq \mid I \mid$ and $\mid Q' \mid < \mid Q \mid$. Since $L(\mathcal{A}(Q, Q)) = L(\mathcal{A}'(Q', Q')) = F(L(\mathcal{A}(I, F)))$, then, from the minimality of $\mathcal{A}(Q, Q)$, we deduce $Q = Q'$, a contradiction. $\square$

Remark 3. As a consequence of Theorem 2, uniformly minimal automata correspond to Fisher covers of irreducible sofic shifts in Symbolic Dynamics. Thus, there are infinitely many uniformly minimal automata.

We have observed that the unique uniformly minimal *complete* automaton is the trivial automaton with only one state. Therefore, in the case of complete automata, one introduces a notion which is weaker than uniform minimality. A complete DFA $\mathcal{A} = (Q, \Sigma, \delta)$ is *almost uniformly minimal* if, for all *proper* subsets $F \subset Q$, it is minimal.

Theorem 3. *There is an infinite sequence of almost uniformly minimal DFAs* $(\mathcal{M}_n)_{n \geq 3}$*, where n is the number of states.*

This assertion can be verified by considering the automaton $\mathcal{M}_n = (Q, \Sigma, \delta)$, where $Q = \{1, 2, ..., n\}$, $n \geq 3$, $\Sigma = \{a, b\}$ and the transition defined as follows. For input a we have

$$\delta(i, a) = \begin{cases} \text{i+1, if } 1 \leq i < n; \\ 1, \quad \text{if } i = n. \end{cases}$$

As regards input b, if n is an even number

$$\delta(i, b) = \begin{cases} \text{i}, \quad \text{for } i \in \{1, n\}; \\ \text{i+1, if } i = 2k \text{ for } 1 \leq k \leq \frac{n}{2} - 1; \\ \text{i-1, if } i = 1 + 2k \text{ for } 1 \leq k \leq \frac{n}{2} - 1; \end{cases}$$

else

$$\delta(i, b) = \begin{cases} \text{i}, \quad \text{for } i \in \{1, n\}; \\ \text{i}, \quad \text{if } i = 2k \text{ for } \frac{n+1}{4} \leq k \leq \frac{n+3}{4}; \\ \text{i+1, if } i = 2k \text{ for } 1 \leq k < \frac{n+1}{4}; \\ \text{i-1, if } i = 1 + 2k \text{ for } 1 \leq k < \frac{n+1}{4}; \\ \text{i+1, if } i = n - 2k \text{ for } 1 \leq k \leq \frac{n-3}{4}; \\ \text{i-1, if } i = n + 1 - 2k \text{ for } 1 \leq k \leq \frac{n-3}{4}. \end{cases}$$

See Fig. 3 for the state transition diagram of $\mathcal{M}_5$.

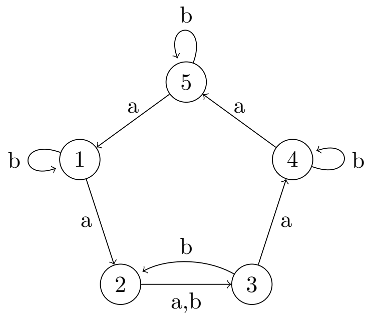


Fig. 3. The automaton $\mathcal{M}_5$

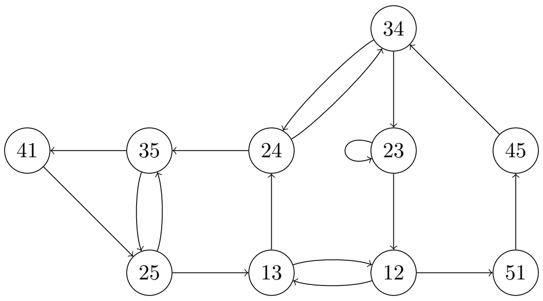


Fig. 4. The state-pair graph $G(\mathcal{M}_5)$

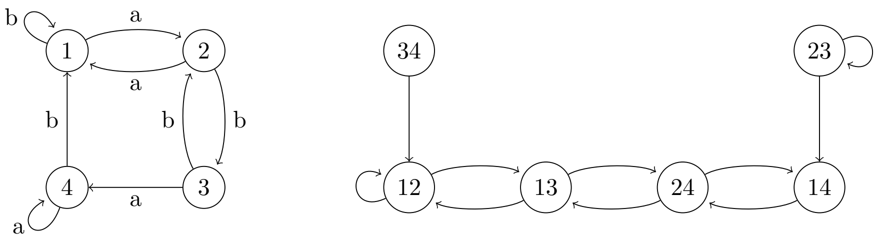


Fig. 5. An almost uniformly minimal automaton $\mathcal{A}$ whose state-pair graph $G(\mathcal{A})$ is not strongly connected

Proposition 1. *The state-pair graph $G(\mathcal{M}_n)$ is strongly connected.*

Fig. 4 shows the state-pair graph of $\mathcal{M}_5$. From Proposition 1 we have that, for any $F \subset Q$, there exists at least one vertex v in $G(\mathcal{M}_n)$ such that $\gamma_F(v) = B$. So, from Theorem 1, it follows that $\mathcal{M}_n$ is almost uniformly minimal.

Remark 4. In previous example the state-pair graph is strongly connected. Note however that, in general, the strong connectedness of the state-pair graph of a complete automaton is not a necessary condition for the almost uniform minimality. An example is shown in Fig. 5. One can check that, in that case, the automaton is almost uniformly minimal but its state-pair graph is not strongly connected. Obviously, if this condition was necessary, then we could have a polynomial time algorithm to test whether a complete automaton is almost uniformly minimal. However at present, a polynomial-time algorithm to solve the uniform minimality problem for complete automata is not known.

5 Never-Minimal Automata

In this section we consider the opposite extremal case. In particular we ask whether there exist strongly connected DFAs which aren't minimal for any choice of their final states. A positive answer to the question is given by next theorem. We call *never-minimal* a strongly connected DFA which isn't minimal for any choice of their final states.

Theorem 4. *There is an infinite sequence of never-minimal automata* $(\mathcal{N}_n)_{n\geq 4}$*, where* n *is the number of states.*

Proof. Let $n \geq 4$. Consider the automaton $\mathcal{N}_n = (Q, \Sigma, \delta)$ where $Q=\{1, 2, ..., n\}$, $\Sigma = \{a, b\}$ and the transition function is given by

$$\delta(i,a) = \begin{cases} 1, & \text{if } i \leq 3; \\ \text{i-1}, & \text{if } 4 \leq i \leq n. \end{cases}$$

$$\delta(i,b) = \begin{cases} 4, & \text{if } i \leq 3; \\ \text{i+1}, & \text{if } 3 < i \leq n-1; \\ 2, & \text{if } i = n. \end{cases}$$

The state graph of $\mathcal{N}_6$ is depicted in Fig. 6.

We suppose by contradiction that there exists a set F of final states that makes $\mathcal{N}_n$ a minimal DFA. We observe that from the vertices $(1, 2)$, $(1, 3)$ and $(2, 3)$ of $G(\mathcal{N}_n)$ there aren't outgoing edges. Thus from Theorem 1 it follows $\gamma_F(1, 2) = \gamma_F(1, 3) = \gamma_F(2, 3) = B$. Without loss of generality, assume $1 \in F$. If we look at the pair $(1, 2)$, we have that $2 \notin F$. Analogously, since $\gamma_F(2, 3) = B$,

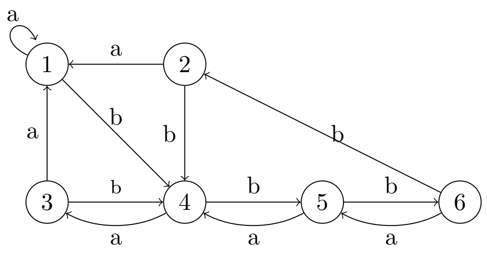


Fig. 6. The automaton $\mathcal{N}_6$

we have that $3 \in F$. However the conditions $1 \in F$ and $3 \in F$ contradict the hypothesis $\gamma_F(1,3) = B$. □

We now consider the problem to characterize never-minimal automata. Let $\mathcal{A} = (Q, \Sigma, \delta)$ a DFA. For $a \in \Sigma$, denote by δ_a the application $\delta_a : Q \to Q$ defined as follows: $\delta_a(q) = \delta(q, a)$, for all $q \in Q$.

Definition 2. *We say that a DFA $\mathcal{A} = (Q, \Sigma, \delta)$ satisfies condition C_h if there is $Q_h \subseteq Q$, with $\mid Q_h \mid= h$, such that, for all $a \in \Sigma$, the restriction of δ_a to Q_h is a constant or an identity function.*

Theorem 5. *Let $\mathcal{A} = (Q, \Sigma, \delta)$ a DFA. If $\mathcal{A}$ satisfies C_3 then it is never-minimal.*

This assertion can be proved by using a similar argument as that in the proof of Theorem 4. Indeed, we have only to say that if $Q_3 = \{p, q, r\}$, then the vertices of $G(\mathcal{A})$ that involve states belonging to Q_3 are in the same situation of vertices $(1, 2), (2, 3)$ and $(1, 3)$ of the proof of Theorem 4.

Remark 5. The previous theorem shows that condition C_3 is a sufficient condition for an automaton to be never-minimal.

Now a natural and interesting question arises: for strongly connected DFA, is condition C_3 also necessary? If the answer were affirmative, one could derive a polynomial-time algorithm for testing if a strongly connected DFA is never-minimal. Unfortunately, however, this problem is still open:

Open problem: *Does every never-minimal automaton satisfy condition C_3?*

6 Automata over a Unary Alphabet

The situation for automata over a unary alphabet is slightly different from that for automata on larger alphabet. First of all, note that a strongly connected DFA $\mathcal{A} = (Q, \Sigma, \delta)$ over a one letter alphabet $\Sigma = \{\sigma\}$ is simply a *cyclic* automaton. The corresponding state-pair graph, although similar to those of an element of the family $(\mathcal{M}_n)_{n \geq 3}$, consists of separated cyclic components. Moreover, it can be seen that for each $q \in Q$ there is at least one vertex in any cyclic component of $G(\mathcal{A})$ that contains q. It follows that $\mathcal{A}$ is minimal for every choice of the set of final states F with $\mid F \mid= 1$. Consequently, in the case of one letter alphabet, there do not exist never-minimal automata.

As regards the family of uniformly minimal automata, we have the following result.

Theorem 6. *Let $\mathcal{A} = (Q, \{\sigma\}, \delta)$ be a cyclic DFA with $\mid Q \mid= n$. $\mathcal{A}$ is almost uniformly minimal if and only if n is a prime number.*

Proof. If $n = hk$ for some natural numbers h and k, with regard to the set of final states we may consider the set $F = \{q_1, ..., q_h\}$ such that

$$\delta^*(q_i, \sigma^k) = \begin{cases} q_{i+1}, & \text{if } i \in \{1, ..., h-1\}; \\ q_1, & \text{if } i = h. \end{cases}$$

Thus, if we consider as initial state a state of F, it follows that $\mathcal{A}$ recognizes the set of all words over $\{\sigma\}$ whose length is a multiple of k. However, it is easy to see that this language can be recognized also by a DFA over $\{\sigma\}$ with k states. Now, suppose that n is a prime number and let F be a set of final states with cardinality $m < n$. For any initial state, $L(\mathcal{A})$ is given by all words over $\{\sigma\}$ whose length belongs to the union of exactly m equivalence classes modulo n. Since n is prime, this resulting set of integer numbers cannot be equal to the union of classes modulo different integers. It follows that $L(\mathcal{A})$ cannot be recognized by a DFA with less than p states, hence the thesis. □

7 Concluding Remarks

The problem to decide whether a strongly connected DFA is never-minimal is related to the "syntactic monoid problem". If M is a finite monoid and P a subset of M, there is a largest congruence σ_P saturating P defined by:

$$x\sigma_P y \;\Leftrightarrow\; \forall s,t \in M\ (sxt \in P \;\Leftrightarrow\; syt \in P).$$

The set P is called *disjunctive* if σ_P is the equality in M. A monoid M is *syntactic* if it has a disjunctive subset. The syntactic monoid problem is to decide whether a finite monoid is syntactic (cf. [8]). It is an open problem whether the syntactic monoid problem is polynomial or not. Now, if a monoid M is the transition monoid of a DFA $\mathcal{A}$, and M is not syntactic, then $\mathcal{A}$ is a never-minimal automaton. It is not clear whether such implication could be reversed. In any case, a positive solution of the open problem in Sect. 5 gives, as a consequence, a polynomial algorithm for deciding whether a DFA is never-minimal, and this could provide some insight on the syntactic monoid problem.

References

1. Beal, M.P., Perrin, D.: Symbolic dynamics and finite automata. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 2, pp. 463–505. Springer, Heidelberg (1997)
2. Berstel, J., Perrin, D.: Theory of codes. Academic Press, Inc., London (1985)
3. Brzozowski, J.: On Single-Loop Realizations of Sequential Machines. Information and Control 10(3), 292–314 (1967)
4. Carpi, A., D'Alessandro, F.: Strongly transitive automata and the Černý conjecture. Acta Informatica 46(8), 591–607 (2009)
5. Eilenberg, S.: Automata, Languages, and Machines, vol. A (1974)
6. Friedman, A.D.: Feedback in Synchronous Sequential Switching Circuits. IEEE Trans. Electronic Computers EC-15(3), 354–367 (1966)
7. Gill, A., Kou, L.T.: Multiple-entry finite automata. Journal of Computer and System Sciences 9(1), 1–19 (1974)
8. Goralcik, P., Koubek, V.: On the Disjunctive Set Problem. Theoretical Computer Science 204(1-2), 99–118 (1998)
9. Han, Y.S., Wang, Y., Wood, D.: Infix-free Regular Expressions and Languages. International Journal of Foundations of Computer Science 17(2), 379–393 (2006)

10. Holzer, M., Salomaa, K., Yu, S.: On the state complexity of k-entry deterministic finite automata. Journal of Automata, Languages and Combinatorics 6(4), 453–466 (2001)
11. Hopcroft, J.E.: An $n \log n$ algorithm for minimizing the states in a finite automaton. In: Kohavi, Z., Paz, A. (eds.) Theory of Machines and Computations, Proc. Internat. Sympos. Technion, Haifa, pp. 189–196. Academic Press, New York (1971)
12. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison-Wesley, Reading (1979)
13. Kao, J.Y., Rampersd, N., Shallit, J.: On NFA's where all states are final, initial, or both. TCS 410, 5010–5021 (2009)
14. Lind, D., Marcus, B.: An introduction to symbolic dynamics and coding. Cambridge University Press, New York (1995)
15. Moore, E.F.: Gedanken-experiments on sequential machines. The Journal of Symbolic Logic 23(1) (1958)
16. Veloso, P.A.S., Gill, A.: Some remarks on multiple-entry finite automata. Journal of Computer and System Sciences 18, 304–306 (1979)
17. Volkov, M.V.: Syncronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)

On the Existence of Minimal β-Powers

Arseny M. Shur

Ural State University

Abstract. If all proper factors of a word u are β-power-free while u itself is not, then u is a minimal β-power. We consider the following general problem: for which numbers k, β, and p there exists a k-ary minimal β-power of period p? For the case $\beta \geq 2$ we completely solve this problem. If the number $\beta < 2$ is relatively "big" w.r.t. k, we show that any number p can be the period of a minimal β-power. Finally, for "small" β we describe some sets of forbidden periods and provide a numerical evidence that for $k \geq 9$ these sets are almost exhaustive.

Introduction

The study of repetition-free words and languages, which remains one of the central topics in combinatorics of words, is closely connected to the study of minimal repetitions. A word avoids a repetition (say, a square) if it contains no *minimal* squares. *Minimal* squares generate the monoid ideal which is the complement of the square-free language in the corresponding free monoid. To estimate the growth rate of a repetition-free language, one needs to list relatively short *minimal* repetitions (see, e.g., [5,12,16,17]), and so on. As one will see, the minimal repetitions are also closely connected to repetition-free circular words.

The repetitions studied in this paper are *fractional powers*. If a word w of length n has the minimal period p, then w is an $\frac{n}{p}$-power. Studying minimal repetitions, it is convenient to treat the notion of fractional power in a loose way: we say that w is a β-power for all β such that $\frac{n-1}{p} < \beta \leq \frac{n}{p}$. The β-power is *minimal*, if it contains no other β-powers. Thus, a word avoids β-powers if and only if it contains no minimal β-powers.

The general problem which we study here is to determine all triples (k, β, p) such that a minimal β-power with the period p exists over the k-letter alphabet. Note that a particular question, concerning the existence of the minimal squares over the ternary alphabet, was asked in the book [3]. In this paper we provide an answer in a much broader context.

It is decidable by brute force whether a given triple (k, β, p) generates a minimal power. It is also decidable, due to the proof of Dejean's conjecture (see [6,8,10,13]), whether a given pair (k, β) generates infinitely many minimal powers. Recall that this conjecture states that the set of k-ary words avoiding β-powers is infinite if and only if $\beta > RT(k)$, where the *repetition threshold* $RT(k)$ equals $\frac{7}{4}$ for $k = 3$, $\frac{7}{5}$ for $k = 4$, and $\frac{k}{k-1}$ for $k = 2$ and $k \geq 5$. We study only the pairs (k, β) generating infinitely many minimal powers, because the other case is trivial. For any pair (k, β), our goal is to determine all periods of minimal powers or at least to find some "forbidden" periods, if any.

The general problem can be subdivided into two big cases: $\beta \geq 2$ ("true" powers) and $\beta < 2$ ("sesquipowers"). We study both of them. In Sect. 2 we give the full solution to the case of true powers. The binary alphabet provides the most nontrivial part of the problem in this case. As a corollary of our results, we complete the classification of binary power-free circular words w.r.t. length, strengthening the result of [2]. In Sect. 3 we study sesquipowers. We give a sufficient condition on β w.r.t. k to generate minimal powers of all positive periods and describe some sets of forbidden periods for small β. Finally, in Sect. 4 we discuss some numerical results obtained by computer and give a conjecture which classifies all sesquipowers on the base of obtained theoretic and experimental results.

1 Preliminaries

We recall some notation and definitions on words, see [11] for more background.

An *alphabet* Σ is a nonempty finite set, the elements of which are called *letters*. *Words* are finite sequences of letters. As usual, we write Σ^* for the set of all words over Σ, including the *empty word* λ. A word u is a *factor* (resp., *prefix*, *suffix*) of a word w if w can be represented as $\bar{v}u\hat{v}$ (resp., $u\hat{v}$, $\bar{v}u$) for some (possibly empty) words $\bar{v}$ and $\hat{v}$. A *factor* (*prefix*, *suffix*) of w is called *proper* if it does not coincide with w. Words u and w are *conjugates* if $u = \bar{v}\hat{v}$, $w = \hat{v}\bar{v}$ for some words $\bar{v}$ and $\hat{v}$. Conjugacy is obviously an equivalence relation.

A word $w \in \Sigma^*$ can be viewed as a function $\{1, \ldots, n\} \to \Sigma$. Then a *period* of w is any period of this function. The *exponent* of w is given by $\exp(w) = |w|/\operatorname{per}(w)$, where $\operatorname{per}(w)$ is the minimal period of w, $|w|$ is the length of w. The word w is said to be β-free (β^+-free) if all its factors have exponents less than β (resp., at most β). Following [5], we use only the term β-free, assuming that β belongs to the set of "extended rationals". This set consists of all rational numbers and all such numbers with a plus; the number x^+ covers x in the usual $\leq$ order such that the inequalities $y \leq x$ and $y < x^+$ are equivalent.

If $\exp(w) > 1$, then w is a *fractional power*. The prefix of w of length $\operatorname{per}(w)$ is called the *root* of w. If $(|w|-1)/\operatorname{per}(w) < \beta \leq \exp(w)$ for some extended rational β, we say that w is a *β-power* and write $w = u^\beta$, where u is the root of w. (Thus, $(yo)^{(3/2)^+} = (yo)^{5/3} = (yo)^2 = yoyo$.) The 2-powers are called *squares*. The β-power is *minimal* if it contains no other β-powers as factors. A period p is *forbidden* for the pair (k, β), if no minimal k-ary β-powers of period p exist.

The *Thue-Morse word* $\mathbf{t}$ is the infinite binary word obtained by iteration of the morphism θ defined by $\theta(a) = ab$, $\theta(b) = ba$. The word $\mathbf{t}$ is 2^+-free, and each square in it has the period 2^m or $3{\cdot}2^m$ for an appropriate integer m.

If we replace the linear ordering of letters in a word w with a cyclic ordering such that the last letter precedes the first one, we get a *circular word*, denoted by (w). There is an obvious one-to-one correspondence between circular words and conjugacy classes of ordinary words. The factors of circular words are ordinary words, including w and its conjugates. Thus, the definition of β-freeness naturally extends to circular words.

2 Minimal Powers of the Exponent $\beta \geq 2$

In this section we completely describe the periods of minimal β-powers. The case of binary alphabet is quite nontrivial, so we start with larger alphabets.

Proposition 1. *If $k \geq 4$, then each positive integer is the period of some k-ary minimal square.*

Proof. To obtain a minimal square u^2 over the alphabet $\{a_1, \ldots, a_k\}$ with the given period $p = |u|$, take an arbitrary 2-free word z of length p over $\{a_1, a_2, a_3\}$ and change the last letter of z to a_4 to get u. Indeed, let v^2 be a factor of u^2. Since z is 2-free, the word v^2 contains the letter a_4. Hence, v contains a_4, and v^2 contains both occurrences of a_4 in u^2. The distance between these occurrences is p, whence p is the period of v^2, implying $v = u$. □

Proposition 2. *If $k \geq 3$ and $\beta \geq 2^+$, then each positive integer is the period of some k-ary minimal β-power.*

Proof. Repeat the argument from the proof of Proposition 1, replacing the ternary 2-free word z with a binary β-free word. □

Proposition 3. *A positive integer p is the period of some ternary minimal square if and only if $p \neq 5, 7, 9, 10, 14, 17$.*

Currie [7] proved that ternary 2-free circular words of length p exist for all p except $5, 7, 9, 10, 14$, and 17. Hence, Proposition 3 immediately follows from

Lemma 1. *Let $\beta \geq 2$. If the word u^β is a minimal β-power, then the circular word (u) is β-free. If $\beta = 2$, then the converse is also true.*

Proof. The first statement is trivial, because u^β contains all conjugates of u as factors. To prove the converse, assume that the circular word (u) is 2-free but u^2 is not a minimal square. Hence, u^2 contains a proper factor v^2. If $|v^2| \leq |u|$, then v^2 is a factor of a conjugate of u, contradicting to the 2-freeness of (u). Now examine the case $|v^2| > |u|$. Then $uu = xvvy$ and w.l.o.g. $|x| \geq |y|$. So, we have $v = v_1v_2$, $u = xv_1 = v_2v_1v_2y$. Moreover, the prefix $v_2v_1v_2$ and the suffix v_1 overlap in u. The occurrences of v_1 cannot overlap, because u is square-free. Thus, the suffix v_1 intersects only with the second occurrence of v_2 in the considered prefix. Hence, some word z is a prefix of v_1 and simultaneously a suffix of v_2. Then the prefix v_2v_1 of u contains z^2, which is impossible. □

Remark 1. In general, the converse to the first statement of Lemma 1 is not true, compare Theorem 2 and Lemma 4 below.

For the rest of this section we fix the binary alphabet $\{a, b\}$. The squares cannot be avoided over $\{a, b\}$, while for $\beta > 2$ the description of the minimal β-powers is given by the following theorems.

Theorem 1. *Let $\beta \geq (5/2)^+$. A binary minimal β-power with the period p exists for any positive integer p.*

Theorem 2. *Let $\beta \in [(7/3)^+, 5/2]$. A binary minimal β-power with the period p exists for any positive integer p except $5, 9, 11, 17$, and 18.*

Theorem 3. *Let $\beta \in [2^+, 7/3]$. A binary β-power is minimal if and only if its root is a conjugate of one of the words $\theta^m(a)$, $\theta^m(b)$, $\theta^m(aba)$, $\theta^m(bab)$ for some $m \geq 0$. In particular, the period of any minimal β-power equals 2^m or $3 \cdot 2^m$.*

The "only if" part of the last theorem follows from [9, Theorem 2.2] (see also [4, Lemma 6]). Further, the β-powers of the words a, b, aba, and bab are obviously minimal. Applying the following two lemmas, we are done with the "if" part.

Lemma 2 ([14]). *For any word $u \in \{a,b\}^*$ (a) if $\exp(u) > 1$ then $\exp(u) = \exp(\theta(u))$; (b) for any $\beta > 2$, u is β-free if and only if $\theta(u)$ is β-free.*

Lemma 3 ([15]). *Let $\beta \geq 2$ and w be the root of a minimal β-power. Then any conjugate of w is also the root of a minimal β-power.*

Proof (of Theorem 1). The result by Aberkane and Currie [1] says that the Thue-Morse word $\mathbf{t}$ contains a factor u of any given length n such that the circular word (u) is $\frac{5}{2}^+$-free. This result remains true if we require u to be primitive (i. e., not an integral power of another word). Indeed, $\mathbf{t}$ avoids cubes and contains squares of lengths 2^m and $3{\cdot}2^m$ only. By Theorem 3, $(\theta^m(a))$ is a $\frac{5}{2}^+$-free circular word of length 2^m, while $(\theta^m(aba))$ is a $\frac{5}{2}^+$-free circular word of length $3 \cdot 2^m$.

Take a primitive factor u of length n of $\mathbf{t}$ such that (u) is $\frac{5}{2}^+$-free, and let $\beta \geq \frac{5}{2}^+$. Let us prove that the word u^β is a minimal β-power. Arguing by contradiction, let u^β have a proper factor v^β. Then $|v^\beta| > n$, since all factors of length n in u^β are conjugates of u, and hence are β-free. We also note that v^β is a prefix of the word $\bar{u}^\beta$ for some conjugate $\bar{u}$ of u. Consider the mutual location of parts of the words v^β and $\bar{u}^\beta$.

Note that $|v| \neq n/2$, because $\bar{u}$ is primitive as u is. First consider the case $|v| > n/2$. The word v^β has the periods $|v|$ and n, but no period $\gcd(n, |v|)$, since $\bar{u}$ is primitive. By Fine and Wilf's theorem (cf. [11]), we have $|v^\beta| < n + |v| - \gcd(n, |v|)$, whence $|v^\beta| - n < |v|$. Thus, for some nonempty words x, y, and z we have $v = xyz$, $\bar{u} = xyzxy$, $v^\beta = xyzxyzx$, and in addition $|x| > |yz|$. Both words xyz and zx are prefixes of $\bar{u}$, and these two occurrences of x overlap in $\bar{u}$, since $|x| > |z|$. Using another basic property of words [11], we get $zx = (qr)^n q$ for some $n \geq 2$ and the words $q \neq \lambda$ and r. Moreover, rq is a prefix of yz. Consider the word $zxyz$. Its length is less than $n = |xyzxy|$, because $|z| < |x|$, so it should be $\frac{5}{2}^+$-free as a factor of a conjugate of u. But $zxyz$ has the prefix $(qr)^3 q$, a contradiction.

Now let $|v| < n/2$. First suppose that $\beta \geq 3$. Then for some nonempty words x, y we have $v = xy$, $\bar{u} = xyxyx$, $(xy)^3$ is a prefix of v^β, and $|x| \leq |y|$. Both words y and xy are prefixes of $\bar{u}$. Hence, x is a prefix of y. We see that $\bar{u}$ has the prefix xx and the suffix x. Thus, $\bar{u}^\beta$ has the factor x^3 whose length is less than n, a contradiction with the choice of u. Therefore, $\beta < 3$. Then there exist nonempty words x, y, z such that $v = xyz$, $\bar{u} = xyzxyzx$, $v^\beta = xyzxyzxy$, and $|x| \le |yz|$. As in the previous case, both words y and xy are prefixes of $\bar{u}$, and if x is a prefix of y, then the existence of the factor x^3 leads to a contradiction. Hence, $x = ys$ for a nonempty word s. Consider the word

$$\bar{u}^2 = ysyzysyz\boldsymbol{ys}\,\boldsymbol{ysy}zysyzys\,.$$

The 2^+-free word u is among the factors of length n of $\bar{u}^2$. Since in the middle of $\bar{u}^2$ we see the factor $ysysy$, whose length is less than n, u is the factor either of the word $ysyzysyzysys$, or of the word $sysyzysyzys$. But in both cases any factor of length n contains a 2^+-power with the period $|ysyz| = |v|$. This contradiction completes the proof. □

Now turn to the proof of Theorem 2. It is based on two key lemmas.

Lemma 4. *There exist no binary* $(5/2)$*-free circular words of length* $5, 9, 11$*, and* 18*. The only such words of length* 17 *are* $(aabbabbaabbabbaab)$ *and its automorphic image* $(bbaabaabbaabaabba)$*.*

Proof. Try to construct a (5/2)-free circular word (u) of the required length. W.l.o.g., we assume that at least a half of the letters in u are a's. Each binary circular word can be encoded by the sequence of distances between consequtive b's (this sequence is a circular word also). If such a distance is greater than 3, then the original word contains a^3. Hence the codeword of a (5/2)-free circular word contains only 1's, 2's, and 3's. Moreover, such a codeword satisfies the following restrictions:

(i) no factor 11 (this factor encodes bbb);
(ii) no factor 22 (this factor encodes $babab$);
(iii) any 2 is preceded or followed by 1 (to avoid $ababa$ in the original word);
(iv) any 33 (encodes $baabaab$) is preceded **and** followed by 1;
(v) no factor 2121 (this factor encodes $babbabb$, which cannot be extended to the right); symmetrically, no factor 1212;
(vi) similar to (v), no factors 3131 (this factor encodes $baabbaabb$) and 1313.

Considering the codewords of the circular words of required lengths, one can check that these words are not (5/2)-free except for $(bbaabaabbaabaabba)$. Due to space constraints, we examine here only one case. We also note that this lemma can be proved by a brute-force computer search.

The set of distances 33332111 (length 17): since the codeword contains 21 or 12 (iii), it also contains 33, and then 1331 (iv). This factor cannot be extended by 1 (i). Since there is only one digit 2, 1331 should be extended by 3 (w.l.o.g., to the factor 13313). By (vi), the factor 13313 is followed either by 2 or by 3. If it is followed by 2, the next digit is 1 (iii), and we get the circular word (13313213), violating (vi). Hence the codeword has the factor 133133. By (iv), we get a unique codeword (13313312). Note that this circular word coincides with its reversal. The coded word is $(u) = (bbaabaabbabbaabaa)$, mentioned in the conditions of the lemma. It can be directly verified that (u) is $\frac{5}{2}$-free. □

Lemma 5. *For any $\beta \in [(7/3)^+, 5/2]$ and any positive even number $n \neq 6$ there exists a word $u \in \{a,b\}^*$ of length n such that the word $v = abbaabaabba\,u$ is the root of a minimal β-power.*

Proof. Let v^β have a proper factor of the form w^β. Note that $aabaa$ is a factor of v. If $aabaa$ occurs only once in the word va, then $aabaa$ cannot occur in w^β. Indeed, if w^β has only one occurrence of $aabaa$, then the two factors aa of $aabaa$ belong to different w's, implying $|w| = 3$. But the longest 3-periodic factor of v^β containing $aabaa$ has the exponent 7/3 only. Further, if w^β has two occurrences of $aabaa$, then $|w| = |v|$, which is impossible. Thus, if the word u satisfies the conditions (a) *the word va has a unique occurrence of $aabaa$* and (b) *the word $abaabba\,u\,abbaaba$ is β-free*, then the β-power v^β is minimal.

We set $z = baabba\,u\,abbaab$ (then the word mentioned in (b) is aza) and $u = \theta(\bar{u})$. Then $z = \theta(\bar{z})$, where $\bar{z} = bab\,\bar{u}\,aba$. Note that if z is 2^+-free, then both (a) and (b) are satisfied. Taking $u = ba$ (resp., $u = baab$), we see that z is a factor of $\theta^2(bbaa)$ (resp., $\theta^2(aabaa)$), and hence a 2^+-free word by Lemma 2. So, for the rest of the proof we assume $|u| \geq 8$, whence $|\bar{u}| \geq 4$. To build the words u and z, we consider four cases.

1. $|\bar{u}| = 4n$. We put $\bar{u} = \theta(\bar{\bar{u}})$, where the word $\bar{\bar{u}}$ has even length. Then $a\bar{z}b = \theta(aa\bar{\bar{u}}aa)$. So, by Lemma 2 it is enough to find the word $\bar{\bar{u}}$ such that the word $\bar{\bar{z}} = aa\bar{\bar{u}}aa$ is 2^+-free. If we choose $\bar{\bar{u}}$ in such a way that $\bar{\bar{z}}$ is a factor of $\mathbf{t}$, we are done. Note that for any $n \geq 0$ there exists a word x such that the word $baabxb$ (and hence the word bxb) is a factor of $\mathbf{t}$. To see this, take m such that $2^m > n$ and $\theta^m(a)$ ends with $baab$. Both words $\theta^m(aa)$, $\theta^m(ab)$ are factors of $\mathbf{t}$, and one of the words $\theta^m(a)$, $\theta^m(b)$ contains b in the $(n{+}1)$th position.

If $|\bar{\bar{u}}| = 4n'{+}2$, consider the factor of $\mathbf{t}$ of the form $\theta^2(bxb) = baab\,\theta^2(x)\,baab$, where $|x| = n'$. Deleting the first and the last letters, we get the required word $\bar{\bar{z}}$. If $|\bar{\bar{u}}| = 4n'{+}8$, take the factor of $\mathbf{t}$ of the form $\theta^2(baabxb) = baababbaabbabaab\,\theta^2(x)\,baab$, where $|x| = n'$, and delete the last letter and the first seven letters to get $\bar{\bar{z}}$. Finally, if $|\bar{\bar{u}}| = 4$, take $\bar{\bar{z}} = aabbabaa$ (it is a factor of $\theta^4(b)$). Thus, we have built the word $\bar{\bar{z}}$ in all cases.

2. $|\bar{u}| = 4n{+}2$. Set $\bar{u} = a\theta(\bar{\bar{u}})b$, where the word $\bar{\bar{u}}$ has even length. Hence $\bar{z} = \theta(bb\bar{\bar{u}}bb)$, and the argument symmetric to that of case 1 holds.

3. $|\bar{u}| = 4n{+}1$. Set $\bar{u} = \theta(\bar{\bar{u}})b$. Similar to the above, the length of $\bar{\bar{u}}$ is even, we have $a\bar{z} = \theta(aa\bar{\bar{u}}bb)$, and it is enough to choose $\bar{\bar{u}}$ such that the word $\bar{\bar{z}} = aa\bar{\bar{u}}bb$ is 2^+-free. As in case 1, one can show that for any $n \geq 0$ there exists a word x such that the word $baabxa$ (and hence the word bxa) is a factor of $\mathbf{t}$. Further, if $|\bar{\bar{u}}| = 4n'{+}2$ (resp., $|\bar{\bar{u}}| = 4n'{+}8$) we let $|x| = n'$ and take an appropriate factor of the word $\theta^2(bxa)$ (resp., $\theta^2(baabxa)$) as $\bar{\bar{z}}$. Finally, if $|\bar{\bar{u}}| = 4$, we take $\bar{\bar{z}} = aabbaabb$.

4. $|\bar{u}| = 4n{+}3$. Set $\bar{u} = baabab\,\theta(\bar{\bar{u}})b$. The length of the word $\bar{\bar{u}}$ is even (possibly zero). We also have $a\bar{z} = \theta(\bar{\bar{z}})$, where $\bar{\bar{z}} = aabaa\bar{\bar{u}}bb$. As in case 3, we choose the word $\bar{\bar{u}}$ such that $z' = aa\bar{\bar{u}}bb$ is a 2^+-free word ($\bar{\bar{u}} = \lambda$ also satisfies this condition). Now we consider all possible 2^+-powers in the word $\bar{\bar{z}} = aabz'$ and verify that the word aza is always $(7/3)^+$-free.

Adding b before the word $z' = aabbaabb$, we prolong the period 4 to get a 2^+-power, but this period gets broken when we add aa. So, $\bar{\bar{z}}$ ends with $(baab)^{9/4}$

and contains no other 2^+-powers. Thus, z ends with $(\theta^4(b))^{9/4}$ and aza ends with $(\theta^4(b))^{37/16}$. So, aza is $(7/3)^+$-free.

If $z' \neq aabbaabb$, then z' is a factor of $\mathbf{t}$, and so is bz'. Since bz' is not a square (because of the odd length), the word abz' is not a 2^+-power. If abz' begins with a 2^+-power axx, then $|x| \geq 6$. (Since x is a factor of $\mathbf{t}$, its length equals 2^m or $3{\cdot}2^m$ for some m. The word bz' begins with $baabab$ or $baabba$.) Since $aabaa$ is not a factor of $\mathbf{t}$, the word $aaxx$ has no period $|x|$. Hence, axx cannot induce a $(7/3)^+$-power in the word aza. It remains to consider the case when a 2^+-power is a prefix of $\bar{\bar{z}} = aabz'$. But since $aabaa$ is not a factor of $\mathbf{t}$, this 2^+-power is exactly $aabaaba$. Then the word z begins with a $(13/6)$-power $ba(\theta^2(aba))^2$, and aza is obviously $(7/3)^+$-free. The case examination is finished. □

Proof (of Theorem 2). By Lemmas 1, 4, the minimal powers of lengths $5, 9, 11$, and 18 do not exist. Consider the $\frac{5}{2}$-free circular word $(u) = (bbaabaabbaabaabba)$ mentioned in Lemma 4. None of the conjugates of u is the root of a minimal β-power. Indeed, one of these conjugates is the word $w = abbaaba\,abbaaba\,abb$, having the period 7. The word wa has the same period and is a $(5/2)^+$-power. Since a is the first letter of w, the word wa is a factor of the word $\bar{u}^\beta$ for any conjugate $\bar{u}$ of u. Thus, the minimal powers of length 17 do not exist also.

By Lemma 2, if a word u is the root of a minimal β-power, then so is the word $\theta(u)$. Hence the existence of the minimal binary β-power of period n implies the existence of such a power of period $2n$. Lemma 5 gives us minimal β-powers of any odd period $n \geq 13$, except for $n = 17$. Thus, to prove the theorem it suffices to find minimal β-powers with the periods $1, 3, 7, 10, 22, 34$, and 36. The words

$$a, aba, aabaabb, aabaabbabb, aabaabbaababbabaabbabb,$$
$$aabaabbaababbaabbabaababbabaabbabb, aabaabbaababbaabaabbabaabbaabaabbabb$$

are roots of minimal β-powers, as one can check directly. The theorem is proved.

As a corollary, we strengthen the result of [2] which states the existence of binary $(7/3)^+$-free circular words of any length $n \geq 210$. Thus, we complete the classification of possible lengths of binary β-free circular words (all other cases are covered by [1,2]).

Corollary 1. *Let $\beta \in [(7/3)^+, 5/2]$. A binary β-free circular word of length p exists if and only if $p \notin \{5, 9, 11, 18\}$.*

3 Minimal Powers of the Exponent $\beta < 2$

This section contains several results towards the dichotomy of all triples (k, β, p) w.r.t. the existence of a minimal k-ary β-power of period p. Note that any β-power has the form xyx and the period $|xy|$. By Proposition 1, for $k \geq 4$ there exist minimal squares of any period. The following theorem shows that for $k \geq 6$ the square can be replaced by a smaller exponent depending on k.

Theorem 4. *If $\beta > RT(\lfloor k/2 \rfloor)$, then a minimal k-ary β-power with the period p exists for any positive integer p.*

Proof. We partition the alphabet into two parts, containing $\lfloor k/2 \rfloor$ and $\lceil k/2 \rceil$ letters respectively, and build the word xy of the length p such that $xyx = (xy)^\beta$ is a minimal β-power. From p and β we calculate the length p_1 of x and the length p_2 of y. If $p_2 = 0$ (this can happen if p is small and β is close to 2), then $z^\beta = z^2$ for all z such that $|z| \le p_1$. Hence, the minimal β-power of length p is a minimal square. Such a square exists by Proposition 1. For the rest of the proof we assume that $p_2 > 0$. Take an arbitrary β-free word of the length p_1 (resp., p_2) over the first (resp., second) part of the alphabet as x (resp., y). Such words exist by the restriction on β. Now let us prove that the β-power xyx is minimal.

Assume that the word xyx has a proper factor $ztz = (zt)^\beta$. Since the letters in x and y are distinct, the left (resp., right) z is a factor of the left (resp., right) x. Since $|zt| < |xy|$, the word x contains at least two occurrences of z. These occurrences neither overlap nor touch, because x is β-free and $\beta < 2$. Hence we can write $x = x_1zx_2zx_3$, where $x_2 \ne \lambda$. From this equality we get $|z| \le (|x|-1)/2$. Using the inequality

$$\frac{|z| + |y|}{|z|} = 1 + \frac{|y|}{|z|} \ge 1 + \frac{2|y|}{|x|-1} \ge 1 + \frac{|y|+1}{|x|-1} = \frac{|x|+|y|}{|x|-1},$$

we estimate the exponent of the word ztz:

$$\beta \le \exp(ztz) = \frac{|ztz|}{|zt|} = 1 + \frac{|z|}{|zx_3yx_1|} \le 1 + \frac{|z|}{|z| + |y|} \le 1 + \frac{|x|-1}{|x|+|y|} = \frac{|xyx|-1}{|xy|}.$$

The last fraction is exactly the exponent of the maximal proper prefix of xyx. By the definition of β-power, this exponent is less than β, a contradiction. Thus, the β-power xyx is minimal by definition. □

Now consider the values of β which are close to the repetition threshold. We assume $k \ge 5$, so that $RT(k) = \frac{k}{k-1}$. We use the term *m-repetition* for the minimal β-powers xyx such that $|x| = m$. First, let $\beta \in \left[\frac{k}{k-1}^+, \frac{k-1}{k-2}\right]$. Then any successive

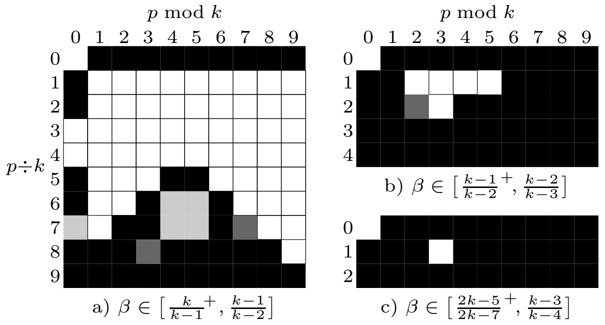


Fig. 1. The existence of short minimal k-ary β-powers ($k = 10$). Black (resp., white, grey) squares mark the periods which are allowed (resp., forbidden by Theorems 5–7; forbidden and found by computer). Light (resp., dark) grey color marks the periods forbidden for all (resp., for some) exponents in the given range.

$k{-}1$ letters in a β-free word are different (in particular, if $m < k$ then all letters in x are different). The distribution of periods of minimal β-powers is illustrated by Fig. 1, a. Some short forbidden periods are described by the following theorem.

Theorem 5. *Let $\beta \in \left[\frac{k}{k-1}^{+}, \frac{k-1}{k-2}\right]$. If an integer p satisfies one of the conditions*
(a) $k < p < \left\lceil \frac{k+1}{2} \right\rceil (k{-}1)$ and $p \bmod k \neq 0$,
(b) $p \in [(m{-}2)(k{+}1){+}1, m(k{-}1){-}1]$ for an integer $m \geq \left\lceil \frac{k+3}{2} \right\rceil$ and $p \bmod k \neq 0$,
(c) $p = 3k$ or $p = 4k$,
then no minimal k-ary β-powers of period p exist.

The following simple observation is very useful.

Remark 2. The distance between two closest occurrences of a letter in a β-free word is $k{-}1$, k, or $k{+}1$.

Let xyx be an m-repetition. We refer to the letters occurring in x as to *X-letters*. If all X-letters have the same number of occurrences in xy then we say that the repetition is *uniform*. We begin with the case $\beta = \frac{k}{k-1}^{+}$. The following lemma collects some results of [18].

Lemma 6 ([18]). *(a) The set of all possible periods of 1- and 2-repetitions coincides with the interval $[1, k]$;*
(b) any m-repetition such that $3 \leq m \leq \left\lceil \frac{k+1}{2} \right\rceil$ is uniform;
(c) the period of any uniform m-repetition satisfying $3{\leq}m{<}k$ is divisible by k;
(d) uniform 4- and 5-repetitions do not exist.

Since the condition $m \leq \left\lceil \frac{k+1}{2} \right\rceil$ is equivalent to $p < \left\lceil \frac{k+1}{2} \right\rceil (k{-}1)$, the statements (a)–(c) of Lemma 6 imply the statement (a) of Theorem 5 for the case $\beta = \frac{k}{k-1}^{+}$. The statement (c) for this case is implied by Lemma 6(d), while the statement (b) readily follows from the next lemma concerning non-uniform repetitions.

Lemma 7. *If $m \leq k$, then the period of a non-uniform m-repetition belongs to the interval $[(m{-}1)(k{-}1), (m{-}2)(k{+}1)]$.*

Proof. First suppose that all X-letters are different. Note that the period of any m-repetition xyx ($|x| = m$) belongs to the interval $[(m{-}1)(k{-}1), m(k{-}1) - 1]$. Indeed, the exponent of a longer word xyx with $|x| = m$ is too small, while a shorter word has a proper prefix which is a β-power. The period coincides with the distance between the two marginal occurrences of any X-letter a. By Remark 2, the number of a's in the word xyx is at most m (for $(m{+}1)$ a's, the distance between the two marginal ones is at least $m(k{-}1)$). On the other hand, the number of a's is at least $m{-}1$ (for $m{-}2$ a's, the distance between the two marginal ones is at most $(m{-}3)(k{+}1) < (m{-}1)(k{-}1)$). Since the considered repetition is non-uniform, some X-letter a occurs m times in xyx, while some other X-letter b occurs only $m{-}1$ times. By Remark 2, the distance between the two marginal a's (resp., b's) is at least $(m{-}1)(k{-}1)$ (resp., at most $(m{-}2)(k{+}1)$), whence the result.

Now let some X-letters coincide. Since $m \leq k$, we have $m = k$ and $x = a_1 a_2 \ldots a_{k-1} a_1$, where the letters $a_1, \ldots, a_{k-1}$ are different. Suppose that $p = |xy| > (k{-}2)(k{+}1)$. Then the letter a_1 (resp., each of the letters $a_2, \ldots, a_{k-1}$)

occurs at least $k{+}1$ (resp., k) times in xyx. Consider the letter a_k: by Remark 2, it occupies the first and the last positions of the word y. The distance between these positions is $|xy| - |x| - 1 = p - (k{+}1) > (k{-}3)(k{+}1)$. Hence, by Remark 2 a_k occurs in y at least $k{-}1$ times. We get $|xyx| \geq k{+}1 + (k{-}2)k + k{-}1 = k^2$, yielding $\exp(xyx) = |xyx|/(|xyx|{-}k) \leq k/(k{-}1)$, which is impossible, because xyx is a k-repetition. The lemma is proved. □

The proofs of both Lemma 6 (in [18]) and Lemma 7 refer to Remark 2 only. But if a $\frac{k}{k-1}^{+}$-power xyx is not minimal because of Remark 2, then any bigger β-power $(xy)^{\beta}$ is not minimal by the same reason. Hence, all statements of Theorem 5 hold for all $\beta \in \left[\frac{k}{k-1}^{+}, \frac{k-1}{k-2}\right]$. By a mere computation, we have

Corollary 2. *If $\beta \in \left[\frac{k}{k-1}^{+}, \frac{k-1}{k-2}\right]$, then minimal k-ary β-powers of period p do not exist for at least $\frac{3k^2-10k+16-(k \bmod 2)}{4}$ different values of p.*

Now choose β from the interval $\left[\frac{k-1}{k-2}^{+}, \frac{k-2}{k-3}\right]$. It is not so close to repetition threshold but still admits forbidden periods. Note that any successive $k{-}2$ letters in a β-free word are different.

Theorem 6. *Let $\beta \in \left[\frac{k-1}{k-2}^{+}, \frac{k-2}{k-3}\right]$. If $p \in [(m{-}1)(k{+}1){+}1, m(k{-}2){-}1]$ for some integers $m \in [2, k{-}2]$ and p, then no minimal k-ary β-powers of period p exist.*

Proof. First, let $\beta = \frac{k-1}{k-2}^{+}$. Take an m-repetition xyx. Note that $|xy| < m(k{-}2)$, because for $|xy| = m(k{-}2)$ we have $\exp(xyx) = \frac{k-1}{k-2}$. Since $m \leq k{-}2$, all letters in x are different. Moreover, each of the X-letters occurs in y at most $m{-}2$ times. Indeed, the distance between the closest occurrences of a letter is at least $k{-}2$, while the distance between the marginal occurrences of an X-letter is $|xy|$, which is less than $m(k{-}2)$.

Any factor of y of length k contains at least $k{-}1$ different letters (and then, at least $m{-}1$ X-letters). If $p = |xy| \geq (m{-}1)(k{+}1) + 1$, then $|y| \geq (m{-}1)k$. Hence, y contains at least $(m{-}1)^2$ occurrences of X-letters. On the other hand, we have m X-letters, each one occurring at most $m{-}2$ times in y. This contradiction proves that an m-repetition of period p cannot exist.

Now note that if a k-ary $\frac{k-1}{k-2}^{+}$-power of period p certainly contains $k{-}2$ successive letters which are not all distinct, than any bigger k-ary power of the same period also contains such letters. Hence, the periods that are forbidden for the exponent $\frac{k-1}{k-2}^{+}$, are forbidden for any exponent $\beta \in \left[\frac{k-1}{k-2}^{+}, \frac{k-2}{k-3}\right]$. □

Corollary 3. *If $\beta \in \left[\frac{k-1}{k-2}^{+}, \frac{k-2}{k-3}\right]$ and $k \geq 7$, then the minimal k-ary β-powers of period p do not exist for some values of p. The number of such values is at least $\frac{k^2-9k+q}{6}$, where $q = 18$ if k is divisible by 3 and $q = 20$ otherwise.*

We also found one period which remains forbidden for even bigger exponents:

Theorem 7. *If $k \geq 9$ and $\beta \in \left[\frac{2k-5}{2k-7}^{+}, \frac{k-3}{k-4}\right]$, then no minimal k-ary β-powers of period $2k{-}7$ exist.*

Remark 3. It is quite interesting that for smaller exponents (namely, for all $\beta \in \left[\frac{k-2}{k-3}^{+}, \frac{2k-5}{2k-7}\right]$) the k-ary minimal β-powers of the period $2k-7$ exist.

Proof (of Theorem 7). The period $2k-7$ corresponds to a 3-repetition, since $\frac{2k-5}{2k-7} < \beta$. Assume that such a 3-repetition xyx exists. Note that each successive $k-3$ letters in xyx are distinct. Then all X-letters are different and do not occur in y, since otherwise the distance between two occurrences of an X-letter would be less than $k-3$. Hence at most $k-3$ different letters occur in y (and each successive $k-3$ letters are distinct). Thus, $|y| \leq k-2$. We then have a contradiction, because $|xy| \leq k+1 < 2k-7$. □

4 Computer-Assisted Results and the Conjecture

The results of Sect. 3 cover only part of all possible triples (k, β, p) with $\beta < 2$. Using computer, we shed some light on the following questions:
(1) for the exponents that are not mentioned in Theorems 5–7, are there any forbidden periods?
(2) for the exponents studied in Theorems 5–7, what forbidden periods were not mentioned there?

We studied β-powers over the alphabets from 3 to 15 letters and get the following. For $\beta \geq \frac{k-2}{k-3}^{+}$, no forbidden period except the one mentioned in Theorem 7, was found. The same is true for $\beta = \frac{k-1}{k-2}^{+}$, but some additional forbidden periods appear with the growth of β within the interval $\left[\frac{k-1}{k-2}^{+}, \frac{k-2}{k-3}\right]$, e. g., the period 7 for $k = 6$, $\beta > \frac{9}{7}$, and the period 22 for $k = 10$, $\beta > \frac{25}{22}$, see Fig. 1, b. These forbidden periods reflect the situation when the period of an m-repetition becomes the period of an $(m+1)$-repetition with increasing β.

In the "lowest" interval $\left[\frac{k}{k-1}^{+}, \frac{k-1}{k-2}\right]$ we discovered some forbidden periods for m-repetitions with $m \leq k$ other than the periods mentioned in Theorem 5. For example, there are seven "additional" forbidden periods for $k = 10$, see Fig. 1, a. The "square" 64, 65, 74, 75 was found for other even-size alphabets also, while the periods 70, 77, 83 are sporadic. Concerning the case $m > k$, we present a table with some statistics on the forbidden periods for $\beta = (RT(k))^{+}$:

k	β	range	$\#P(k,\beta)$	$\max p$
3	$(7/4)^{+}$	4 – 66	8	22
4	$(7/5)^{+}$	10 – 205	78	165
5	$(5/4)^{+}$	20 – 113	28	64
6	$(6/5)^{+}$	30 – 93	14	52
7	$(7/6)^{+}$	42 – 97	9	59
8	$(8/7)^{+}$	56 – 104	4	61
9	$(9/8)^{+}$	72 – 103	0	
10	$(10/9)^{+}$	90 – 104	0	

Third to fifth column contain the range of examined periods, the number of forbidden periods found, and the maximum of these periods, respectively.

So, these computer-assisted results suggest that Theorems 5–7 list all the exponents for which forbidden periods exist, and describe the most part of these forbidden periods for large enough alphabets. Hence we finish with a conjecture which summarizes our knowledge of minimal β-powers obtained from the theoretic results of Sect. 3 and computer experiments.

Conjecture 1. Let $k \geq 3$.

1) A forbidden period for the pair (k, β) exists if and only if one of the following conditions is satisfied: (a) $\beta \leq \frac{k-1}{k-2}$; (b) $k = 6$ and $\beta \in [\frac{9}{7}^+, \frac{4}{3}]$, or $k \geq 7$ and $\beta \in \left[\frac{k-1}{k-2}^+, \frac{k-2}{k-3}\right]$; (c) $k \geq 9$ and $\beta \in \left[\frac{2k-5}{2k-7}^+, \frac{k-3}{k-4}\right]$.
2) For any pair (k, β) the set of all forbidden periods is finite.
3) If $k \geq 9$, then a forbidden period for any pair (k, β) cannot exceed $k(k-1)-1$.

Note that the period $k(k-1)-1$ is forbidden for any $\beta \leq \frac{k-1}{k-2}$ by Theorem 5.

References

1. Aberkane, A., Currie, J.D.: The Thue-Morse word contains circular $(5/2)^+$-power-free words of every length. Theor. Comput. Sci. 332, 573–581 (2005)
2. Aberkane, A., Currie, J.D.: Attainable lengths for circular binary words avoiding k-powers. Bull. Belg. Math. Soc. Simon Stevin 12(4), 525–534 (2005)
3. Allouche, J.-P., Shallit, J.: Automatic Sequences: Theory, Applications, Generalizations, 588 p. Cambridge Univ. Press, Cambridge (2003)
4. Blondel, V.D., Cassaigne, J., Jungers, R.: On the number of α-power-free binary words for $2 < \alpha \leq 7/3$. Theor. Comput. Sci. 410, 2823–2833 (2009)
5. Brandenburg, F.-J.: Uniformly growing k-th power free homomorphisms. Theor. Comput. Sci. 23, 69–82 (1983)
6. Carpi, A.: On Dejean's conjecture over large alphabets. Theor. Comput. Sci. 385, 137–151 (2007)
7. Currie, J.D.: There are ternary circular square-free words of length n for $n \geq 18$. Electron. J. Comb. 9(1), 10 (2002)
8. Currie, J.D., Rampersad, N.: A proof of Dejean's conjecture, `http://arxiv.org/PScache/arxiv/pdf/0905/0905.1129v3.pdf`
9. Currie, J.D., Rampersad, N.: Infinite words containing squares at every position. Theor. Inform. Appl. 44, 113–124 (2010)
10. Dejean, F.: Sur un Theoreme de Thue. J. Comb. Theory, Ser. A 13(1), 90–99 (1972)
11. Lothaire, M.: Combinatorics on words. Addison-Wesley, Reading (1983)
12. Noonan, J., Zeilberger, D.: The Goulden-Jackson Cluster Method: Extensions, Applications, and Implementations. J. Difference Eq. Appl. 5, 355–377 (1999)
13. Rao, M.: Last Cases of Dejean's Conjecture. In: Proceedings of the 7th International Conference on Words, Salerno, Italy, p. 115 (2009)
14. Shur, A.M.: The structure of the set of cube-free Z-words over a two-letter alphabet. Izv. Math. 64(4), 847–871 (2000)
15. Shur, A.M.: Growth rates of complexity of power-free languages. Theor. Comp. Sci. (2008) doi:10.1016/j.tcs.2010.05.017
16. Shur, A.M.: Growth rates of power-free languages. Russian Math. (Iz VUZ) 53(9), 73–78 (2009)
17. Shur, A.M.: Two-sided bounds for the growth rates of power-free languages. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 466–477. Springer, Heidelberg (2009)
18. Shur, A.M., Gorbunova, I.A.: On the growth rates of complexity of threshold languages. RAIRO Theor. Inf. Appl. 44, 175–192 (2010)

The Averaging Trick and the Černý Conjecture

Benjamin Steinberg[⋆]

School of Mathematics and Statistics
Carleton University
1125 Colonel By Drive
Ottawa, ON, Canada
bsteinbg@math.carleton.ca

Abstract. The results of several papers concerning the Černý conjecture are deduced as consequences of a simple idea that I call the averaging trick. This idea is implicitly used in the literature, but no attempt was made to formalize the proof scheme axiomatically. Instead, authors axiomatized classes of automata to which it applies.

1 Introduction

Recall that a (complete deterministic) automaton $\mathscr{A} = (Q, \Sigma)$ with state set Q and alphabet Σ is called *synchronizing* if there is a word $w \in \Sigma^*$ such that $|Qw| = 1$. The word w is called a *synchronizing word*. The main conjecture in this area is:

Conjecture 1 (Černý [1]). *An n-state synchronizing automaton admits a synchronizing word of length at most $(n-1)^2$.*

There is a vast literature on this subject. See for example [2,3,4,5,1,6,7,8,9,10, 11,12,13,14,15,16,17,18,19,20,21,22,23,24,25]. The best known upper bound is cubic [26], whereas it is known that one cannot do better than $(n-1)^2$ [1].

My goal here in this note is not to prove the Černý conjecture for a new class of automata, but rather to give a no-frills, uniform approach to an argument that underlies a growing number of results in the Černý conjecture literature (cf. [7,13,24,20,21,22]). Underlying all these results (as well as the more difficult results of [5] and [17]) are two simple ideas:

- if a finite sequence of numbers is not constant, then it must at some place exceed its average;
- finite dimensional vector spaces satisfy the ascending chain condition on subspaces.

The latter idea is often cloaked in the language of rational power series.

The paper is organized as follows. In the next section I state what I call the "Averaging Lemma." It is a method, with a probabilistic flavor, for obtaining

⋆ The author gratefully acknowledges the support of NSERC.

bounds on lengths of synchronizing words. Before proving the lemma, I show how to deduce from it Kari's solution of the Černý conjecture for Eulerian automata, as well as recent results of Béal and Perrin [20] for one-cluster automata and Carpi and d'Alessandro [21,22] for (locally) strongly transitive automata. We also recover an old result of Rystsov [7] on regular automata (which is essentially the same thing as strongly transitive automata). In fact, we obtain new generalizations of all these results. The final section proves the Averaging Lemma.

2 The Averaging Trick

Let Σ be an alphabet. Denote by Σ^* the free monoid on Σ and put

$$\Sigma^{\leq d} = \bigcup_{m=0}^{d} \Sigma^m.$$

The ring of polynomials with real coefficients in the non-commuting variables Σ is denoted $\mathbb{R}\Sigma$. By a (finitely supported) *probability* on Σ^*, we mean an element

$$P = \sum_{w \in \Sigma^*} P(w)w \in \mathbb{R}\Sigma$$

such that: $P(w) \geq 0$ for all $w \in \Sigma^*$, and

$$\sum_{w \in \Sigma^*} P(w) = 1.$$

The *support* of P is

$$\sigma(P) = \{w \in \Sigma^* \mid P(w) > 0\}.$$

Notice that if P_1 and P_2 are probabilities, then so is P_1P_2. Also note that $\sigma(P_1P_2) = \sigma(P_1)\sigma(P_2)$.

If $X \colon \Sigma^* \to \mathbb{R}$ is a *random variable*, then the *expected value* of X (with respect to the probability P) is:

$$\mathbf{E}_P(X) = \sum_{w \in \Sigma^*} P(w)X(w) = \sum_{w \in \sigma(P)} P(w)X(w). \tag{1}$$

The fundamental property of a random variable that we exploit in this paper is that either it is almost surely constant (and equal to its expectation), or with positive probability it exceeds it expectation. More precisely, it is immediate from (1) and the definition of a probability that either $X(w) = \mathbf{E}_P(X)$ for all $w \in \sigma(P)$, or there is a value $w \in \sigma(P)$ with $X(w) > \mathbf{E}_P(X)$.

Suppose now that $\mathscr{A} = (Q, \Sigma)$ is an automaton with $|Q| = n$. We view elements of $\mathbb{R}Q$ as row vectors. Let $\pi \colon \mathbb{R}\Sigma \to M_n(\mathbb{R})$ be the corresponding matrix representation (cf. [27]); so if

$$f = \sum_{w \in \Sigma^*} f(w)w,$$

and $q, r \in Q$, then

$$\pi(f)_{q,r} = \sum_{\{w \in \Sigma^* \mid qw = r\}} f(w).$$

We shall usually omit π from the notation and view $\mathbb{R}\Sigma$ as acting on row and column vectors. If $S \subseteq Q$, then $[S]$ denotes the characteristic row vector of S; e.g., $[Q]$ is the all ones row vector. We use $[S]^T$ to denote the transpose vector. A key fact is that $w[S]^T = [Sw^{-1}]^T$ for $w \in \Sigma^*$, where as usual $Sw^{-1} = \{q \in Q \mid qw \in S\}$.

Lemma 2 (Averaging Lemma). *Let $\mathscr{A} = (Q, \Sigma)$ be a synchronizing automaton with n states, let P_1 be a probability on Σ^* and let $R \subseteq Q$. Set $c = 2$ if, for each proper non-empty subset $S \subsetneq R$, there exist $w_1, w_2 \in \sigma(P_1)$ with $Sw_1^{-1} \neq Sw_2^{-1}$ and otherwise put $c = 1$. Suppose that there exists a probability P_2 with support $\Sigma^{\leq n-c}$ such that:*

1. *$[R]P_2P_1 = [R]$;*
2. *$R \subseteq q\Sigma^*$ for all $q \in R$;*
3. *there exists $w_0 \in \Sigma^*$ with $Qw_0 \subseteq R$.*

Then $\mathscr{A}$ has a synchronizing word of length at most:

- *$c + (n-2)(n-c+L)$ if $R = Q$;*
- *$(r-1)(n-c+L) + \ell + c - 1$ if $R \subsetneq Q$*

where $r = |R|$, L is the maximum length of a word in $\sigma(P_1)$ and $\ell = |w_0|$.

Remark 3. If r is odd, then the proof shows that the bounds in Lemma 2 can be improved to $1 + (n-2)(n-c+L)$ and $(r-1)(n-c+L) + \ell$, respectively.

Before, proving the lemma, let us use it to derive anew some results from the literature. The first is a result of Kari on synchronizing Eulerian automata [13]. An automaton is *Eulerian* if its underlying graph admits an Eulerian directed path, or equivalently, it is strongly connected and the in-degree of every vertex is the same as the out-degree (and hence is the alphabet size). Actually, we can generalize his result.

Let us say that a strongly connected automaton $\mathscr{A} = (Q, \Sigma)$ is *pseudo-Eulerian* if we can find a probability P with support Σ such that the matrix $\pi(P)$ is doubly stochastic (i.e., each row and column of P adds up to 1). For instance, if $\mathscr{A}$ is Eulerian with adjacency matrix A and $d = |\Sigma|$, then we can set

$$P = \sum_{a \in \Sigma} d^{-1} a.$$

One checks that $\pi(P) = d^{-1}A$, and hence is doubly stochastic by the Eulerian hypothesis. Thus every Eulerian automaton is pseudo-Eulerian. It is easy to

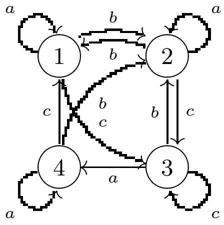


Fig. 1. A pseudo-Eulerian automaton

check whether a strongly connected automaton is pseudo-Eulerian: one just needs to look for a strictly positive solution to the system of $|Q| + 1$ linear equations

$$\begin{aligned} 1 &= \sum_{a \in \Sigma} p_a \\ 1 &= \sum_{a \in \Sigma} p_a \cdot |qa^{-1}| \quad (q \in Q). \end{aligned}$$

The automaton in Figure 1 is pseudo-Eulerian but not Eulerian. Indeed, if we put $P = a/2 + b/6 + c/3$, then

$$\pi(P) = \begin{bmatrix} \frac{1}{2} & \frac{1}{6} & \frac{1}{3} & 0 \\ \frac{1}{6} & \frac{1}{2} & \frac{1}{3} & 0 \\ 0 & \frac{1}{6} & \frac{1}{3} & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{6} & 0 & \frac{1}{2} \end{bmatrix}$$

is doubly stochastic.

Theorem 4. *An n-state synchronizing pseudo-Eulerian automaton has a synchronizing word of length at most $1 + (n-2)(n-1)$.*

Proof. Let $\mathscr{A} = (Q, \Sigma)$ and suppose that P is a probability with support Σ such that $\pi(P)$ is doubly stochastic. Let P_1 be the probability with support concentrated on the empty word and take $R = Q$. As pseudo-Eulerian automata are strongly connected, $Q \subseteq q\Sigma^*$ for all $q \in Q$. Put

$$P_2 = \frac{1}{n} \sum_{m=0}^{n-1} P^m;$$

it is a probability with support $\Sigma^{\leq n-1}$. The condition that $\pi(P)$ is doubly stochastic is equivalent to $[Q]P = [Q]$. Thus

$$[Q]P_2P_1 = [Q] \cdot \frac{1}{n} \sum_{m=0}^{n-1} P^m = [Q].$$

The Averaging Lemma now yields the upper bound of $1 + (n-2)(n-1)$ on the length of a synchronizing word. □

The next result simultaneously generalizes results of Rystsov [7] on regular automata, Béal [24] on circular automata, Béal, Berlinkov and Perrin [20, 28] on one-cluster automata and Carpi and d'Alessandro [21,22] on strongly and locally strongly transitive automata.

Theorem 5. *Let $\mathscr{A} = (Q, \Sigma)$ be a synchronizing automaton. Suppose there is a set of words $W \subseteq \Sigma^*$ and $k \geq 1$ so that, for each state $q \in Q$ and each state $s \in R = QW$, there are exactly k elements of W taking q to s. Let ℓ be the length of the shortest word in W and L be the length of the longest. If $R = Q$, then there is a synchronizing word for $\mathscr{A}$ of length at most $2 + (n-2)(n-2+L)$; if $R \subsetneq Q$, then there is a synchronizing word of length at most $(r-1)(n-2+L) + \ell + 1$ where $r = |R|$.*

Proof. A straightforward counting argument establishes that $|W| = kr$. It remains to define our probabilities in order to apply the Averaging Lemma. Take P_1 to be the uniform distribution on W (so $P_1(w) = 1/|W|$ for $w \in W$ and is otherwise 0). To verify that $c = 2$, let $\emptyset \neq S \subsetneq R$ and suppose that $s \in S$ and $q \in R \setminus S$. Then by the hypothesis on W, there exist $w_1, w_2 \in W$ with $rw_1 = s$ and $qw_2 = q$. Then $q \in Sw_1^{-1}$ but $q \notin Sw_2^{-1}$.

Now let P_2 be an arbitrary probability with support $\Sigma^{\leq n-c}$. The only condition remaining to check in order to apply the Averaging Lemma is that $[R]P_2P_1 = [R]$. First observe that the columns of $\pi(P_1)$ corresponding to elements of $Q \setminus R$ are zero, while if $s \in R$, then the corresponding column of $\pi(P_1)$ is $(k/|W|)[Q]^T = (1/r)[Q]^T$. Since $\pi(P_2)$ is a stochastic matrix (each of its rows sum to 1), this means that $\pi(P_2P_1) = \pi(P_1)$. Next observe that if $s \in R$, then $s\sum_{w \in W} w = k[R]$. Thus

$$[R]\sum_{w \in W} w = \sum_{s \in R} s \sum_{w \in W} w = rk[R] = |W|[R].$$

Therefore, $[R]P_1 = [R]$ and hence $[R]P_2P_1 = [R]$, as required. □

For example, Béal and Perrin [20] call $\mathscr{A} = (Q, \Sigma)$ a *one-cluster automaton* if there exists $a \in \Sigma$ so that a has only one cycle R on Q; see Figure 2. Suppose that the cycle has size r. Then each state of Q is taken to exactly one element of R by the set of words $W = \{a^{n-r}, \ldots, a^{n-1}\}$. Theorem 5 then yields the bound of $2n^2 - 7n + 8$. This should be compared with the bound of $2n^2 - 7n + 7$ from [28], which improves on the earlier bound of $2n^2 - 6n + 5$ from [20]. Indeed, if $r = n$, Theorem 5 immediately yields a bound of $2 + (n-2)(2n-3) = 2n^2 - 7n + 8$. Otherwise, using $L = n - 1$ and $\ell = n - r$, we obtain a bound of

$$\begin{aligned}(r-1)(2n-3) + n - r + 1 &= r(2n-4) - n + 4 \\ &\leq (n-1)(2n-4) - n + 4 \\ &= 2n^2 - 7n + 8.\end{aligned}$$

Similarly, one recovers the results of Rystsov [7] and the results of Carpi and d'Alessandro [21,22] with an improved bound. Indeed, the locally strongly transitive automata of [22] constitute the special case of Theorem 5 where $k = 1$.

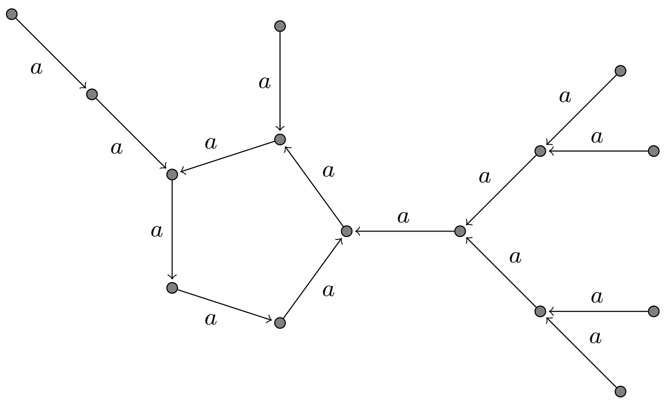


Fig. 2. a-skeleton of a one-cluster automaton with $n = 15$ and $r = 5$

Rystsov's notion of a regular automaton is essentially (but slightly more rigid) than the case $R = Q$.

The proof of Theorem 5 can easily be adapted to obtain the same bound if W is an arbitrary set of words such that there is a probability P_1 supported on W so that each column of $\pi(P_1)$ corresponding to an element of $Q \setminus R$ is 0, whereas each column corresponding to an element of R is $1/r[Q]^T$.

3 Proof of the Averaging Lemma

The proof of the Averaging Lemma rests on our observation about expectations of random variables and the ascending chain condition for finite dimensional vector spaces. Suppose that Σ^* acts on the left of a vector space V by linear maps. Let $X \subseteq \Sigma^*$ and let W be a subspace. Then by XW, we mean the span of all vectors xw with $x \in X$ and $w \in W$.

Lemma 6. *Let $\pi\colon \Sigma^* \to M_n(K)$ be a matrix representation with K a field. Suppose that one has subspaces $W, V \subseteq K^n$ of column vectors with $W \subseteq V$, but $\Sigma^* W \nsubseteq V$. Let S be a spanning set for W. Then there exist $s \in S$ and $w \in \Sigma^*$ with $|w| \leq \dim V - \dim W + 1$ and $ws \notin V$.*

Proof. Put $W_m = \Sigma^{\leq m} W$. Then there is an ascending chain of subspaces

$$W = W_0 \subseteq W_1 \subseteq W_2 \subseteq \cdots$$

and moreover as soon as this chain stabilizes it equals $\Sigma^* W$. By our assumption, there is a greatest $m \geq 0$ with $W_m \subseteq V$. In particular, the chain does not stabilize until after m steps and so

$$W_0 \subsetneq W_1 \subsetneq \cdots \subsetneq W_m \subseteq V$$

and hence $\dim W_0 + m \leq \dim V$, that is, $m + 1 \leq \dim V - \dim W + 1$. Therefore, there is a word $w \in \Sigma^*$ with $|w| \leq \dim V - \dim W + 1$ and $wW \nsubseteq V$. But W is spanned by S, so we can find $s \in S$ with $ws \notin V$.

Proof (of the Averaging Lemma). For convenience, put $X = \sigma(P_1)$. We show that for each $\emptyset \neq S \subsetneq R$, there exists $w \in \Sigma^*$ of length at most $n - c + L$ with $|Sw^{-1} \cap R| > |S|$ except for when $c = 2$ and $|S| = r/2$, in which case we can only guarantee that w has length at most $n - 1 + L$. If $R = Q$, the result is then immediate: one can find a state $q \in Q$ and a letter $a \in \Sigma$ so that $|qa^{-1}| > 1$; now we expand by inverse images $n - 2$ times with words of length at most $n-c+L$ (except for when $c = 2$ and $|S| = r/2$, in which case we expand by $n - 1 + L$) to obtain the result. If $R \subsetneq Q$, we can find w of length at most $(r-1)(n-c+L)+c-1$ with $|Rw| = 1$ using the same idea. Then as $Qw_0 \subseteq R$, it follows $|Qw_0w| \leq |Rw| = 1$. This yields the bound of $(r-1)(n-c+L)+\ell+c-1$ on the length a synchronizing word.

Consider the probability $P = P_2P_1$ on Σ^* and define a random variable $Z_S \colon \Sigma^* \to \mathbb{R}$ by

$$Z_S(w) = |Sw^{-1} \cap R| = [R][Sw^{-1}]^T = [R][w][S]^T.$$

Let us compute the expected value of this random variable:

$$\begin{aligned}\mathbf{E}_P(Z_S) &= \sum_{w\in\Sigma^*} P(w)|Sw^{-1} \cap R| = \sum_{w\in\Sigma^*} P(w)[R]w[S]^T \\ &= [R]P[S]^T = [R]P_2P_1[S]^T = [R][S]^T \\ &= |S|\end{aligned}$$

where we have used $[R]P_2P_1 = [R]$. The support of P is $\sigma(P_2)\sigma(P_1) = \Sigma^{\leq n-c}X$. If we can find $v \in \Sigma^{\leq n-c}X$ with $Z_S(v) = |Sv^{-1} \cap R| \neq |S|$, then we can find $w \in \Sigma^{\leq n-c}X$ with $|Sw^{-1} \cap R| = Z_S(w) > |S|$ by our discussion earlier on random variables that are not almost surely constant. As $|w| \leq n - c + L$, this will finish the proof.

If $|Sx^{-1} \cap R| \neq |S|$ for some $x \in X$, then we are done. Otherwise, we may assume $|Sx^{-1} \cap R| = |S|$ for all $x \in X$. Let γ be the column vector $[S]^T - (|S|/r)[Q]^T$. Notice that if $w \in \Sigma^*$, then one has $w\gamma = [Sw^{-1}]^T - (|S|/r)[Q]^T$ and so $[R]w\gamma = |Sw^{-1} \cap R| - |S|$. In particular, if $x \in X$ our assumption implies $[R]x\gamma = 0$. Moreover, $x\gamma \neq 0$ as $|S| < r$. Thus if W is the subspace spanned by the column vectors $x\gamma$ with $x \in X$, then $0 \neq W \subseteq [R]^{\perp}$.

Our next goal is to verify that $\dim W \geq c$ unless $c = 2$ and $|S| = r/2$ (in which case it is at least 1). The only non-trivial case is when $c = 2$ and $|S| \neq r/2$. Then we can find $w_1, w_2 \in X$ with $Sw_1^{-1} \neq Sw_2^{-1}$. We claim that $w_1\gamma$ and $w_2\gamma$ are linearly independent elements of W. Indeed, if they were linearly dependent, then since both vectors are non-zero we must have $w_1\gamma = kw_2\gamma$ for some $k \in \mathbb{R}$. Moreover, $k \neq 1$ because $Sw_1^{-1} \neq Sw_2^{-1}$. Thus $[Sw_1^{-1}]^T - k[Sw_2^{-1}]^T = (|S|/r)(1 - k)[Q]^T$. Since $[Q]^T$ is the all ones column vector and $[Sw_1^{-1}]^T$, $[Sw_2^{-1}]^T$ are column vectors of zeroes and ones, it follows that $k = -1$ and Sw_1^{-1}, Sw_2^{-1} are complementary subsets of Q. Then we obtain $[Q]^T = (2|S|/r)[Q]^T$, whence $|S| = r/2$, a contradiction. We conclude that $w_1\gamma$ and $w_2\gamma$ are linearly independent and so $\dim W \geq 2 = c$.

Our next claim is that $\Sigma^*W \nsubseteq [R]^{\perp}$. Indeed, let w be a synchronizing word. Then ww_0 synchronizes $\mathscr{A}$ to an element of $q \in R$. But $q\Sigma^* \supseteq R$, so we can synchronize to any state of R. In particular, we can synchronize $\mathscr{A}$ via some word y into $Sx^{-1} \cap R$ for some $x \in X$. Then $Sx^{-1}y^{-1} = Q$ and so $[R]yx\gamma = |Sx^{-1}y^{-1} \cap R| - |S| > 0$. This shows that $yx\gamma \notin R^{\perp}$ and hence $\Sigma^* W \nsubseteq [R]^{\perp}$. As $\dim W \geq c$ and $\dim[R]^{\perp} = n-1$, Lemma 6 now provides $u \in \Sigma^{\leq n-c}$ and $z \in X$ with $uz\gamma \notin [R]^{\perp}$. Putting $v = uz \in \Sigma^{\leq n-c}X$, we have $0 \neq [R]v\gamma = |Sv^{-1} \cap R| - |S|$. This completes the proof. □

Remark 7. The above proof and the proof of the main result of [28] give an improved bound for one-cluster automata. It is shown in [28] that if we have an n-state one-cluster automaton with unique a-cycle R of length r, then we can find a state $q \in R$ and a word w of length at most $2n-r-1$ such that $|qw^{-1} \cap R| > 1$. Since the Černý conjecture is proved for the case $r = n$ [5], we may assume $r \leq n-1$. Combining this with the above proof yields a bound of

$$\begin{aligned}(r-2)(2n-3)+2n-r-1+n-r+1 &= (r-2)(2n-3)+3n-2r\\ &= r(2n-5)-n+6\\ &\leq (n-1)(2n-5)-n+6\\ &= 2n^2-8n+11.\end{aligned}$$

Acknowledgments

I would like to thank Volker Diekert for a suggestion that led to the improvement of the bound in the Averaging Lemma in the case $c = 2$, as well as for detecting a minor flaw in an earlier version of this paper. Steffen Kopecki produced the diagram in Figure 2.

References

1. Černý, J.: A remark on homogeneous experiments with finite automata. Mat.-Fyz. Časopis Sloven. Akad. Vied 14, 208–216 (1964)
2. Pin, J.E.: Sur un cas particulier de la conjecture de Cerny. In: Automata, languages and programming, Fifth Internat. Colloq., Udine. LNCS, vol. 62, pp. 345–352. Springer, Berlin (1978)
3. Pin, J.E.: Le problème de la synchronisation et la conjecture de Černý. In: Noncommutative structures in algebra and geometric combinatorics (Naples, 1978). Quad. "Ricerca Sci." CNR, Rome, vol. 109, pp. 37–48 (1981)
4. Arnold, F., Steinberg, B.: Synchronizing groups and automata. Theoret. Comput. Sci. 359, 101–110 (2006)
5. Dubuc, L.: Sur les automates circulaires et la conjecture de Černý. RAIRO Inform. Théor. Appl. 32, 21–34 (1998)
6. Ananichev, D.S., Volkov, M.V., Zaks, Y.I.: Synchronizing automata with a letter of deficiency 2. Theoret. Comput. Sci. 376, 30–41 (2007)
7. Rystsov, I.K.: Quasioptimal bound for the length of reset words for regular automata. Acta Cybernet. 12, 145–152 (1995)
8. Rystsov, I.K.: On the length of reset words for automata with simple idempotents. Kibernet. Sistem. Anal. 187, 32–39 (2000)

9. Almeida, J., Margolis, S., Steinberg, B., Volkov, M.: Representation theory of finite semigroups, semigroup radicals and formal language theory. Trans. Amer. Math. Soc. 361, 1429–1461 (2009)
10. Trahtman, A.N.: The Černý conjecture for aperiodic automata. Discrete Math. Theor. Comput. Sci. 9, 3–10 (2007) (electronic)
11. Trahtman, A.N.: An efficient algorithm finds noticeable trends and examples concerning the Černy conjecture. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 789–800. Springer, Heidelberg (2006)
12. Ananichev, D.S., Volkov, M.V.: Some results on Černý type problems for transformation semigroups. In: Semigroups and Languages, pp. 23–42. World Sci. Publ., River Edge (2004)
13. Kari, J.: Synchronizing finite automata on Eulerian digraphs. Theoret. Comput. Sci. 295, 223–232 (2003); Mathematical foundations of computer science (Mariánské Lázně, 2001)
14. Ananichev, D.S., Volkov, M.V.: Synchronizing generalized monotonic automata. Theoret. Comput. Sci. 330, 3–13 (2005)
15. Rystsov, I.: Reset words for commutative and solvable automata. Theoret. Comput. Sci. 172, 273–279 (1997)
16. Rystsov, I.C.: On the rank of a finite automaton. Kibernet. Sistem. Anal. 187, 3–10 (1992)
17. Steinberg, B.: Černý's conjecture and group representation theory. J. Algebr. Comb. 31, 83–109 (2010)
18. Kari, J.: A counter example to a conjecture concerning synchronizing words in finite automata. Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 146 (2001)
19. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)
20. Béal, M.P., Perrin, D.: A quadratic upper bound on the size of a synchronizing word in one-cluster automata. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 81–90. Springer, Berlin (2009)
21. Carpi, A., d'Alessandro, F.: The synchronization problem for strongly transitive automata. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 240–251. Springer, Heidelberg (2008)
22. Carpi, A., d'Alessandro, F.: The synchronization problem for locally strongly transitive automata. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 211–222. Springer, Heidelberg (2009)
23. Almeida, J., Steinberg, B.: Matrix mortality and the Černý-Pin conjecture. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 67–80. Springer, Berlin (2009)
24. Béal, M.P.: A note on Cerny's conjecture and rational series (2003) (unpublished)
25. Salomaa, A.: Composition sequences for functions over a finite domain. Theoret. Comput. Sci. 292, 263–281 (2003); Selected papers in honor of Jean Berstel
26. Pin, J.E.: On two combinatorial problems arising from automata theory. In: Combinatorial Mathematics (Marseille-Luminy, 1981). North-Holland Math. Stud., vol. 75, pp. 535–548. North-Holland, Amsterdam (1983)
27. Berstel, J., Reutenauer, C.: Rational series and their languages. EATCS Monographs on Theoretical Computer Science, vol. 12. Springer, Berlin (1988)
28. Béal, M.P., Berlinkov, M.V., Perrin, D.: A quadratic upper bound on the size of a synchronizing word in one-cluster automata. International Journal of Foundations of Computer Science (to appear)

Pseudo-power Avoidance

Ehsan Chiniforooshan, Lila Kari[⋆], and Zhi Xu

Department of Computer Science, University of Western Ontario
London, Ontario, Canada, N6A 5B7
{ehsan,lila,zhi_xu}@csd.uwo.ca

Since Thue's work [10] in the early 1900's, repetition avoidance has been intensely studied [9,8,7,4]. From the point of view of DNA computing [5], we study another type of repetition, called a pseudo-power, inspired by the property of the Watson-Crick complementarity in molecular biology.

A *deoxyribonucleic acid* (DNA) single strand can be viewed as a string over the four-letter alphabet $\{A, C, G, T\}$, wherein A is the complement of T, while C is the complement of G. Such a DNA single strand will bind to a reverse complement DNA single strand, called its Watson-Crick complement, to form a helical double-stranded DNA molecule. The Watson-Crick complement of a DNA strand is deducible from, and thus informationally equivalent to, the original strand. We use this fact to generalize the notion of the power of a word by relaxing the meaning of "sameness" to include the image through an *antimorphic involution* (also called an *involutory antimorphism* in the literature), the model of DNA Watson-Crick complementarity. Similar generalizations exists in the literature for other concepts in combinatorics on words, such as pseudo-primitive words [2] and pseudo-palindromes [3,6,1].

Given a finite alphabet Σ, an antimorphic involution is a function $\theta : \Sigma^* \longrightarrow \Sigma^*$ which is an involution, i.e., θ^2 equals the identity, and an antimorphism, i.e., $\theta(uv) = \theta(v)\theta(u)$, for all $u, v \in \Sigma^*$. For a positive integer k, we call a word w a *pseudo-kth-power with respect to θ* if it can be written as $w = u_1 \ldots u_k$, where for $1 \leq i, j \leq k$ we have either $u_i = u_j$ or $u_i = \theta(u_j)$. The classical kth-power of a word is a special case of a pseudo-kth-power, where all the repeating units are identical.

We classify the alphabets Σ and the antimorphic involutions θ for which there exist arbitrarily long words that do not contain pseudo-kth-powers as a factor (*pseudo-kth-power-free*). We show that, for any Σ with $|\,\Sigma\,| \geq 4$ (resp., $|\,\Sigma\,| \geq 3$), and any antimorphic involution θ over Σ, there exist pseudo-square-free (resp., pseudo-cube-free) infinite words with respect to θ. No pseudo-square-free infinite word exists over Σ with $|\,\Sigma\,| \leq 2$ and the existence of pseudo-square-free infinite words depends on θ for $|\,\Sigma\,| = 3$. No pseudo-cube-free infinite word exists over Σ with $|\,\Sigma\,| \leq 2$. For any integer $k \geq 4$, pseudo-kth-power-free infinite words exist except when either $|\,\Sigma\,| = 1$ or $|\,\Sigma\,| = 2$ and θ transposes the two letters.

[⋆] This research was supported by Natural Sciences and Engineering Research Council of Canada Discovery Grant R2824A01, and Canada Research Chair Award to L.K.

We discuss algorithms for testing whether a word w of length N is pseudo-kth-power-free. For arbitrary integer k, we provide an $O(N^2 \log N)$-time algorithm to find all pseudo-kth-powers in w. In addition, we provide an $O(N)$-time algorithm and an $O(N^2)$-time algorithm for testing whether w is pseudo-square-free and pseudo-cube-free, respectively. The computational complexity of an optimal algorithm for testing whether w is pseudo-kth-power-free is still unknown.

References

1. Anne, V., Zamboni, L., Zorca, I.: Palindromes and pseudo-palindromes in episturmian and pseudo-palindromic infinite words. In: Brlek, S., Reutenauer, C. (eds.) Publications du LACIM, Proc. of Words 2005, vol. 36, pp. 91–100 (2005)
2. Czeizler, E., Kari, L., Seki, S.: On a special class of primitive words. Theoret. Comput. Sci. 411(3), 617–630 (2010)
3. de Luca, A., de Luca, A.: Pseudopalindrome closure operators in free monoids. Theoret. Comput. Sci. 362, 282–300 (2006)
4. Dekking, F.: Strongly non-repetitive sequences and progression-free sets. J. Combin. Theory Ser. A 27(2), 181–185 (1979)
5. Kari, L.: DNA computing — the arrival of biological mathematics. Math. Intelligencer 19(2), 9–22 (1997)
6. Kari, L., Mahalingam, K.: Watson-Crick palindromes in DNA computing. Nat. Comput. (2009), doi:10.1007/s11047-009-9131-2
7. Keränen, V.: Abelian squares are avoidable on 4 letters. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 41–52. Springer, Heidelberg (1992)
8. Leech, J.: A problem on strings of beads. Math. Gaz. 41(338), 277–278 (1957)
9. Morse, H.: Recurrent geodesics on a surface of negative curvature. Trans. Amer. Math. Soc. 22(1), 84–100 (1921)
10. Thue, A.: Über unendliche Zeichenreihen. Norske Vid. Selsk. Skr. I. Mat.-Nat. Kl. (7), 1–22 (1906)

On Restricted Context-Free Grammars

Jürgen Dassow[1] and Tomáš Masopust[2,⋆]

[1] Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
PSF 4120, D-39016 Magdeburg, Germany
dassow@iws.cs.uni-magdeburg.de

[2] Institute of Mathematics, Czech Academy of Sciences
Žižkova 22, CZ-61662 Brno, Czech Republic
masopust@ipm.cz

In context-free grammars, each derivation step can be characterized so that (i) a nonterminal of the current sentential form is chosen and (ii) rewritten by a rule. However, it is well-known that context-free grammars are not able to cover all aspects of natural languages and/or programming languages. Therefore, there were defined many grammars with context-free rules and some mechanism controlling the application of rules, e. g., in random context grammars and their variants, a rule is only applicable if the current sentential form contains some letters or subwords and some letters or words do not occur in it. Therefore, in grammars controlled by context, each derivation step can be characterized so that (i) subsets of applicable nonterminals and rules are determined according to the symbols appearing in the current sentential form, (ii) an applicable nonterminal is chosen and (iii) rewritten by an applicable rule.

In [2] a simpler type of context-free grammars with a controlled derivation process was introduced, which is formally defined as follows.

A restricted context-free grammar *is a tuple* $G = (N, T, P, S, f)$ *where* N *is the alphabet of nonterminals,* T *is the alphabet of terminals such that* $N \cap T = \emptyset$*,* $S \in N$ *is the axiom or start symbol,* P *is a finite set of context-free rules (i. e., each rule has the form* $A \to w$ *with* $A \in N$ *and* $w \in (N \cup T)^*$*), and* $f : N \to \{+, -\} \times N$ *is a function which maps any nonterminal to a signed nonterminal.*

We say that x *directly derives* y *in* G*, written as* $x \Longrightarrow y$*, if the following two conditions are satisfied:*
– $x = x_1 A x_2$*,* $y = x_1 w x_2$*,* $A \to w \in P$*,*
– $f(A) = (+, B)$ *implies that* x *contains* B*, and*
  $f(A) = (-, B)$ *implies that* x *does not contain* B*.*

The language $L(G)$ *of a restricted context-free grammar* $G = (N, T, P, S, f)$ *is defined by* $L(G) = \{z \mid z \in T^*,\ S \Longrightarrow^* z\}$*, where* $\Longrightarrow^*$ *is the reflexive and transitive closure of the relation* $\Longrightarrow$*.*

In the case of restricted context-free grammars the derivation step can be characterized so that (i) a set of applicable nonterminals is determined according to the symbols appearing in the current sentential form, (ii) an applicable nonterminal is chosen and (iii) rewritten by an arbitrary rule rewriting this nonterminal.

⋆ The research by T. Masopust was supported by the Czech Academy of Sciences, Institutional Research Plan No. AV0Z10190503.

We also consider the special case of *permitting* and non-erasing restricted context-free grammars where we require that $f : N \to \{+\} \times N$ holds and P does not contain rules of the form $A \to \lambda$, respectively.

We denote the families of languages generated by restricted context-free grammars, non-erasing restricted context-free grammars, permitting restricted context-free grammars, and permitting non-erasing restricted context-free grammars by $\mathcal{L}(rCF)$, $\mathcal{L}(rCF-\lambda)$, $\mathcal{L}(rCF_+)$, and $\mathcal{L}(rCF_+ - \lambda)$, respectively.

The families of context-free, context-sensitive and recursively enumerable languages are denoted by $\mathcal{L}(CF)$, $\mathcal{L}(CS)$ and $\mathcal{L}(RE)$, respectively. Furthermore, we designated the families of languages generated by random context grammars, non-erasing random context grammars, permitting random context grammars, and permitting non-erasing random context grammars by $\mathcal{L}(RC)$, $\mathcal{L}(RC-\lambda)$, $\mathcal{L}(P)$, and $\mathcal{L}(P-\lambda)$, respectively. For the definition of random context grammars, their variants and matrix grammars, we refer to [1] and [3].

Our first result characterizes the generative power of restricted context-free grammars. Essentially, their capacity coincides with that of the corresponding type of random context-grammars.

Theorem 1. *i)* $\mathcal{L}(CF) \subset \mathcal{L}(rCF_+) = \mathcal{L}(P) \subset \mathcal{L}(rCF) = \mathcal{L}(RC) = \mathcal{L}(RE)$.
ii) $\mathcal{L}(CF) \subset \mathcal{L}(rCF_+ - \lambda) = \mathcal{L}(P-\lambda) \subset \mathcal{L}(rCF-\lambda) = \mathcal{L}(RC-\lambda) \subset \mathcal{L}(CS)$.

As a consequence we obtain new normal form results for random context grammars and matrix grammars where we use only rules, which are much simpler than those in other known normal forms.

Corollary 1. *For any language $L \in \mathcal{L}(RE)$ $(L \in \mathcal{L}(RC-\lambda))$, there is a (non-erasing) random context grammar $G = (N, T, P, S)$ with the following properties:*
– $L(G) = L$,
– if $(A \to w, Q, R) \in P$, then $|w| \le 2$ and $\#(Q \cup R) = 1$, and
– if $(A \to w_1, Q_1, R_1) \in P$ and $(A \to w_2, Q_2, R_2) \in P$, then $Q_1 = Q_2$ and $R_1 = R_2$.

Corollary 2. *For any language $L \in \mathcal{L}(RE)$ $(L \in \mathcal{L}(RC-\lambda))$, there is a (non-erasing) matrix grammar $G = (N \cup \{Z\}, T, M, S, F)$, $Z \notin N \cup T$, with the following conditions:*
– $L(G) = L$,
– any matrix has the form $[A \to A, B \to x]$ with $A, B \in N$, $|w| \le 2$ or
$[A \to Z, B \to x]$ with $A, B \in N$, $|w| \le 2$, and
– F consists of all rules of the form $A \to Z$ occurring in matrices of M.

References

1. Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. Springer, Berlin (1989)
2. Masopust, T.: Simple Restriction in Context-Free Rewriting. (submitted for publication)
3. Meduna, A., Švec, M.: Grammars with Context Conditions and Their Applications. John Wiley & Sons, New York (2005)

Graphs Capturing Alternations in Words

Magnús M. Halldórsson[1], Sergey Kitaev[1], and Artem Pyatkin[2]

[1] School of Computer Science, Reykjavík University, 101 Reykjavik, Iceland
{mmh@ru.is,sergey}@ru.is
[2] Sobolev Institute of Mathematics, pr-t Koptyuga 4, 630090, Novosibirsk, Russia
artem@math.nsc.ru

A graph $G = (V, E)$ is representable if there exists a word W over the alphabet V such that letters x and y alternate in W if and only if $(x, y) \in E$ for each $x \neq y$. If W is k-uniform (each letter of W occurs exactly k times in it) then G is called k-representable. A graph is representable if and only if it is k-representable for some k [1].

In this note, we introduce the applicability of representable graphs, and answer several open questions from [1].

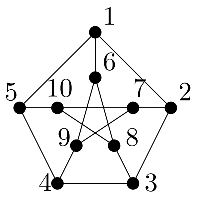
Circular precedence constraints: Consider a scenario with n recurring tasks with requirements on the alternation of certain pairs of tasks. This captures typical situations in periodic scheduling, where there are recurring *precedence* requirements, e.g., "before each ignition, check the oil level". When tasks occur only once, the pairwise requirements form precedence constraints, which are modeled by partial orders. When the directionality of the constraints is omitted, the resulting pairwise constraints form *comparability graphs.* We consider here graphs formed by pairwise alternation constraints

Execution sequences of recurring tasks can be viewed as words over an alphabet V, where V is the set of tasks. Thus, when tasks recur, the resulting alternation relationship forms a *representable* graph.

Proposition 1 ([1]). *Let $W = AB$ be a k-uniform word representing a graph G. Then the word $W' = BA$ also k-represents G.*

Representability of the Petersen graph: It was shown in [3] to be 3-representable:
- 1, 3, 8, 7, 2, 9, 6, 10, 7, 4, 9, 3, 5, 4, 1, 2, 8, 3, 10, 7, 6, 8, 5, 10, 1, 9, 4, 5, 6, 2
- 1, 3, 4, 10, 5, 8, 6, 7, 9, 10, 2, 7, 3, 4, 1, 2, 8, 3, 5, 10, 6, 8, 1, 9, 7, 2, 6, 4, 9, 5

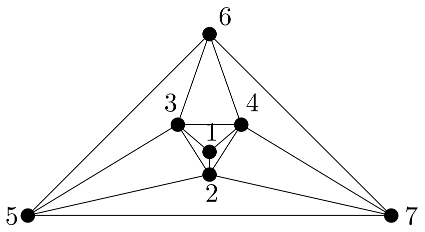
We can show that it is not 2-representable. Let W be a word 2-representing it. Some letter x must appear with the exactly three distinct letters between its two appearances. By symmetry and Prop. 1, $x = 1$ and W starts with 1. By symmetry and independence of 2,5,6, we can write $W = 12561W_1 6W_2 5W_3 2W_4$. To alternate with 6 but not to with 5, both W_1 and W_2 contain 8. To alternate with 2 but not with 5, both W_3 and W_4 contain 3. But then 8833 is a subsequence in W, so 8 and 3 are non-adjacent in the graph, a contradiction.



On forming non-representable graphs: The following open problem was posed in [1]: *Are there any non-representable graphs that do not satisfy the conditions of Theorem 1 below?*

Theorem 1. *([1]) If G is representable, then for every $x \in V(G)$ the graph induced by $N(x)$ is permutationally representable, where $N(x)$ is the set of neighbors of x in G.*

We give a positive answer. A counterexample to the converse of Theorem 1 is given by the graph below called co-(T_2) in [4]. It is easy to check that the induced neighborhood of any node of the graph co-(T_2) is a comparability graph.



Theorem 2. *The graph co-(T_2) is non-representable.*

Proof. Assume that co-(T_2) is k-representable for some k and W is a word representing it. The vertices 1,2,3,4 form a clique; so, their appearances $1^i, 2^i, 3^i, 4^i$ in W must be in the same order for each $i = 1, 2, \ldots, k$. By symmetry and Proposition 1 we may assume that the order is 1234. Now let $I_1, I_2, \ldots, I_k$ be the set of all $[2^i, 4^i]$-intervals in W. Two cases are possible.

1. There is an interval I_j such that 7 belongs to it. Then since 2,4,7 form a clique, 7 must be inside each of the intervals $I_1, I_2, \ldots, I_k$. But then 7 is adjacent to 1, a contradiction.
2. 7 does not belong to any of the intervals $I_1, I_2, \ldots, I_k$. Again, since 7 is adjacent to 2 and 4, each pair of consecutive intervals I_j, I_{j+1} must be separated by a single 7. But then 7 is adjacent to 3, a contradiction.

The effect of graph operations: Finally, we observe that the following operation on a representable graph preserves representability: Replace any node with a comparability graph, connecting all the new nodes to the neighbors of the original node. I.e., replacing a node with a comparability graph *module*. On the other hand, several other operations on representable graphs do not necessarily result in a representable graph: Taking the complement, taking the line graph, or identifying cliques of size more than 1 from two representable graphs.

References

1. Kitaev, S., Pyatkin, A.: On representable graphs. Automata, Languages and Combinatorics 13(1), 45–54 (2008)
2. Kitaev, S., Seif, S.: Word problem of the Perkins semigroup via directed acyclic graphs. Order (2008), doi:10.1007/s11083-008-9083-7
3. Konovalov, A., Linton, S.: On word representations of the Petersen graph. CIRCA preprint 2010/8, University of St. Andrews (2010)
4. `http://wwwteo.informatik.uni-rostock.de/isgci/smallgraphs.html`

On the Iterated Hairpin Completion

Steffen Kopecki

Universität Stuttgart, FMI
steffen.kopecki@fmi.uni-stuttgart.de

Abstract. The hairpin completion is a natural operation on formal languages which has been inspired by biochemistry and DNA-computing. In this paper we solve two problems which were posed first in 2008 and 2009, respectively, and still left open:

1.) It is known that the iterated hairpin completion of a regular language is not context-free in general, but it was open whether the iterated hairpin completion of a singleton or finite language is regular or at least context-free. We will show that it can be non-context-free.

2.) A restricted but also very natural variant of the hairpin completion is the bounded hairpin completion. It was unknown whether the iterated bounded hairpin completion of a regular language remains regular. We prove that this is indeed the case.

1 Introduction

The inspiration of the hairpin completion is rooted in DNA-computing and biochemistry, where it appears naturally in chemical reactions. It turned out that the corresponding operation on formal languages gives rise to very interesting and quite subtle decidability and computational problems. The focus of our paper is therefore on these formal language theoretical results.

Consider an alphabet Σ together with an involution $\overline{} : \Sigma \to \Sigma$ (i.e., $\overline{\overline{a}} = a$ for all $a \in \Sigma$). We say $\overline{a}$ is the complement of a. For words $\overline{a_1 \cdots a_n} = \overline{a_n} \cdots \overline{a_1}$. The hairpin completion is best explained by Fig. 1. If a word has a factorization $\gamma\alpha\beta\overline{\alpha}$ (a), then the suffix $\overline{\alpha}$ can bind to the infix α and form a hairpin (b). By complementing the unbound prefix γ, the hairpin completion (c) arises.

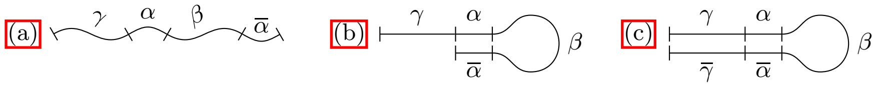


Fig. 1. Hairpin completion of a word

Let $k \geq 1$ be a small constant. The *one-sided hairpin completion* of a formal language L is defined as *os*-$\mathcal{H}_k(L) = \{\gamma\alpha\beta\overline{\alpha}\overline{\gamma} \mid \gamma\alpha\beta\overline{\alpha} \in L \wedge |\alpha| = k\}$. For the *(two-sided) hairpin completion* we also allow the suffix of a word to belong to L, this leads to $\mathcal{H}_k(L) = \{\gamma\alpha\beta\overline{\alpha}\overline{\gamma} \mid (\gamma\alpha\beta\overline{\alpha} \in L \vee \alpha\beta\overline{\alpha}\overline{\gamma} \in L) \wedge |\alpha| = k\}$. See, e.g., [1,5]. A familiar operation has been introduced in [2], the *bounded hairpin* *completion* $\mathcal{H}_{k,\ell}(L) = \{\gamma\alpha\beta\overline{\alpha}\overline{\gamma} \mid (\gamma\alpha\beta\overline{\alpha} \in L \vee \alpha\beta\overline{\alpha}\overline{\gamma} \in L) \wedge |\gamma| \leq \ell \wedge |\alpha| = k\}$, where the length of the γ-part is bounded by a constant $\ell \geq 0$.

The *iterated hairpin completion* of a formal language L is defined as $\mathcal{H}_k^*(L) = \bigcup_{i \geq 0} \mathcal{H}_k^i(L)$ where $\mathcal{H}_k^0(L) = L$ and $\mathcal{H}_k^{i+1}(L) = \mathcal{H}_k(\mathcal{H}_k^i(L))$. The *iterated one-sided hairpin completion os-*$\mathcal{H}_k^*(L)$ and the *iterated bounded hairpin completion* $\mathcal{H}_{k,\ell}^*(L)$ are defined analogously.

2 Results

Our first result concerns the iterated hairpin completion of singletons. We prove that a singleton exists whose iterated hairpin completion is non-context-free. This was stated as an open problem in [4]. It was even unknown if a singleton exits whose iterated hairpin completion is non-regular.

Theorem 1. *The iterated one- and two-sided hairpin completion of a singleton (or finite language) is context-sensitive and is not context-free in general.*

The next result gives a new representation for the iterated bounded hairpin completion.

Theorem 2. *Let L be a formal language and $\ell \geq 0$. The iterated bounded hairpin completion $\mathcal{H}_{k,\ell}^*(L)$ can be represented by an expression using L and the operations union, intersection with regular sets, and concatenation with regular sets.*

Consequentially, all language classes which are closed under these operations — which includes the classes in the Chomsky Hierarchy — are closed under iterated parameterized hairpin completion, too. From [2] it is known that the classes of context-free, context-sensitive, and recursively enumerable languages are closed under iterated bounded hairpin completion, but the status for regular languages was unknown. From Thm. 2 we conclude:

Corollary 1. *The class of regular languages is closed under iterated bounded hairpin completion.*

For the formal proofs of our results we refer to the technical report [3].

References

1. Cheptea, D., Martin-Vide, C., Mitrana, V.: A new operation on words suggested by DNA biochemistry: Hairpin completion. Trans. Comp., 216–228 (2006)
2. Ito, M., Leupold, P., Mitrana, V.: Bounded hairpin completion. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 434–445. Springer, Heidelberg (2009)
3. Kopecki, S.: On the Iterated Hairpin Completion. Technical Report Computer Science 2010/02, University of Stuttgart (May 2010)
4. Manea, F., Mitrana, V., Yokomori, T.: Some remarks on the hairpin completion. In: Proceedings of 12th International Conference AFL 2008, pp. 302–312 (2008)
5. Mitrana, V., Manea, F., Martín-Vide, C.: On some algorithmic problems regarding the hairpin completion. Electronic Notes in Discrete Mathematics 27, 71–72 (2006)

On Lookahead Hierarchies for Monotone and Deterministic Restarting Automata with Auxiliary Symbols (Extended Abstract)

Natalie Schluter

IT University of Copenhagen
Rued Langgaards Vej 7, 2300 Copenhagen S., Denmark
nael@itu.dk

A restarting automaton is a special type of linearly bounded automaton with fixed lookahead length k, whose computation proceeds in cycles performing one length-reducing rewrite of the lookahead contents per cycle as the only modification of the input. Through various restrictions on this machine model, a vast number of traditional and new language classes have been excavated. In two studies on lookahead hierarchies for restarting automata without auxiliary symbols [2,3], it was shown that lookahead length is often a significant restriction on the power of these types of restarting automata. No similar study on lookahead hierarchies for restarting automata with auxiliary symbols has been explicitly carried out. Such a study could provide more insight into the most important open problem concerning restarting automata: $\mathcal{L}(RRWW) =? \mathcal{L}(RWW)$. In this paper, we present a focussed study on language hierarchies for monotone or deterministic restarting automata with auxiliary symbols, whose rewrite step is separated from their restart step, to those for which these two steps are not separated. In doing so, we establish tight hierarchies for such restarting automata, the proofs of which are essentially a generalisation of the simulation of (right-) monotone RRWW-automata by a pushdown automaton in [1]. We show, in particular, that the class of languages for any type of right- or left- monotone restarting automaton (deterministic or not) whose restart step and rewrite step are separated coincides with that of the same monotone restarting automaton whose restart and rewrite steps are not separated, for any fixed lookahead size; furthermore, for the right- and left-monotone (non-deterministic) cases, the lookahead hierarchies collapse. For the non-monotone deterministic case, the lookahead length must be approximately doubled.

Theorem 1. *For each, X-RRWW(k)-automaton M_1, there is an X-RWW(k)-automaton M_2, such that $\mathcal{L}(M_1) = \mathcal{L}(M_2)$, when $k > 1$ and $X \in \{$mon, det-mon, left-mon, right-left-mon, det-left-mon, det-right-left-mon$\}$, where k is the lookahead length.*

Proof idea of Theorem 1. The proof constructs an mon-RWW-automaton M_2 to simulate an mon-RRWW-automaton M_1. The computations of M_1 in each cycle can be partitioned into computations preceding the rewrite step (left-computations), the rewrite step, and computations following the rewrite step (right-computations). M_2 will simulate M_1's previous right-computations in later cycles or in the tail phase of its computation. M_2 does this by recording the state that M_1 would have been in following the rewrite (if M_1 does not also restart at that point) in a compound symbol of its tape when rewriting. When M_2 finds a compound symbol, and providing the information it gives about how to continue simulating M_1's right-computations is accurate for the current phase, it is able to carry out simulation of (part of) the preceding right-computations (of M_1) that it was not able to carry out when producing this compound symbol, because it had to restart.

If all computations considered are monotone, then there is no need to check the accuracy of information collected from a compound symbol regarding how to continue with preceding right-computations. When M_2 finds two successive compound symbols, it will always abandon the information contained in the former for that of latter, for monotonicity, or incorporate all information found, in the case of left-monotonicity.

For at least the non-deterministic case, the hierarchy collapses.

Corollary 1. *For all $k \geq 3$ and $X \in \{mon, left\text{-}mon\}$, we have $\mathcal{L}(X\text{-}RWW(k)) = \mathcal{L}(X\text{ -}RRWW(k)) = CFL$, where k is the lookahead length.*

For deterministic case, one must approximately double the lookahead length.

Theorem 2. *For each, det-RRWW(k)-automaton M_1, there is a det-RWW(2k-2)-automaton, M_2 such that $\mathcal{L}(M_1) = \mathcal{L}(M_2)$, $k > 1$, where k and $2k-2$ the respective lookahead lengths.*

Proof idea for Theorem 2. The proof builds on that of Theorem 1. Every det-RRWW(k) automaton is weakly j-monotone for some $j \leq k-2$ [3]. So if we extend M_2's lookahead length by $k-2$ and rewrite the first k letters of the lookahead (as M_1 would), while just reprinting or marking the other $k-2$, we should obtain an automaton that, on the current phase, will only rewrite when it has seen all of the result of the rewrite from the last phase.

Monotone segments of computations are carried out similarly to the part of the proof of Theorem 1 for monotonicity: M_2 abandons information from former compound symbols (here, marked regions) for new compound symbols (marked regions). Non-monotone segments (where M_1 has a rewrite back-up of at most $k-2$) are marked so that M_2 knows to merge all information collected in these segments and abandon nothing as in the part of the proof of Theorem 1 for left-monotonicity.

References

1. Jančar, P., Mráz, F., Plátek, M., Vogel, J.: On monotonic automata with a restart operation. Journal of Automata, Languages and Combinatorics 4(4), 287–312 (1999)
2. Mráz, F.: Lookahead hierarchies of restarting automata. Journal of Automata, Languages and Combinatorics 6(4), 493–506 (2001)
3. Mráz, F., Otto, F.: Hiearchies of weakly monotone restarting automata. RAIRO-Theoretical Informatics and Applications 39(2), 325–342 (2005)

Joint Topologies for Finite and Infinite Words

Ludwig Staiger

Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg
D06099 Halle (Saale), Germany
staiger@informatik.uni-halle.de

Abstract. Infinite words are often considered as limits of finite words. As topological methods have been proved to be useful in the theory of ω-languages it seems to be providing to include finite and infinite words into one (topological) space. The attempts so far (see [3, Section 2.4]) have their drawbacks.

Therefore, in the present paper we investigate the possibility to join separate topologies on the space of finite words with a topology in the space of infinite words via a natural mapping. A requirement in this linking of topologies consists in the compatibility of topological properties (opennenss, closedness etc) of images with pre-images and vice versa.

Here we choose the natural CANTOR topology for infinite words and the δ-limit as linking mapping, nad we show that several natural topologies on the space of finite words prove to be compatible with the topology of the CANTOR space. It is interesting to observe that besides the well-known prefix topology there are at least two more whose origin is from language theory, from the construction of centers and super-centers of languages.

These center- and supercenter-topologies on the space of finite words, $\mathcal{T}_c$ and $\mathcal{T}_{sc}$, respectively, fit into the class of $\mathcal{L}$-topologies investigated in [2]. Moreover they exhibit special properties within the classes of topologies compatible with the CANTOR topology.

The paper presents the main results, omitting, however, due to lack of space the results on $\mathcal{L}$-topologies. For notation see Sections 1 and 2.1 of [3], and for topological background see e.g. [1].

In Section 2.4 of [3] it was shown that one can link the prefix topology, that is, the right topology of the prefix order $\sqsubseteq$ on X^* via the δ-limit to CANTOR topology in such a way that

$$\begin{array}{rcl} W = \mathbf{A}(W) \subseteq X^*, \text{ closed} & \longrightarrow & W^\delta = F \subseteq X^\omega \text{ closed} \\ W = W \cdot X^* \subseteq X^*, \text{ open} & \longrightarrow & W^\delta = F \subseteq X^\omega \text{ open} \\ W \subseteq X^*, W \in \mathcal{B}(\mathbf{G}) & \longrightarrow & W^\delta = F \subseteq X^\omega, F \in \mathcal{B}(\mathbf{G}) \\ W \subseteq X^* \text{ a } (\sigma,\delta)\text{-set} & \longrightarrow & W^\delta = F \subseteq X^\omega, F \in \mathbf{F}_\sigma \cap \mathbf{G}_\delta \text{ and} \\ W \subseteq X^* \text{ arbitrary} & \longrightarrow & W^\delta = F \subseteq X^\omega, F \in \mathbf{G}_\delta\,. \end{array}$$

Moreover, every subset $F \subseteq X^\omega$ in a class on the right hand side has a δ-pre-image in the corresponding class on the left hand side. This observation leads to the following definition.

Definition 1 (Compatibility). A topology $\mathcal{T}$ on X^* is *compatible* with the Cantor topology on X^ω provided

1. If $F \subseteq X^\omega$ is closed (open) in Cantor space then $F = W^\delta$ for some $W \subseteq X^*$ closed (open) in $\mathcal{T}$, and
2. W^δ is closed (open) if $W \subseteq X^*$ is closed (open) in $\mathcal{T}$.

The prefix topology is, however, only a T_0-topology on X^*, in particular, not every finite subset of X^* is closed. In the sequel we present two T_1-topologies refining the prefix topology which are also compatible with the Cantor topology on X^ω. The first one is the coarsest T_1-topology refining the prefix topology. It can be described by the closure operator $\alpha_c(W) := W \cup \mathbf{A}(\mathbf{A}(W)^\delta)$.

Lemma 1. *1. The prefix topology is the coarsest topology having all subsets $\mathbf{A}(F) \subseteq X^*$ closed.*
2. The topology $\mathcal{T}_c$ defined by the closure operator α_c is the coarsest topology having all subsets $\mathbf{A}(F) \subseteq X^*$ and all finite subsets closed.
3. $\mathcal{T}_c$ is compatible with the Cantor *topology on X^ω.*

In the prefix topology and in $\mathcal{T}_c$ every subset $\mathbf{A}(F) \subseteq X^*$ is the smallest closed subset such that $\mathbf{A}(F)^\delta = F$ whenever F is closed. In the sequel we investigate topologies having this property.

Definition 2 (Strong compatibility). A topology $\mathcal{T}$ on X^* is *strongly compatible* with the Cantor topology provided $\mathcal{T}$ is compatible with the Cantor topology and

$$\forall F\big(\mathbf{A}(F) = \min\nolimits_{\subseteq}\{W : W \text{ is closed in } \mathcal{T} \wedge F \subseteq W^\delta\}\big).$$

Theorem 1. *A topology $\mathcal{T}$ on X^* is strongly compatible with the* Cantor *topology if and only if the corresponding closure operator $\alpha_{\mathcal{T}}$ satisfies $\alpha_{\mathcal{T}}(W) \supseteq \mathbf{A}(W^\delta)$.*

The prefix topology is the coarsest and the closure $\alpha_{sc}(W) := W \cup \mathbf{A}(W^\delta)$ defines the finest topology strongly compatible with the Cantor *topology.*

Corollary 1. *If a topology $\mathcal{T}$ on X^* is strongly compatible with the* Cantor *topology then every closed set in $\mathcal{T}$ is a (σ,δ)-subset of X^*.*

The converse of Corollary 1, however is not true. There are topologies $\mathcal{T}$ on X^* compatible but not strongly compatible with the Cantor topology such that every $\mathcal{T}$-closed set is a (σ,δ)-subset of X^*.

References

1. Engelking, R.: General Topology. PWN, Warszawa (1977)
2. Prodinger, H.: Topologies on free monoids induced by closure operators of a special type. RAIRO Inform. Théor. Appl. 14(2), 225–237 (1980)
3. Staiger, L.: ω-languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 339–387. Springer, Berlin (1997)

Author Index