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

Language and Automata Theory and Applications

LNCS 6638

5th International Conference, LATA 2011
Tarragona, Spain, May 2011
Proceedings

Springer

Lecture Notes in Computer Science 6638

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

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

Language and Automata Theory and Applications

5th International Conference, LATA 2011
Tarragona, Spain, May 26-31, 2011
Proceedings

Springer

Volume Editors

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

Shunsuke Inenaga
Kyushu University, Department of Informatics
744 Motooka, Fukuoka, 819–0395 Japan
E-mail: inenaga@c.csce.kyushu-u.ac.jp

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-21253-6 e-ISBN 978-3-642-21254-3
DOI 10.1007/978-3-642-21254-3

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: Applied for

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

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

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

Printed on acid-free paper

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

Preface

These proceedings contain the papers that were presented at the 5th International Conference on Language and Automata Theory and Applications (LATA 2011), held in Tarragona, Spain, during May 26–31, 2011.

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

LATA 2011 received 91 submissions. Each one was reviewed by four Program Committee members, many of whom consulted with external referees. After a thorough and lively discussion phase, the committee decided to accept 36 papers (which represents an acceptance rate of 39.56%). The conference program also included three invited talks and two invited tutorials. Part of the success in the management of such a large number of submissions is due to the excellent facilities provided by the EasyChair conference management system.

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

March 2011

Adrian-Horia Dediu
Shunsuke Inenaga
Carlos Martín-Vide

Organization

LATA 2011 was organized by the Research Group on Mathematical Linguistics (GRLMC), Rovira i Virgili University, Tarragona, Spain.

Program Committee

Andrew Adamatzky	Bristol, UK
Cyril Allauzen	New York, USA
Amihood Amir	Ramat-Gan, Israel
Franz Baader	Dresden, Germany
Marie-Pierre Béal	Marne-la-Vallée, France
Philip Bille	Lyngby, Denmark
Miklós Bóna	Gainesville, USA
Symeon Bozapalidis	Thessaloniki, Greece
Vasco Brattka	Cape Town, South Africa
Maxime Crochemore	London, UK
James Currie	Winnipeg, Canada
Jürgen Dassow	Magdeburg, Germany
Cunsheng Ding	Hong Kong, China
Rodney Downey	Wellington, New Zealand
Manfred Droste	Leipzig, Germany
Enrico Formenti	Nice, France
Amy Glen	Perth, Australia
Serge Haddad	Cachan, France
Shunsuke Inenaga (Co-chair)	Fukuoka, Japan
Jesper Jansson	Tokyo, Japan
Jarkko Kari	Turku, Finland
Marek Karpinski	Bonn, Germany
Maciej Koutny	Newcastle, UK
Gregory Kucherov	Lille, France
Markus Lohrey	Leipzig, Germany
Benedikt Löwe	Amsterdam, The Netherlands
Salvador Lucas	Valencia, Spain
Sebastian Maneth	Sydney, Australia
Carlos Martín-Vide (Co-chair)	Brussels, Belgium
Giancarlo Mauri	Milan, Italy
Alexander Meduna	Brno, Czech Republic
Kenichi Morita	Hiroshima, Japan
Sven Naumann	Trier, Germany
Gonzalo Navarro	Santiago, Chile
Mark-Jan Nederhof	St. Andrews, UK
Joachim Niehren	Lille, France

Joakim Nivre	Uppsala, Sweden
Kemal Oflazer	Doha, Qatar
Alexander Okhotin	Turku, Finland
Witold Pedrycz	Edmonton, Canada
Dominique Perrin	Marne-la-Vallée, France
Giovanni Pighizzini	Milan, Italy
Alberto Policriti	Udine, Italy
Lech Polkowski	Warsaw, Poland
Helmut Prodinger	Stellenbosch, South Africa
Mathieu Raffinot	Paris, France
Philippe Schnoebelen	Cachan, France
Ayumi Shinohara	Sendai, Japan
Jamie Simpson	Perth, Australia
Magnus Steinby	Turku, Finland
James Storer	Boston, USA
Jens Stoye	Bielefeld, Germany
Andrzej Tarlecki	Warsaw, Poland
Richard Thomas	Leicester, UK
György Vaszil	Budapest, Hungary
Heiko Vogler	Dresden, Germany
Pascal Weil	Bordeaux, France
Damien Woods	Pasadena, USA
Thomas Zeugmann	Sapporo, Japan

External Reviewers

B
Giorgio Bacci
Florent Becker
Djamal Belazzougui
Béatrice Bérard
Alberto Bertoni
Dietmar Berwanger
Benedikt Bollig
Luca Bortolussi
Patricia Bouyer-Decitre
Matthias Büchse
Wojciech Buszkowski

C
Antonio Cau
Jean-Marc Champarnaud
Christian Choffrut
Vincenzo Ciancia
Alexander Clark
Francisco Claude
Stefano Crespi Reghizzi

D
Stéphane Demri
Alberto Dennunzio
Alexander Dikovsky
Michael Domaratzki
Frank Drewes
Marie Duflot
Jacques Duparc

E
Péter L. Erdös

F
Jacques Farré
John Fearnley

Gabriele Fici
Emmanuel Filiot
Francesca Fiorenzi
Andrea Formisano
Kimmo Fredriksson
Christiane Frougny

G
Dora Giammarresi
Robert Giegerich
Hugo Gimbert
Valentin Goranko
Archontia Grammatikopoulou
Pierre Guillon

H
Christoph Haase
Vesa Halava
Yo-Sub Han
Reiko Heckel
José Hernández-Orallo
Larry Holder

I
Daisuke Ikegami
Chuzo Iwamoto

J
Florent Jacquemard
Artur Jeż
Timo Jolivet

K
Antonios Kalampakas
Tomi Kärki
Victor Khomenko
Bakhadyr Khoussainov
Nguyen Kim
Daniel Kirsten
Felix Klaedtke
Bartek Klin
Vladimir Komendantsky
Eryk Kopczynski
Manfred Kufleitner
Orna Kupferman
Alexander Kurz
Dietrich Kuske
Temur Kutsia

L
Martin Lange
Jia Lee
Tommi Lehtinen
Aurélien Lemay
Richard S. Lemence
Christof Löding
Sylvain Lombardy
Violetta Lonati
Jack Lutz

M
Andreas Maletti
Eleni Mandrali
Radu Mardare
Wim Martens
Miguel A. Martínez-Prieto
Ingmar Meinecke
Neeldhara Misra
Benjamin Monmege
Niall Murphy

N
Turlough Neary
Kim Nguyen
Cyril Nicaud
Lasse Nielsen
Christos Nomikos
Dirk Nowotka

O
Friedrich Otto

P
Miguel Palomino
Christophe Papazian
Madhusudan Parthasarathy
David Pearce
Dominique Perrin
Tamar Pinhas

Thomas Place
Franck Pommereau
Anne Preller

Q
Karin Quaas

R
Tomasz Radzik
George Rahonis
Narad Rampersad
Sarah Rees
Klaus Reinhardt
Andrei Romashchenko
Paolo Rosso

S
Carlo Sartiani
Sylvain Schmitz
Stefan Schwoon
Géraud Sénizergues
Frédéric Servais
Jeffrey Shallit
Otakar Smrž
Christian Soltenborn
Andrei Ştefănescu
Howard Straubing
Mari Carmen Suárez-Figueroa
Nathalie Sznajder

T
Kohtaro Tadaki
Jean-Marc Talbot
D. Gnanaraj Thomas
Jakob Thomsen
Cătălin Tîrnăucă
Cristina Tîrnăucă
German Tischler
Sophie Tison
Alexandru I. Tomescu
Tayssir Touili

U
Irek Ulidowski

V
Camille Vacher

W
Thomas Wächter
Dominic Welsh
Anthony Widjaja To
Roland Wittler
Ronald De Wolf
Paul Wollan

Z
Marc Zeitoun
Charalampos Zinoviadis

Organizing Committee

Adrian-Horia Dediu, Tarragona
Shunsuke Inenaga, Fukuoka (Co-chair)
Carlos Martín-Vide, Brussels (Co-chair)
Bianca Truthe, Magdeburg
Florentina-Lilica Voicu, Tarragona

Table of Contents

Invited Talks

Regular Papers

Green's Relations and Their Use in Automata Theory

Thomas Colcombet*

Liafa/Cnrs/Université Paris Diderot–Paris 7, France
thomas.colcombet@liafa.jussieu.fr

Abstract. The objective of this survey is to present the ideal theory of monoids, the so-called Green's relations, and to illustrate the usefulness of this tool for solving automata related questions.

We use Green's relations for proving four classical results related to automata theory: The result of Schützenberger characterizing star-free languages, the theorem of factorization forests of Simon, the characterization of infinite words of decidable monadic theory due to Semenov, and the result of determinization of automata over infinite words of McNaughton.

Introduction

In this lecture, we will establish several classical results related to automata theory, respectively due to Schützenberger, Simon, Semenov, and McNaughton. These problems are all related in a more or less direct way to language theory and automata. Despite their obvious intrinsic interest, these results will be for us excuses for presenting the approach via monoids and semigroups which allows to uniformly apprehend these, *a priori* unrelated, questions. That is why this lecture is structured as the interleaving of the proofs of the above results with the necessary algebraic material.

We devote a particular attention to the theory of ideals in monoids, the so called *Green's relations*. When working in language theory using automata, several tools comes naturally into play. A typical example is the use of the decomposition of the graph of the automaton into strongly connected components, and the use of the dag of the connected components for driving an induction in a proof. The Green's relations provide the necessary tools for using similar arguments on the monoid rather than on the automaton. Since monoids are more informative than automata, the resulting techniques are more powerful than the corresponding ones on automata (this gain usually comes at the price of a worth complexity in decision procedures and constructions). The Green's relations are well known, and presented in deep detail in several places, see for instance [5,13]. For this reason we do not establish here the results related to this theory.

* Supported by the project ANR 2010 BLAN 0202 02 FREC, and the ERC Starting Grant GALE.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 1–21, 2011.

We do not try either to be exhaustive in any way. Our goal is different. We are interested in illustrating how to use this tool.

We use four classical results as illustrations. The first one is the theorem of *Schützenberger* [17] characterizing the languages which can be described by star-free expressions. The second one is the theorem of factorization forests of *Simon* [19], which gives a form of generalized Ramsey argument for regular languages. The third one is a theorem of *Semenov* [18] which gives a necessary and sufficient condition for an infinite word to have a decidable monadic second-order theory. The fourth theorem, due to *McNaughton* [9], states that automata over infinite words can be made deterministic.

The lecture is structured as follows. We first briefly recall some basic definitions concerning semigroups and monoids in Section 1. We then present the results of Schützenberger and Simon in Section 2 and 3 respectively. We then introduce the framework of ω-semigroups in Section 4, and use it for establishing the results of semenov and McNaughton in Sections 5 and 6 respectively.

1 Basics on Monoids

A *monoid* $\mathbf{M}$ is a set together with an associative binary operator $\cdot$ which has a *neutral element* denoted 1 (such that $1x = x1 = x$ for all x). An element e such that $ee = e$ is called an *idempotent*. A *monoid morphism* from a monoid $\mathbf{M}$ to another $\mathbf{M}'$ is an application from M to M' such that $\alpha(1) = 1$, and $\alpha(ab) = \alpha(a)\alpha(b)$ for all a, b in $\mathbf{M}$.

A particular example is the *free monoid generated by a set* A, it is the set of words over the alphabet A equipped with the concatenation product. The neutral element is ε. An example of a finite monoid consists of the two elements a, b equipped with the product $aa = ab = ba = a$, and $bb = b$.

A language $L \subseteq A^*$ is recognizable by a monoid $(M, \cdot)$ if there exists a morphism α from A^* to $\mathbf{M}$ and a subset $F \subseteq M$ such that $L = \alpha^{-1}(F)$.

Theorem 1 (Rabin and Scott [16] with credit to Myhill). *A language of finite words over a finite alphabet is regular (i.e., accepted by some standard form of finite state automaton) if and only if it is recognizable by a finite monoid.*

Given a language L there is a particular, minimal, monoid which recognizes it, the syntactic monoid. The *syntactic monoid* of a language L over the alphabet A is an object gathering the minimal amount of information for each word that is relevant for the language. This is obtained by a quotient of words by the so-called *syntactic congruence* $\sim_L$. Two words are equivalent for this relation if they are undistinguishable by the language in any context. Formally, two words u and v are equivalent for a language L is defined as:

$$u \sim_L v \quad \text{if for all } x, y \in A^*,\ xuy \in L \text{ iff } xvy \in L\ .$$

If two words are equivalent for $\sim_L$, this means that in any context, one can safely exchange one for the other. In particular, as its name suggest, $\sim_L$ is a

congruence, i.e., $u \sim_L v$ and $u' \sim_L v'$ implies $uu' \sim_L vv'$. This means that the equivalence classes for $\sim_L$ can be equipped with a product. The resulting quotiented monoid $\mathbf{M}_L = A^*/_{\sim_L}$ is called the *syntactic monoid of* L. Furthermore, the application η_L which to a word associates its equivalence class is a morphism, called the *syntactic morphism*.

In particular, setting $F = \eta_L(L)$, we have $u \in L$ if and only if $\eta_L(u) \in F$. In other words, the syntactic monoid $\mathbf{M}_L$ recognizes L using the morphism η_L and the subset $F = \eta_L(L) \subseteq \mathbf{M}_L$.

For instance, consider the language over the alphabet $A = \{a, b, c\}$ consisting of "all words which do not contain two consecutive occurrences of the letter a". The equivalence classes of the syntactic monoid are ε, $a((b+c)^+a)^*$, $(a(b+c)^+)^+$, $((b+c)^+a)^+$, $(b+c)^+(a(b+c)^+)^*$ and A^*aaA^*. We will denote them by 1, a, ab, ba, b and 0 respectively. The notations 1 and 0 are conventional, and correspond to the neutral and the absorbing element (absorbing means $0x = x0 = 0$; such an element is unique, but does not always exist). For the other congruence classes, one fixes a word as representative. The product xy of any two elements x and y of the monoid is given in the following table:

$x \backslash y$	1	a	ab	ba	b	0
1	1	a	ab	ba	b	0
a	a	0	0	a	ab	0
ab	ab	a	ab	a	ab	0
ba	ba	0	0	ba	b	0
b	b	ba	b	ba	b	0
0	0	0	0	0	0	0

The language "no two consecutive occurrences of letter a" is recognized by this monoid, together with the morphism which maps letter a to a and letters b and c to b, and the subset $F = \{1, a, ab, ba, b\}$.

We see on this example that the table of product is not very informative for understanding the structure of the monoid. Natural tools, such as the strongly connected components of the graph of the automaton, are frequently used to design proofs and constructions in automata theory. We do not see immediately anything similar in monoids. The Green's relations that we present below gives such an insight in the structure of the monoid. Even better, since the syntactic monoid is more informative than the minimal automaton (at a price: it is also bigger), the structure of the syntactic monoid reveals even more information than the analysis of the minimal automaton.

Furthermore, the syntactic monoid is something one can work with:

Proposition 1. *The syntactic monoid is finite iff the language is regular. Furthermore, the syntactic monoid of a regular language can be effectively computed from any presentation of the language (by automata, regular expressions, etc...)*

2 Schützenberger's Characterization of Star-Free Languages

Our first result concerns star-free languages. The *star-free languages* are the languages of finite words which can be obtained from finite languages using union, concatenation and complement (of course, intersection and set difference can be derived from the union and complement).

An example is simply A^* (for A the alphabet), which is star-free since it is the complement of the empty language, which is itself finite. More generally, B^* for all subsets B of A is star-free since it can be written as $A^* \setminus (\bigcup_{c \notin B} A^* c A^*)$. Very close is the language of words over $A = \{a, b, c\}$ containing no two consecutive occurrences of the letter a. It is star-free since it can be written as $A^* \setminus (A^* a a A^*)$. However, the language of words of even size is not star-free (we do not prove it here). In general, all star-free languages are regular, but the converse does not hold. This motivates the following question:

When is a regular language star-free? Is it decidable?

Schützenberger answered the above question as follows:

Theorem 2 (Schützenberger[17]). *A regular language is star-free iff it is recognized by a monoid which has only trivial subgroups*[1].

This famous result is now well understood and has been enriched in many ways. In particular, star-free languages are known to coincide with first-order definable languages as well as with the languages accepted by counter-free automata [10]. This result was the starting point of the very important literature aiming in precisely classifying families of languages. See for instance [14].

This result in particular establishes the decidability of being star-free. Indeed, if any monoid recognizing a language has only trivial subgroups, then its syntactic monoid has only trivial subgroups. This yields a decision procedure: construct the syntactic monoid of the language and check that all its subgroups are trivial. The later can be done by an exhaustive check.

We will only prove here the right to left direction of Theorem 2, namely, if a regular language is recognized by some (finite) monoid with only trivial subgroups, then it is star-free. The interested reader can find good expositions of the other directions in many places, see for instance [11]. We assume from now and on that we are given a language L recognised by $\mathbf{M}, \alpha, F$, in which $\mathbf{M}$ is a finite monoid which has only trivial subgroups.

The general approach of the proof is natural. For all elements $a \in \mathbf{M}$, we prove that the language $L_a \stackrel{\text{def}}{=} \{u \in A \,:\, \alpha(u) = a\}$ is star-free. This concludes the proof of the right to left direction of Theorem 2, since it yields that

$$L = \alpha^{-1}(F) = \bigcup_{a \in F} L_a \quad \text{is star-free.}$$

[1] Here, a subgroup is a subset of the monoid which is equipped of a group structure by the product of the monoid. This terminology is the original one used by Schützenberger. It is natural in the general context of semigroup theory.

However, how do we prove that each L_a is star-free?

Our basic blocks will be languages consisting of words of length one. For each $a \in \mathbf{M}$, set $C_a \stackrel{\text{def}}{=} \{c \in A \ : \ \alpha(c) = a\}$. This is a finite (hence star-free) language, and $C_a \subseteq L_a$. More precisely, C_a captures all the length-one words in L_a. We could try to get closer to L_a using only concatenations and unions of such elementary languages. However, this would only produce finite languages. This is not sufficient in general. We need another argument. It will take the form of a good induction parameter: we will pre-order the elements of the monoid using one of Green's relations, the $\leq_{\mathcal{J}}$-pre-order, that we introduce now.

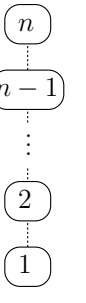
In a monoid $\mathbf{M}$, we define the binary relations $\leq_{\mathcal{J}}$ and $\mathcal{J}$ by:

$a \leq_{\mathcal{J}} b$ if $a = xby$ for some $x, y \in \mathbf{M}$, $\quad a \,\mathcal{J}\, b$ if $a \leq_{\mathcal{J}} b$ and $b \leq_{\mathcal{J}} a$.

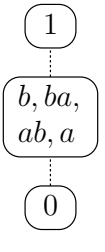
The relation $\leq_{\mathcal{J}}$ is a preorder, and $\mathcal{J}$ is an equivalence relation. In particular, in free monoid of words, $u \geq_{\mathcal{J}} v$ if and only if u appears as a *factor* of v, i.e., v can be written wuw' for some w and w'. We call this the *factor ordering*.

One often describes monoids making explicit the $\mathcal{J}$-pre-order, such as in the following examples.

Our first example is the monoid $(\{1, \ldots, n\}, \min)$. Here, the neutral element is n, and the absorbing element 1. In this case, $\leq_{\mathcal{J}}$ coincide with the usual order $\leq$.



Traditionally, the smaller is an element for the relation $\leq_{\mathcal{J}}$, the lower it appears in the drawing. When one starts from an element and successively performs products to the left or to the right, then the $\mathcal{J}$-class stays the same or goes down. In the later case, one says *falling* in a lower $\mathcal{J}$-class. This supports the intuition behind the relation $\leq_{\mathcal{J}}$ that it captures information that cannot be undone: it is never possible to climb back to 5 from 2 by making any product, to the left or to the right. Informally, "it is impossible to recover from falling in the $\leq_{\mathcal{J}}$ order".



Let us depict now the structure of $\mathcal{J}$-classes of the syntactic monoid of the language "no two consecutive occurrences of the letter a".

Remark the $\mathcal{J}$-equivalence between a, ab, ba, and b (in general, the $\mathcal{J}$-relation is not an order). However, as soon as two consecutive a's are encountered, one falls in the $\mathcal{J}$-class of 0. Once more it is impossible, using any product with a word containing two consecutive a's, to produce one without this pattern.

In general, falling in a $\mathcal{J}$-class can be understood as the discovery of a certain pattern as a factor (here aa, but in general any regular language).

The $\leq_{\mathcal{J}}$-pre-order is a good idea as an induction parameter. Indeed, unless $b \leq_{\mathcal{J}} a$, one does not see how knowing that L_a is star-free can help proving that L_b is also star-free. This is why our proof proceeds by establishing the following induction step.

Induction step: Assuming L_b is star-free for all $b >_{\mathcal{J}} a$, then L_a is star-free.

Assuming this induction step granted for all $a \in \mathbf{M}$, we obtain immediately that L_a is star-free for all $a \in \mathbf{M}$, and hence L itself is star-free. The theorem is established. Let us establish now this induction step. We assume from now and on a fixed, and the hypothesis of the induction step fulfilled. The key lemma is:

Lemma 1. *The language* $L_{\not\geq_{\mathcal{J}} a} \stackrel{def}{=} \{u \ : \ \alpha(u) \not\geq_{\mathcal{J}} a\}$ *is star-free.*

Proof. It follows from the following equation which witnesses the star-freeness:

$$L_{\not\geq_{\mathcal{J}} a} = A^* K_a A^* \ , \qquad \text{where} \quad K_a \stackrel{\text{def}}{=} \bigcup_{b \not\geq_{\mathcal{J}} a} C_b \cup \bigcup_{\substack{bcd \not\geq_{\mathcal{J}} a \\ c >_{\mathcal{J}} a}} C_b L_c C_d \ .$$

We prove this equality by establishing a double inclusion. The easiest direction is the inclusion from right to left. Indeed, we have $K_a \subseteq L_{\not\geq_{\mathcal{J}} a}$ by construction. Furthermore, by definition of the $\mathcal{J}$-pre-order, belonging to $L_{\not\geq_{\mathcal{J}} a}$ is preserved under any product to the left or to the right. Hence $A^* K_a A^* \subseteq L_{\not\geq_{\mathcal{J}} a}$.

For the opposite inclusion, we prove that every word in $L_{\not\geq_{\mathcal{J}} a}$ which is minimal (i.e., such that no strict factor belongs to $L_{\not\geq_{\mathcal{J}} a}$) belongs to K_a. Let u be such a minimal word. Clearly u cannot be of length 0, since this would mean $\alpha(u) = 1_{\mathbf{M}} \geq_{\mathcal{J}} a$, and hence $u \notin L_{\not\geq_{\mathcal{J}} a}$. If u has length 1 then it directly falls in $\bigcup_{b \not\geq_{\mathcal{J}} a} C_b$, which is the first part in the definition of K_a. The interesting case is when u has length at least 2. In this case, u can be written as $u = xvy$ for some letters $x, y \in A$.

Let $b = \alpha(x)$, $c = \alpha(v)$, and $d = \alpha(y)$. We claim that $c >_{\mathcal{J}} a$, and for proving this, we use the following lemma:

Lemma 2. *In a finite monoid, if* $c \,\mathcal{J}\, bc \,\mathcal{J}\, cd$, *then* $c \,\mathcal{J}\, bcd$.

This results is in fact a direct consequence of more elementary results presented below. Let us show this simplicity by giving a proof, though the necessary material has not been yet given.
Assume $c \,\mathcal{J}\, bc \,\mathcal{J}\, cd$, then by Lemma 7, $c \,\mathcal{R}\, cd$. Hence by Lemma 3, $bcd \,\mathcal{R}\, bc$. Thus $bcd \,\mathcal{J}\, bc$. □

For the sake of contradiction, assume $c \not>_{\mathcal{J}} a$. We have $a \leq_{\mathcal{J}} bc$ (by minimality in the choice of u), hence $a \leq_{\mathcal{J}} c$. Combined with $a \not<_{\mathcal{J}} c$, we obtain $a \,\mathcal{J}\, c$. Furthermore we have $c \,\mathcal{J}\, a \leq_{\mathcal{J}} bc \leq_{\mathcal{J}} c$, hence $bc \,\mathcal{J}\, c$ and similarly $cd \,\mathcal{J}\, c$. Using

Lemma 2, we get $bcd \mathcal{J} c \mathcal{J} a$. This contradicts $u \in L_{\not\geq_{\mathcal{J}} a}$ since $\alpha(u) = bcd$. Hence $c >_{\mathcal{J}} a$.

Thus, $u \in C_b L_c C_d$, $bcd \not\geq_{\mathcal{J}} a$, and $c >_{\mathcal{J}} a$, i.e., $u \in K_a$. Consider now a word $u \in L_{\not\geq_{\mathcal{J}} a}$. It has a minimal factor $u' \in L_{\not\geq_{\mathcal{J}} a}$. We have seen that $u' \in K_a$. Hence $u \in A^* K_a A^*$. □

We immediately deduce:

Corollary 1. *The language* $L_{\mathcal{J}a} = \{u \ : \ \alpha(u) \mathcal{J} a\}$ *is star-free.*

Proof. This follows from the equation $L_{\mathcal{J}a} = \bigcap_{b >_{\mathcal{J}} a} L_{\not\geq_{\mathcal{J}} b} \setminus L_{\not\geq_{\mathcal{J}} a}$. □

At this point, we are able to define the $\mathcal{J}$-class of a. However, we need to be even more precise, and define precisely a. We will need some more of Green's relations.

The order $\leq_{\mathcal{J}}$ makes no distinction on whether the products are performed on the left or on the right. The relations $\leq_{\mathcal{L}}$ and $\leq_{\mathcal{R}}$ refine the $\leq_{\mathcal{J}}$ order as follows:

$a \leq_{\mathcal{L}} b$ if $a = xb$ for some $x \in \mathbf{M}$, $\quad a \mathcal{L} b$ if $a \leq_{\mathcal{L}} b$ and $b \leq_{\mathcal{L}} a$,

$a \leq_{\mathcal{R}} b$ if $a = bx$ for some $x \in \mathbf{M}$, and $\quad a \mathcal{R} b$ if $a \leq_{\mathcal{R}} b$ and $b \leq_{\mathcal{R}} a$.

An elementary, but important, property of these relations is:

Lemma 3. *If* $a \leq_{\mathcal{L}} b$ *then* $ac \leq_{\mathcal{L}} bc$. *If* $a \mathcal{L} b$ *then* $ac \mathcal{L} bc$.
If $a \leq_{\mathcal{R}} b$ *then* $ca \leq_{\mathcal{R}} cb$. *If* $a \mathcal{R} b$ *then* $ca \mathcal{R} cb$.

The relation $\mathcal{L}$ considers equivalent elements which are convertible one into the other by product on the left. One can think of the piece of informations preserved by such conversions as located "close to the right" of the element. Considering that $\mathcal{L}$ identifies information concerning the "right" of the element, and $\mathcal{R}$ information which concerns the "left" of the element, the two relations $\mathcal{L}$ and $\mathcal{R}$ may seem rather independent. This intuitive idea is captured by the following key lemma.

Lemma 4. $\mathcal{L} \circ \mathcal{R} = \mathcal{R} \circ \mathcal{L}$.

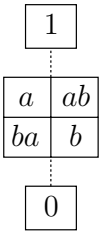
Thus, we define $\mathcal{D} \stackrel{\text{def}}{=} \mathcal{L} \circ \mathcal{R}$. In general, we have $\mathcal{D} \subseteq \mathcal{J}$, but:

Lemma 5. *In a finite monoid,* $\mathcal{D} = \mathcal{J}$.

This result is the central one in the theory of finite monoids. All results presented in this lecture using the finiteness assumption are more or less directly derived from this lemma.

A consequence for us is that we can depict monoids in a refined way.

The presentation of the syntactic monoid of the language "no two consecutive occurrences of a" can be refined as shown. The $\mathcal{J}$-class $\{a, ab, ba, b\}$ has been subdivided according to the $\mathcal{L}$-classes and the $\mathcal{R}$-classes, yielding an "egg-box" presentation of each class. The $\mathcal{R}$-classes are drawn as '$\mathcal{R}$'ows (here $\{1\}$, $\{a, ab\}$, $\{b, ba\}$ and $\{0\}$), and the $\mathcal{L}$-classes as columns (here $\{1\}$, $\{a, ba\}$, $\{b, ab\}$ and $\{0\}$). The last of Green's relations is $\mathcal{H}$, defined by:



$$\mathcal{H} \stackrel{\text{def}}{=} \mathcal{L} \cap \mathcal{R} \ .$$

Quite naturally $\mathcal{H}$-classes correspond to the atomic boxes at the intersection of rows and columns. In our example, all $\mathcal{H}$-classes are singletons.

For groups, on the contrary, there is only one $\mathcal{H}$-class.

Our next objective is to identify the $\mathcal{R}$-class and $\mathcal{L}$-class of a.

Lemma 6. *The language* $L_{\mathcal{R}a} \stackrel{def}{=} \{u \ : \ \alpha(u) \,\mathcal{R}\, a\}$ *is star-free.*

Proof. Assume first (the interesting case), that a is not $\mathcal{R}$-equivalent to $1_{\mathbf{M}}$. In this case, the star-freeness of $L_{\mathcal{R}a}$ is established by the following equation:

$$L_{\mathcal{R}a} = L_{\mathcal{J}a} \cap \left(\bigcup_{bc \leq_{\mathcal{R}} a, \ b >_{\mathcal{J}} a} L_b C_c A^* \right) \ .$$

Clearly, if a word u belongs to the right-hand side of the equation, this means it has a prefix v belonging to $L_b C_c$. We derive $\alpha(u) \leq_{\mathcal{R}} \alpha(v) = bc \leq_{\mathcal{R}} a$. Since furthermore $\alpha(u) \,\mathcal{J}\, a$, one can use the following important lemma:

Lemma 7. *In a finite monoid,* $b \leq_{\mathcal{R}} a$ *and* $a \,\mathcal{J}\, b$ *implies* $a \,\mathcal{R}\, b$.

(Said differently, if $ab \,\mathcal{J}\, a$ then $ab \,\mathcal{R}\, a$.)

from which we immediately get that $\alpha(u) \,\mathcal{R}\, a$, i.e., $u \in L_{\mathcal{R}a}$.

Conversely, consider some $u \in L_{\mathcal{R}a}$. First of all, clearly $u \in L_{\mathcal{J}a}$. Let v be a minimal prefix of u such that $\alpha(v) \leq_{\mathcal{R}} a$. Since $1_{\mathbf{M}} \not\leq_{\mathcal{R}} a$ we have $\alpha(v) \neq 1_{\mathbf{M}}$, and hence $v \neq \varepsilon$. Thus we can write v as wx for some letter x. Setting $b = \alpha(w)$ and $c = \alpha(x)$, we have that u belongs to $L_b C_c A^*$. Furthermore $bc = \alpha(v) \geq_{\mathcal{R}} \alpha(u) \geq_{\mathcal{R}} a$. Finally, by minimality of v, $b = \alpha(w) \not\leq_{\mathcal{R}} a$. Since furthermore $b = \alpha(w) \geq_{\mathcal{R}} \alpha(u) \geq_{\mathcal{R}} a$, we obtain $b >_{\mathcal{R}} a$. This means by Lemma 7 that $b >_{\mathcal{J}} a$.

It remains the case $1_{\mathbf{M}} \in L_{\mathcal{R}a}$. We use the following lemma.

Lemma 8. *In a finite monoid* $\mathbf{M}$*, the* $\mathcal{J}$*-class of* $1_{\mathbf{M}}$ *coincides with its* $\mathcal{H}$*-class.*

Proof. Assume $1_{\mathbf{M}} \mathcal{J} a$. Since furthermore $a \leq_{\mathcal{R}} 1_{\mathbf{M}}$ (by $a = 1_{\mathbf{M}} a$), we have $a \mathcal{R} 1_{\mathbf{M}}$ using Lemma 7. Similarly, $a \mathcal{L} 1_{\mathbf{M}}$. Thus $a \mathcal{H} 1_{\mathbf{M}}$. □

Hence $L_{\mathcal{R}a} = L_{\mathcal{J}a}$ is star-free by Corollary 1. □

By symmetry, $L_{\mathcal{L}a} \stackrel{\text{def}}{=} \{u : \alpha(u) \mathcal{L} a\}$ is also star-free. From which we get.

Corollary 2. *The language* $L_{\mathcal{H}a} \stackrel{def}{=} \{u : \alpha(u) \mathcal{H} a\}$ *is star-free.*

Proof. Indeed $L_{\mathcal{H}a} = L_{\mathcal{R}a} \cap L_{\mathcal{L}a}$, and both $L_{\mathcal{L}a}$ and $L_{\mathcal{R}a}$ are star-free. □

Here the poof is concluded using the next lemma.

A monoid is called $\mathcal{H}$*-trivial* if all its $\mathcal{H}$-classes are singltons.

Lemma 9. *A monoid has only trivial subgroups if and only if it is* $\mathcal{H}$*-trivial.*

One direction is simple. Indeed, if you consider any subgroup of the monoid, then all its elements are $\mathcal{H}$-equivalent in the monoid. Thus $\mathcal{H}$-trivial implies that all subgroups are trivial. The converse requires more work.

Such monoids are also called *aperiodic*, which signifies that there exists some n such that $a^n = a^{n+1}$ for all $a \in \mathbf{M}$.

Hence, we deduce that $L_a = L_{\mathcal{H}a}$ which is star-free by Corollary 2. This completes the proof of the induction step, and hence the proof of the theorem of Schützenberger.

3 The Theorem of Factorization Forests of Simon

The theorem of forest factorizations is due to Simon [19]. It is at first glance, a purely algebraic result. It takes a finite monoid as input, as well as a morphism from words to this monoid, and shows the existence of factorizations with special properties for every word. However, this result has consequences which are purely automata related. Some of them are surveyed in [1]. A remarkable application is its use for proving the decidability of the limitedness problem for distance automata. Distance automata [6] are non-deterministic finite state automata over finite words, equipped with a set of special transitions. The function computed by such an automaton associates to each input word the minimal number of special transitions contained in some accepting run (∞ if there are no accepting runs). The limitedness problem consists in deciding whether the function computed by

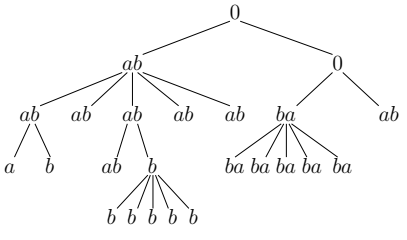
a distance automaton is bounded (in fact, over its domain, i.e., the set of words which are not given value ∞). This problem is decidable [6], and the optimal algorithm for it is due to Leung [8]. A simplified proof of validity of this theorem is due to Simon using the factorization forest theorem [20]. The theorem of factorization forests has other interesting applications. It is used for instance for characterizing the polynomial closure of classes of regular languages [15].

We fix from now and on a finite monoid $\mathbf{M}$.We will work with *sequences* of elements in the monoid. We denote them separated by commas for avoiding confusion with the product. Given a sequence $v = a_1, \ldots, a_n$, $\pi(v)$ denotes the value $a_1 a_2 \cdots a_n$.

A *factorization (tree)* (over the monoid $\mathbf{M}$) is a finite unranked ordered tree T whose leaves are labelled by elements of $\mathbf{M}$ (the label of node x is denoted $T(x)$) such that for every non-leaf node x of children $y_1, \ldots, y_k$ (read from left to right), $T(x) = \pi(T(y_1), \ldots, T(y_k))$. A *factorization of a sequence* $a_1, \ldots, a_n$ (of element in $\mathbf{M}$) is a factorization tree such that the labels of leaves read from left to right are $a_1, \ldots, a_n$. Traditionally, the height of a factorization is computed with leaves excluded, i.e., the height of a single leaf factorization is by convention 0.

A factorization is *Ramsey* if for all nodes x of rank[2] three or more, its children $y_1, \ldots, y_k$ are such that $T(y_1) = \cdots = T(y_n) = e$ where e is an idempotent (in particular, we also have $T(x) = e$).

Here is for instance an example of a Ramsey factorization of height 4, in the context of the syntactic monoid of our running example: the language of words "without two consecutive occurrences of letter a".



The theorem of factorization forests is then the following.

Theorem 3 (Simon [19]). *Every sequence of elements from a finite monoid M admits a Ramsey factorization of height at most $3|M| - 1$.*

We follow here the similar proofs from [4,7]. The original bound given by Simon is $9|M|$ instead of $3|M|$. The bound of $3|M|-1$ has been announced by Kufleitner, but is proved here. The proof completely follows the descriptions of the monoid in terms of the Green's relations.

A sequence $a_1, \ldots, a_n$ over M will be called *X-smooth* for some $X \subseteq \mathbf{M}$ if $a_i \cdots a_j \in X$ for all $1 \leq i \leq j \leq n$ (in particular each $a_i \in X$ and $a_1 \cdots a_n \in X$).

Lemma 10. *Let H be an $\mathcal{H}$-class of a finite monoid M, then every H-smooth sequence admits a Ramsey factorization of height at most $3|H| - 1$.*

Proof. We need a better understanding of the internal structure of $\mathcal{H}$-classes:

[2] The rank of a node is the number of its children.

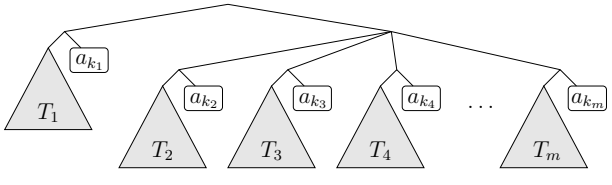
Lemma 11. *If an $\mathcal{H}$-class H contains an idempotent e, it is a group of neutral element e. Otherwise $ab <_{\mathcal{J}} a$ for all $a, b \in H$*

If H contains no idempotent element, then an H-smooth sequence has length at most 1. In this case, the single node factorization is Ramsey. It has height 1.

The interesting case is when H contains an idempotent e. Given an H-smooth sequence $u = a_1, \ldots, a_n$, call its *width* the value $|S(a_1, \ldots, a_n)|$ where the set $S(a_1, \ldots, a_n)$ abbreviates $\{a_1 \ldots a_l \ : \ 1 \leq l \leq n\}$ (remark that $S(a_1, \ldots, a_n)$ is empty iff $n = 0$). The construction is by induction on the width of the sequence. The base case is for $n = 0$. In this case, there exists a factorization of height at most 0 (this is a somewhat distorted case).

Let $v = a_1, \ldots, a_n$ for some $n > 0$. Define a to be $a_1 \cdots a_n$ and let a' be the inverse of a in the group H. Set $0 < k_1 < \cdots < k_m = n$ to be the list of indexes such that $a_1 \cdots a_{k_i} = a$ for all i. Let $v_1 = a_1, \ldots, a_{k_1 - 1}$, $v_2 = a_{k_1+1}, \ldots, a_{k_2-1}$, etc... Remark that $S(v_1) \subseteq S(v)$ and $a \notin S(v_1)$. Hence the width of v_1 is less than the one of v. One can apply the induction hypothesis, and get a Ramsey factorization T_1 for v_1. Similarly for all $i > 1$, $aS(v_i) \subseteq S(v)$, and $a \notin S(v_i)$. Furthermore $S(v_i) = a'aS(v_i)$, hence $|S(v_i)| \leq |aS(v_i)| < |S(v)|$. Thus, we can also apply the induction hypothesis, and get a Ramsey factorization T_i for v_i. Remark here that some of the v_i's may be empty. We do not pay attention to this harmless detail.

We now construct the following factorization:



In this construction, the empty trees are removed, and the labels of nodes are completed (in a unique way). We have to prove that the resulting factorization is indeed Ramsey. For this, it is sufficient to prove it for the only new node of rank possibly greater or equal to 3. Its i'th children has value $a_{k_i+1} \cdots a_{k_{i+1}}$. We compute:

$$\begin{aligned} a_{k_i+1}a_{k_2} &= a'aa_{k_i+1} \cdots a_{k_2} \\ &= a'a_1 \cdots a_{k_i}a_{k_i+1} \cdots a_{k_2} \\ &= a'a = e \ . \end{aligned}$$

Hence the new node is Ramsey.

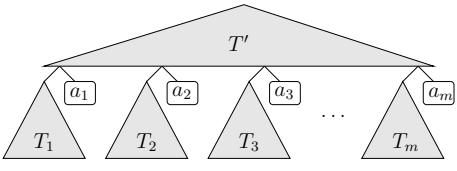
For the height, remark that for the width 1, all the T_i's are empty, and hence the construction can be simplified, and the resulting height is 2 (recall that leaves do not count in the height). At each induction step, the height of the factorization increases by at most 3. Hence, in the end, the factorization resulting from this construction has height at most $3|H| - 1$. □

Lemma 12. *For R an $\mathcal{R}$-class, every R-smooth sequence has a Ramsey factorization of height at most $3|R| - 1$.*

Proof. Let $v = a_1, \dots, a_n$ be the R-smooth sequence. The construction is by induction on the number of $\mathcal{H}$-classes occurring in v. If this number is 0, then one uses (once more) the distorted case of an empty tree.

Otherwise, let H be the $\mathcal{H}$-class of a_n. Let $1 \leq k_1 < \cdots < k_m \leq n$ be the indexes such that $a_{k_i} \in H$. One defines as for the previous lemma $v_1, \dots, v_{m+1}$ to be such that $v = v_1, a_1, v_2, \dots, a_m, v_{m+1}$. Remark first that the $\mathcal{H}$-class H does not appear in any of $v_1, \dots, v_{m+1}$. Thus, one can apply the induction hypothesis for each of the v_i's, and get a Ramsey factorization T_i. We also know that $\pi(v_i, a_i) = \pi(v_i)a_i$ and hence $\pi(v_i, a_i) \leq_{\mathcal{L}} a_i$. It follows from (the $\mathcal{L}$-version of) Lemma 7 that $\pi(v_i, a_i)\, \mathcal{L}\, a_i$, which means $\pi(v_i, a_i)\, \mathcal{L}\, a_i \in H$. Hence, we can apply Lemma 10, and get a Ramsey factorization T' for $\pi(v_1, a_1), \dots, \pi(v_m, a_m)$.

We now construct the following factorization:



It is Ramsey since each part it is composed of (namely $T_1, \dots, T_{m+1}$ and T') is Ramsey. One just needs to check that the values are consistent at the glue points. However, this is by construction.

Concerning its height. Once more in the pathological case of a single $\mathcal{H}$-class, the construction gets simpler, and its height is simply the height of T', which is at most $3|H| - 1$. Then at each step of the induction, the height increases of at most $3|H| - 1$ where H is the $\mathcal{H}$-class treated at this step of the induction. Overall we can over approximate it by $3|R| - 1$. □

In the above proof, we count the size of all the $\mathcal{H}$-classes separately. However, in some situations we would like to have more information. Such results exist:

Lemma 13 (Green's lemma). *Inside a $\mathcal{D}$-class,*

- *The $\mathcal{R}$-classes have all the same size (more precisely, if $ba\, \mathcal{D}\, a$, then the application which to x associates bx is a bijection from the $\mathcal{R}$-class of a onto the $\mathcal{R}$-class of ba).*
- *The $\mathcal{L}$-classes have all the same size, (more precisely, if $ab\, \mathcal{D}\, a$, then the application which to x associates xb is a bijection from the $\mathcal{L}$-class of a onto the $\mathcal{L}$-class of ab).*
- *All $\mathcal{H}$-classes have same size.*

Using the exact same proof, decomposing a $\mathcal{J}$-class into $\mathcal{R}$-classes, we obtain:

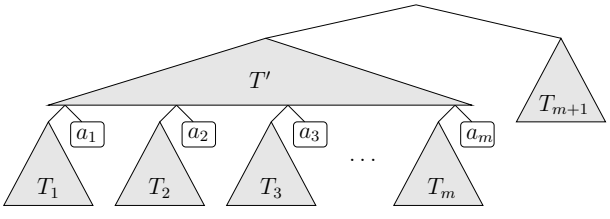
Lemma 14. *For J a $\mathcal{J}$-class, every J-smooth sequence has a Ramsey factorization of height at most $3|J| - 1$.*

We are now ready to prove the factorization forest theorem, Theorem 3.

Proof. Let $v = a_1, \ldots, a_n$ be a sequence over $\mathbf{M}$. This time, the proof is by induction on the $\mathcal{J}$-class J of $\pi(v)$. Assume one knows how to construct a Ramsey factorization for each sequence w such that $\pi(w) >_{\mathcal{J}} \pi(v)$.

Let $k_1 \geq 1$ be the least index such that $\pi(a_1, \ldots, a_{k_1}) \in J$. Continue by constructing $k_2 > k_1$ minimal such that $\pi(a_{k_1+1}, \ldots, a_{k_2}) \in J$, and so on, producing in the end $k_1 < \cdots < k_m$. One decomposes v as $v_1, a_{k_1}, v_2, \ldots, a_{k_m}, v_{m+1}$ as expected. By minimality property in the definition of the k_i's, $\pi(v_i) \notin J$. Since furthermore, $\pi(v_i) \geq_{\mathcal{J}} \pi(v) \in J$, we obtain $\pi(v_i) >_{\mathcal{J}} \pi(v)$. Thus we can apply the induction hypothesis, and get a Ramsey factorization T_i for v_i. Furthermore, for all $i \leq m$, $\pi(v_i, a_i) \in J$ by construction. Hence, we can apply Lemma 14 and get a Ramsey factorization T' for $\pi(v_1, a_1), \ldots, \pi(v_m, a_m)$.

We construct now the following factorization:



As in the previous lemmas, it is Ramsey simply because all its components are Ramsey.

Concerning the height. For a maximal $\mathcal{J}$-class J, the construction can be slightly simplified since all the T_i's are empty. Hence, the height is the one of T', which is at most $3|J| - 1$. Then, the height increases of the height of T' plus one at each step of the induction, which is at most $3|J|$ for a $\mathcal{J}$-class J. Overall, we reach a factorization of height at most $3|M| - 1$. □

An interesting point in this proof is that it is completely driven by the decomposition of the monoid according to Green's relations.

4 On ω-Semigroups and Monadic Logic

The remaining results which we consider involve the study of languages of infinite words, of length ω. We call such words *ω-words*. Regular languages of ω-words are usually defined using Büchi automata. We present in this section the corresponding algebraic notion, ω-semigroups. This is the extension of the notion of monoids (in fact semigroups) which is best suited for dealing with languages of infinite words.

ω-semigroups. An *ω-semigroup* is an algebra $\mathbf{S} = (S_+, S_\omega, \pi)$ consisting of two disjoint sets S_+ and S_ω and a product π mapping finite sequences in $(S_+)^+$

to S_+, finite sequences in $(S_+)^* S_\omega$ to S_ω, and infinite sequences in $(S_+)^\omega$ to S_ω. The product π is required to be associative, i.e., for all meaningful choices of sequences u_1, u_2,...,

$$\pi(\pi(u_1), \pi(u_2), \dots) = \pi(u_1, u_2, \dots) \ .$$

As an example, the *free ω-semigroup* generated by A consists of $S_+ = A^+$, $S_\omega = A^\omega$, and the product is simply the concatenation product for sequences (possibly infinite).

An example of a finite[3] ω-semigroup consists in $S_+ = \{a, b\}$ and $S_\omega = \{0, 1\}$. For all finite sequences u over $\{a, b\}$, $\pi(u) = a$ if a occurs in u, b otherwise. For all finite sequences u, $\pi(u, 0) = 0$ and $\pi(u, 1) = 1$, and for all ω-sequences w over $\{a, b\}$, $\pi(w) = 1$ if w contains infinitely many occurrences of a, $\pi(w) = 0$ otherwise.

A *morphism of ω semigroups* from $\mathbf{S}$ to $\mathbf{S}'$ is an application α mapping S_+ to S'_+ and S_ω to S'_ω which preserves the product, i.e., such that for all meaningful (possibly infinite) sequences $a_1, a_2, \dots$ from S,

$$\alpha(\pi(a_1, a_2, \dots)) = \pi'(\alpha(a_1), \alpha(a_2), \dots) \ .$$

A language of ω-words L is *recognizable* by an ω-semigroup $\mathbf{S}$ if there exists a morphism α from the free ω-semigroup to S, such that for every ω-word w, $w \in L$ if and only if $\alpha(w) \in F$, where $F \subseteq S_\omega$. One also says that L is *recognized by* $\mathbf{S}, \alpha, F$.

For instance, the language of infinite words over $\{a, b, c\}$ which contains infinitely many occurrences of letter a is recognized by the above finite ω-semigroup. The morphism α maps each non-empty finite word to a if it contains an occurrence of the letter a, to b otherwise. The morphism also sends each ω-word to 1 if it contains infinitely many occurrences of the letter a, to 0 otherwise. The finite subset of S_ω is $F = \{1\}$.

Given an ω-semigroup (S_+, S_ω, π), one defines the binary product $\cdot$ over S_+ by $ab \stackrel{\text{def}}{=} \pi(a, b)$. One also uses it from $S_+ \times S_\omega$ to S_ω (with the same definition). The exponentiation by ω from S_+ to S_ω is defined with $a^\omega \stackrel{\text{def}}{=} \pi(a, a, a, \dots)$. It turns out that if S_+ and S_ω are finite, then π is entirely determined by the product $\cdot$ and the exponent by ω [22]. We will not use this direction, however, this explains why it is sufficient to know a finite amount of information for working effectively with ω-semigroups.

The relationship with the monoids we have been using so far is that $(S_+, \cdot)$ is a semigroup: a *semigroup* $\mathbf{S} = (S, \cdot)$ is a set S together with an associative operator $\cdot$. Hence, this is simply a monoid without necessary a neutral element. This difference is not essential. In our case, it simply reflects the special property of the empty word (which is the neutral element of the free monoid) in the study of infinite words: it is the only finite word which, when iterated ω times, does not

[3] Finite means that both S_+ and S_ω are finite, though *a priori*, the product π requires an infinite quantity of information for being described.

yield an infinite word. Using semigroups rather than monoids means avoiding to treat this particular case.

The structure of semigroups and monoids are highly related. Given a semigroup $\mathbf{S}$, one defines $\mathbf{S}^1$ to be $\mathbf{S}$ to which has been added a new neutral element 1, if necessary. This makes a monoid out of a semigroup. When we refer to the Green's relations of the semigroup, we refer in fact implicitly to the Green's relations of the corresponding monoid $\mathbf{S}^1$.

Monadic second-order logic. Let us recall here that *monadic (second-order) logic* is the first-order logic extended with the ability to quantify over sets. I.e., it is possible to quantify existentially or universally over elements (e.g., $\exists x, \forall y, \dots$) and sets of elements (e.g., $\exists X, \forall Y, \dots$), to test membership (e.g., $x \in Y$), to use boolean connectives (i.e., $\vee$,$\wedge$, $\neg$) and to use the predicates of the structure. In our case, we consider ω-words. In this case, the elements are the positions in the word, i.e., non-negative integers, and there are two kinds of predicates. For all letters a, the predicate $a(x)$ allows to test whether the letter at the position x is an a, and the predicate $x \leq y$ tests whether the position x occurs to the left of or at y. Given an ω-word, one says that its *monadic theory is decidable* if there is an algorithm which, given any sentence of monadic logic, decides whether it holds or not over the ω-word. See for instance [21] for more on the subject.

Those various notions are tied together by the following theorem.

Theorem 4 ([2],[12]). *A language of ω-words is regular (i.e., definable by Büchi automata) if and only if it is recognized by a finite ω-semigroup, if and only if it is definable in monadic logic. Furthermore, the translations are effective.*

For this reason, we do not use explicitly monadic logic from now on.

5 The Characterization of Decidable Theories by Semenov

The result of Semenov we are mentioning answers the following question:

When is the monadic theory of an ω-word decidable?

This question differs, though it is related, to the original question solved by Büchi [2], which aims at deciding whether a monadic sentence has a model, i.e., decide if it is satisfied by some ω-word.

A possible answer to the above question is:

Theorem 5 (variation of Semenov[18]). *An ω-word w has a decidable monadic theory if and only if the following questions are decidable for all regular languages of finite words L:*

(A) Does there exist a finite prefix of w in L? I.e., $w \in LA^\omega$?
(B) Does there exist recurrent factors of L in w ? I.e., $w \in (A^*L)^\omega$?

One direction is straightforward. Indeed, it is easy to translate the properties "$w \in LA^\omega$" and "$w \in (A^*L)^\omega$" into equivalent monadic formulas. Hence properties (A) and (B) can be reduced to the decidability of the monadic theory of w.

The interesting implication is the opposite one. We will use as our major tool Lemma 18 below. This requires beforehand to disclose some extra facts concerning the Green's relations.

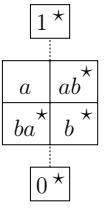
An element a in a monoid is called *regular* if there exists s such that $asa = a$.

Lemma 15. *Let J be a $\mathcal{J}$-class J in a finite monoid. The following items are equivalent:*

- *J contains a regular element,*
- *J contains an idempotent,*
- *every element in J is regular,*
- *every $\mathcal{R}$-class in J contains an idempotent,*
- *every $\mathcal{L}$-class in J contains an idempotent,*
- *there exist two elements in J, the product of which belongs to J.*

Such $\mathcal{J}$-classes are naturally called *regular*.

Keeping the same example as above, one enriches the presentation by adding information concerning the idempotents. Each $\mathcal{H}$-class is now decorated by a star if it contains an idempotent.



This gives an important information:

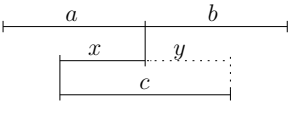
Lemma 16. *In a finite semigroup, if $a \mathcal{J} b$, then $ab \mathcal{J} a$ if and only if there exists an idempotent e such that $e \mathcal{R} b$ and $e \mathcal{L} a$, and in this case, $ab \mathcal{R} a$ and $ab \mathcal{L} b$.*

In our example, on the one hand, *abba* stays in the same $\mathcal{J}$-class since b is an idempotent, and the result is a. On the other hand, *baab* falls in the lower $\mathcal{J}$-class since a is not an idempotent.

The following technical lemma contains the key arguments we need:

Lemma 17. *If $a \mathcal{J} b \mathcal{J} ab \mathcal{J} c$ and $a \mathcal{L} x \mathcal{R} c$, then there exists y such that $c = xy$ and $c \mathcal{L} y \mathcal{R} b$.*

This statement can be depicted as shown. I.e., x can be completed to the left into a ($a \mathcal{L} x$), and to the right into c ($x \mathcal{R} c$). Then it is possible to complete x to the right into $c = xy$ such that y can be completed into b ($b \mathcal{R} y$).



Proof. We have $x \mathcal{L} a$, hence $ab \mathcal{L} xb$ by Lemma 3. Thus $xb \mathcal{J} ab \mathcal{J} x$. It follows $xb \mathcal{R} x \mathcal{R} c$, once more by Lemma 3. Hence $c = xbz$ for some z. Let $y = bz$. Clearly $c = xbz = xy$. Furthermore $y = bz \leq_{\mathcal{R}} b$, and $bz \geq_{\mathcal{L}} xbz = c$. Hence by (twice) Lemma 7, $c \mathcal{L} y \mathcal{R} b$. □

We then obtain by iterated applications of Lemma 17.

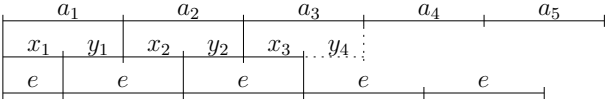
Lemma 18. *If $u = a_1, a_2, \ldots \in (S_+)^\omega$ is $\mathcal{J}$-smooth (i.e., is J-smooth for some $\mathcal{J}$-class J) then (a) there exists an idempotent e such that $e \,\mathcal{R}\, a_1$, and (b) $\pi(u) = e^\omega$ for all idempotents e such that $e \,\mathcal{R}\, a_1$.*

Proof. Let J be the $\mathcal{J}$-class of a_1. Since a_1, a_2, and a_1a_2 all belong to J, it follows that J is regular by Lemma 15. Hence, still by Lemma 15, there exists an idempotent e in the $\mathcal{R}$-class of a_1 (a).

Let e be such an idempotent. One constructs inductively the elements $x_n, y_n \in S_+$ such that for all positive integer n:

(1) $e \,\mathcal{L}\, x_n \,\mathcal{R}\, a_n$, and if $n = 1$, $x_1 = e$, otherwise $y_{n-1}x_n = e$,
(2) $a_n \,\mathcal{L}\, y_n \,\mathcal{R}\, e$, and $x_n y_n = a_n$.

The constructions can be illustrated as follows:



For $n = 1$, one chooses $x_1 = e$, and we know by choice of e that $e \,\mathcal{L}\, x_1 \,\mathcal{R}\, a_1$. Hence (a) holds for $n = 1$. Then for all n, assuming (1) establishes (2) using simply Lemma 17. Similarly, assuming (2), one establishes (1) for $n+1$ using again Lemma 17.

It is then easy to conclude. Indeed, using the associativity of π, we have $\pi(a_1, a_2, \ldots) = \pi(x_1y_1, x_2y_2, \ldots) = \pi(x_1, y_1, x_2, y_2, \ldots) = \pi(x_1, y_1x_2, y_2x_3, \ldots) = e^\omega$. Property (b) holds. □

Lemma 18 provides us a lot of information concerning the monadic theory of an ω-word. Given a $\mathcal{J}$-class J and a word w, we say that J *occurs in* w if $w \in A^*L_JA^\omega$ where $L_J = \{u \in A^+ \,:\, \alpha(u) \in J\}$. We say that J *is recurrent in* w if $w \in (A^*L_J)^\omega$.

Lemma 19. *For all w, there is a minimum (for the $\mathcal{J}$-pre-order) $\mathcal{J}$-class recurrent in w.*

Proof. Assume that both J and J' are recurrent, we shall prove that there is a $\mathcal{J}$-class below or equal to both J and J' which is recurrent. Since both J and J' are recurrent, w can be written as $u_1v_1u'_1v'_1u_2u'_2\ldots$, where $v_i \in L_J$ and $v'_i \in L_{J'}$ for all i. Let J_i be the $\mathcal{J}$-class of $\alpha(v_iu'_iv'_i)$. By definition of the $\mathcal{J}$-pre-order, $J_i \leq_\mathcal{J} J$ and $J_i \leq_\mathcal{J} J'$. Furthermore, since there are only finitely many $\mathcal{J}$-class, one of the J_i's has to occur infinitely often, which means it is recurrent in w. □

Lemma 20. *If J is recurrent in w, and no $\mathcal{J}$-class $J' \not\geq_\mathcal{J} J$ occurs in w, then w can be decomposed into $v_1v_2\ldots$ such that $\alpha(v_1), \alpha(v_2), \ldots$ is J-smooth.*

Proof. Find the first non-empty prefix v_1 of w such that $\alpha(v_1) \in J$. This is possible since J is recurrent. Then proceeds with the remaining suffix, and construct $v_2, \dots$ For the sake of contradiction, assume $\alpha(v_1), \alpha(v_2), \dots$ is not $\mathcal{J}$-smooth, this would mean that $\alpha(v_i \dots v_j) \notin J$ for some $i \leq j$. However, we have $\alpha(v_i \dots v_j) \leq_{\mathcal{J}} \alpha(v_i)\, \mathcal{J}\, J$, thus this means that $\alpha(v_i \dots v_j) <_{\mathcal{J}} J$. A contradiction since by hypothesis no $\mathcal{J}$-class below J does occur in w. □

Corollary 3. *Assume the minimum recurrent $\mathcal{J}$-class of w is J and all $\mathcal{J}$-classes occurring in w are above or equal to J, then:*

- *w has a finite prefix u such that $\alpha(u) \in J$,*
- *for all prefix u of w such that $\alpha(u) \in J$,*

$$\alpha(w) = \alpha(u)^{\rightarrow} ,$$

where $a^{\rightarrow} \stackrel{def}{=} e^{\omega}$ for some/all idempotents e such that $e\, \mathcal{R}\, a$ (if defined).

Proof. By hypothesis and Lemma 20, w can be written as $w = v_1 v_2 \dots$ such that $\alpha(v_1), \alpha(v_2), \dots$ is $\mathcal{J}$-smooth. Thus by Lemma 18, $\alpha(w) = \alpha(v_1)^{\rightarrow}$.

Furthermore, u is either a prefix or a suffix of v_1, yielding $\alpha(u) \geq_{\mathcal{J}} \alpha(v_1)$ or $\alpha(u) \leq_{\mathcal{J}} \alpha(v_1)$ respectively. In any case, since $\alpha(u)\, \mathcal{J}\, \alpha(v_1)$, we have $\alpha(u)\, \mathcal{R}\, \alpha(v_1)$ by Lemma 7. Hence $e\, \mathcal{R}\, \alpha(v_1)$ if and only if $e\, \mathcal{R}\, \alpha(u)$. This means $\alpha(w) = \alpha(v_1)^{\rightarrow} = \alpha(u)^{\rightarrow}$. □

We are now ready to establish Theorem 5.

Proof. Assume that properties (A) and (B) hold for an ω-word w, and that one is given a monadic sentence φ. This sentence φ defines a regular language of ω-words which is recognized by an ω-semigroup $\mathbf{S}$ by Theorem 4.

Using (A), we can decide what is the minimum recurrent $\mathcal{J}$-class in w (it exists by Lemma 19). Call it J. The next step consists in finding a decomposition of w in uw' such that all $\mathcal{J}$-class occurring in w' are above or equal to J. Such a word u exists. For finding it, one just tries all possible u's, and stop when both $w \in uA^{\omega}$, and $w \notin uA^* L_{\not\geq_{\mathcal{J}} J} A^{\omega}$, where $L_{\not\geq_{\mathcal{J}} J}$ is the set of non-empty words which are not mapped by α to J or above. This is obviously doable using iterated applications of item (A). Then one finds v such that uv is a prefix of w, and $\alpha(v) \in J$. It is sufficient once more to test all possible such v's using (A). Then, we have $\alpha(w) = \alpha(u)\alpha(v)^{\rightarrow}$ by Corollary 3. Hence, we can decide if φ holds or not. □

6 Deterministic Automata over ω-Words: McNaughton

A *Büchi automaton* is a tuple (Q, A, I, Δ, B) where Q is a finite set of *states*, A is the alphabet, $I \subseteq Q$ is a set of *initial states*, $\Delta \subseteq Q \times A \times Q$ is the *transition relation*, and $B \subseteq \Delta$ is the set of *Büchi transitions*. An automaton is deterministic if Δ is a function from $Q \times A$ to Q. A *run of the automaton over an ω-word w* is defined as usual as an infinite sequence of transitions such that

the first state is initial, the letters in the transitions yield w, and consecutive transitions agree on the common states. A run is *accepting* if it contains infinitely many Büchi transitions. The *language accepted* by an automaton $\mathcal{A}$ is the set of ω-words over which there is an accepting run of the automaton. A language is said *regular* if it is accepted by some Büchi automaton. A language is *deterministic Büchi* if it is accepted by a deterministic Büchi automaton.

It is known that not all regular languages are deterministic Büchi. However McNaughton's result still gives a strong relationship:

Theorem 6. *A language of ω-words is regular if and only if it is a Boolean combination of deterministic Büchi languages.*

Usually, this theorem is stated as the existence of deterministic automata belonging to more general classes of automata (such as parity/Rabin/Streett/Müller automata). In fact, with just a slightly more involved construction, one can immediately get a deterministic parity automaton along the lines presented here. We choose here the simplest presentation. Standard constructions are directly performed on the automaton, here we translate an ω-semigroup directly into a Boolean combination of deterministic Büchi automata. The first direct translation of ω-semigroup to deterministic automata is due to Carton [3].

Once more, we start from a regular language L which is presented by an ω-semigroup (S_+, S_ω, π), a morphism α, and some subset $F \subseteq S_\omega$.

Lemma 21. *Given a $\mathcal{J}$-class J, the language of words such that the minimum recurrent $\mathcal{J}$-class J' is such that $J' \not> J$ is deterministic Büchi.*

Proof. Without loss of generality, one assumes $1 \notin J$ (otherwise, this is the language A^ω). Let K be the set of elements $\{a \in S_+^1 \ : \ a >_\mathcal{J} J\}$. By the assumption $1 \notin J$, we have $1 \in K$. One constructs the following deterministic Büchi automaton:

- The set of states is K.
- The initial state is 1.
- The transition from state a, reading letter x, ends in state

$$\Delta(a,x) = \begin{cases} a\alpha(x) \text{ if } a\alpha(x) \in K, & (a) \\ 1 \qquad \text{otherwise.} & (b) \end{cases}$$

- The automaton accepts if some transition of kind (b) is seen infinitely often.

This automaton decomposes an input ω-word into $w = u_1 u_2 \dots$ in such a way that each u_i is minimal such that $\alpha(u_i) \not> J$. It accepts if and only if this decomposition is infinite.

Assume the automaton accepts an ω-word w. Then there is a $\mathcal{J}$-class visited by infinitely many u_i's, which is a witness that the minimum recurrent $\mathcal{J}$-class is not above J. Conversely, assume the automaton does not accept an ω-word w. This means that after some time no more transitions of kind (b) are visited anymore. Thus all $\mathcal{J}$-classes appearing after this moment are above J. □

Lemma 22. *Given a $\mathcal{J}$-class J, the language of ω-words:*

$$L \cap \{w \in A^\omega \ : \ J \text{ is the minimum recurrent } \mathcal{J}\text{-class in } w\}$$

is the difference of two deterministic Büchi languages.

Proof. We construct a deterministic Büchi automaton such that:

A. It accepts all ω-words such that the minimum recurrent $\mathcal{J}$ class is $\not\geq_{\mathcal{J}} J$.
B. An ω-word such that the minimum recurrent $\mathcal{J}$-class is J is accepted if and only if it does not belong to L.

Then, it is easy to see that if we subtract this language to the one of Lemma 21, we obtain the expected language.

Let K be the set of elements $\{a \in S^1_+ \ : \ a \geq_{\mathcal{J}} J\}$. One constructs the following deterministic Büchi automaton:

- The set of states is $S^1_+ \times K$.
- The initial state is $(1, 1)$.
- The transition from state (a, b) while reading letter x goes to state:

$$\Delta((a,b),x) = \begin{cases} (a, b\alpha(x)) & \text{if } b\alpha(x) \in K \quad (a) \\ (ab\alpha(x), 1) & \text{otherwise.} \quad (b) \end{cases}$$

  Transition (a) is called $(a1)$ if $a(b\alpha(x))^\to \in F$, otherwise, it is called $(a2)$.
- The automaton accepts if a transition of the kind (b) or $(a2)$ is visited infinitely often.

First of all, remark that, for the same reasons as in the proof of Lemma 21, the transitions of kind (b) are visited infinitely often if and only if the minimum recurrent $\mathcal{J}$-class is not above or equal to $\mathcal{J}$. This settles item A.

Consider now some ω-word such that the minimum recurrent $\mathcal{J}$-class is J. This means that after some steps, no more (b)-transitions will be encountered. This uniquely decomposes the ω-word into $w = uv$ in which u is the prefix of w which terminates when the last (b)-transition is visited (possibly $u = \varepsilon$ as a pathological case if no (b)-transition is ever encountered).

One easily sees that for all finite prefix of w of the form uv', the automaton reaches the state $(\alpha(u), \alpha(v'))$ after reading it. Since the minimum recurrent $\mathcal{J}$-class is J, $\alpha(v')$ will eventually enter $\mathcal{J}$. By Corollary 3, if $w \in L$, then all transitions from this moment are of kind (a1) and the word is rejected, while if $w \notin L$, all transitions from this moment are of kind (a2), and the word is accepted. This settles item B. □

Of course, Theorem 6 follows directly since L is the union of the languages of Lemma 22 for J ranging over the possible $\mathcal{J}$-classes.

Acknowledgments

I am grateful to Olivier Carton, Denis Kuperberg and Jean-Éric Pin for helping me in the preparation of this document.

References

1. Bojańczyk, M.: Factorization forests. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 1–17. Springer, Heidelberg (2009)
2. Bühi, J.R.: On a decision method in restricted second order arithmetic. In: Proceedings of the International Congress on Logic, Methodology and Philosophy of Science, pp. 1–11. Stanford Univ. Press, Stanford (1962)
3. Carton, O.: Mots infinis, ω-semigroupes et topologie. PhD thesis, University Paris VII (1993)
4. Colcombet, T.: Factorisation forests for infinite words and applications to countable scattered linear orderings. Theoretical Computer Science 411, 751–764 (2010)
5. Grillet, P.A.: Semigroups. An introduction to the structure theory. In: Pure and Applied Mathematics, vol. 193, ix, p. 398. Marcel Dekker, Inc., New York (1995)
6. Hashiguchi, K.: Limitedness theorem on finite automata with distance functions. J. Comput. Syst. Sci. 24(2), 233–244 (1982)
7. Kufleitner, M.: The height of factorization forests. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 443–454. Springer, Heidelberg (2008)
8. Leung, H.: Limitedness theorem on finite automata with distance functions: An algebraic proof. Theoretical Computer Science 81(1), 137–145 (1991)
9. McNaughton, R.: Testing and generating infinite sequences by a finite automaton. Information and Control 9, 521–530 (1966)
10. McNaughton, R., Papert, S.: Counter-free Automata. MIT Press, Cambridge (1971)
11. Perrin, D.: Finite automata. In: Handbook of Theoretical Computer Science, vol. B, pp. 1–57. Elsevier, Amsterdam (1990)
12. Perrin, D., Pin, J.E.: Semigroups and automata on infinite words. In: Fountain, J. (ed.) NATO Advanced Study Institute Semigroups, Formal Languages and Groups, pp. 49–72. Kluwer Academic Publishers, Dordrecht (1995)
13. Pin, J.E.: Varieties of Formal Languages. North Oxford Academic, London (1986)
14. Pin, J.E.: Syntactic semigroups. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 679–746. Springer, Heidelberg (1997)
15. Pin, J.E., Weil, P.: Polynomial closure and unambiguous product. Theory Comput. Syst. 30(4), 383–422 (1997)
16. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. and Develop. 3, 114–125 (1959)
17. Schützenberger, M.P.: On finite monoids having only trivial subgroups. Information and Control 8, 190–194 (1965)
18. Semenov, A.L.: Decidability of monadic theories. In: Chytil, M.P., Koubek, V. (eds.) MFCS 1984. LNCS, vol. 176, pp. 162–175. Springer, Heidelberg (1984)
19. Simon, I.: Factorization forests of finite height. Theoretical Computer Science 72, 65–94 (1990)
20. Simon, I.: On semigroups of matrices over the tropical semiring. RAIRO ITA 28(3-4), 277–294 (1994)
21. Thomas, W.: Languages, automata and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Language Theory, vol. 3, pp. 389–455. Springer, Heidelberg (1997)
22. Wilke, T.: An Eilenberg theorem for ∞-languages. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) ICALP 1991. LNCS, vol. 510, pp. 588–599. Springer, Heidelberg (1991)

Automatic Structures and Groups

Bakhadyr Khoussainov

Department of Computer Science, University of Auckland, New Zealand
bmk@cs.auckland.ac.nz

1 Introduction

Automata theory has unified many areas of computer science and mathematics. These include group theory (Thurston automatic groups [11], branch and self similar groups [1] [24]), computable model theory (the theory of automatic structures [5] [17] [16]), finite model theory, algorithms and decidability, logic, model checking and verification.

In this tutorial we emphasize the use of automata in representation of infinite mathematical structures and concentrate on two topics. One topic is automatic structures, and the other, representations of groups by automata. There are already many papers motivating the study of automatic structures and surveying some of the results in the area (e.g. [28] [16] [18]). The book [11] is a standard reference for automatic groups in the sense of Thurston. See also a survey by S. Gersten on automatic groups and their relations with hyperbolic groups [12]. In this paper we present definitions and theorems in the area of automatic structures, automatic groups, outline some recent results, and discuss possible topics for research.

We note that there are several models of automata: finite automata, tree automata, various types of ω-automata, and ω-tree automata. Here we restrict ourselves to finite automata and tree automata. Finite automata are designed to process finite strings over a finite alphabet, while tree automata are designed to process finite labeled trees. We assume that the reader is familiar with these automata and their basic properties. For completeness we will, however, provide definitions for these type of machines in the next section.

A brief outline of this paper is as follows. The next two sections contain basic definitions, and a proof that each automatic structure posses a decidable first order theory and is closed under definability. Section 4 stresses the domain dependency of automatic structures. The main idea is that algebraic (and even algorithmic) properties of automatic structures depend heavily on the underlying domains. This is formalized in the definition of the algebraic spectra of automata recognizable sets. Given a (finite, or tree) automaton recognizable language X, we define *the algebraic spectrum of* X to be the class of automatic structures whose domain is X. This definition calls for a refined analysis of automaticity for structures. As an example, we prove that every infinite automatic scattered linear order is isomorphic to an automatic linear order over the domain of all finite binary strings. In contrast, no scattered tree-automatic linear order exists over the domain of all finite trees. As a corollary, no tree-automatic well order exists on the set of all finite trees.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 22–40, 2011.

Sections 5 and 6 study automaticity in groups. There are several ways to represent groups by finite automata. The first is to consider finite automata with letter-by-letter outputs, known as Mealy automata. Every such automaton determines length preserving functions on the set of strings (over the alphabet of the automaton). The number of these functions is bounded by the number of states of the automaton. If these functions are permutations, one can consider groups generated by them under the composition operation. These determine groups called *automata groups*. Examples of such groups include the famous Grigorchuk groups. The reader is referred [2] for background. The second way is to consider groups as defined by Thurston and his collaborators [11]. Roughly, a group G generated by a finite set X is Thurston-automatic if there exists a regular subset L of $X^\star$ such that the natural mapping $u \to \bar{u}$ from L into G is bijective, and the left-multiplication by each of the generators can be performed by finite automata. Many natural examples of Thurston-automatic groups arise in topology and geometry. For instance, all hyperbolic groups are Thurston-automatic. So are virtually abelian groups. We propose a natural generalization of Thurston automaticity for groups: *Cayley automatic groups*. Roughly, a finitely generated group G is *Cayley automatic* if there is a coding of the vertices of a Cayley graph of G under which both the set of codes and multiplication by each of the generators are finite automata recognizable. Every group, automatic in the sense of Thurston, is Cayley automatic. However, there are many examples of Cayley automatic groups that are not Thurston automatic. These include Heisenberg groups and groups of nilpotency class at most 2. We show that the class of Cayley automatic groups is closed under many group-theoretic constructions.

The last section discusses the isomorphism problem for automatic structures. We outline several known results in the study of the isomorphism problem. The emphasis is on to showing that the isomorphism problem lies in various spectrum of decidability and undecidability. This spectrum depends on the classes of structures (e.g. classes of linearly ordered sets, Boolean algebras, graphs). We give examples of automatic structures for which the isomorphism problem is is (highly) undecidable, and examples of automatic structures for which the isomorphism problem is decidable.

2 Basics

A finite alphabet is denoted by Σ. As always, $\Sigma^\star$ denotes the set of all finite words over Σ. We define finite automata as follows.

Definition 1. *A* finite automaton *is a tuple* $\mathcal{M} = (S, \iota, \Delta, F)$*, where* S *is the set of* states *and* $\iota \in S$ *is the* initial state*,* $\Delta \subseteq S \times \Sigma \times S$ *is the* transition table*, and* $F \subseteq S$ *is the set of* final states*. The set* S *of states is always finite.*

Note that given an automaton $\mathcal{M}$, its transition table Δ determines the alphabet Σ. A *run* of the automaton $\mathcal{M}$ on word $w = \sigma_1\sigma_2\ldots\sigma_n \in \Sigma^\star$ is a sequence of states $q_0, q_1, \ldots, q_n$ such that $q_0 = \iota$ and $(q_i, \sigma_{i+1}, q_{i+1}) \in \Delta$ for all

$i \in \{0, 1, \ldots, n-1\}$. If $q_n \in F$, for some run of $\mathcal{M}$ on w, then the automaton $\mathcal{M}$ *accepts* w. The *language* of the automaton $\mathcal{M}$ is

$$L(\mathcal{M}) = \{w \mid w \text{ is accepted by } \mathcal{M}\}.$$

We call such languages *finite automaton (FA) recognisable* or *regular* languages.

We now need to define tree automata. *A Σ–tree* is a mapping $t : dom(t) \to \Sigma$ such that the domain $dom(t)$ of t is a finite subtree of the binary tree $\{0,1\}^\star$ with the property that every non-leaf node $v \in dom(t)$ has both its children $v0$ and $v1$ in $dom(t)$. The symbol λ denotes the root of $dom(t)$. The *boundary* of the domain $dom(t)$ is the set

$$\partial dom(t) = \{xb \mid x \text{ is a leaf of } dom(t) \text{ and } b = 0 \text{ or } b = 1\}.$$

The set of all Σ-trees is denoted by $T(\Sigma)$.

Definition 2. *A* tree automaton *is a tuple $\mathcal{M} = (S, \iota, \Delta, F)$, where S is the set of* states *and $\iota \in S$ is the* initial state*, $\Delta \subseteq S \times \Sigma \times (S \times S)$ is the* transition table*, and $F \subseteq S$ is the set of* final states*. As for finite automata S is always a finite set.*

Next we define tree automata recognizable languages. Let $\mathcal{M}$ be a tree automaton and t be a Σ-tree. A *run* of $\mathcal{M}$ on the tree t is a mapping $r : dom(t) \cup \partial dom(t) \to S$ such that $r(\lambda) = \iota$ and for all $x \in dom(t)$ if $t(x) = \sigma$ and $r(x) = s$ then $r(x0) = s_0$ and $r(x1) = s_1$ where $(s, \sigma, (s_0, s_1)) \in \Delta$. If for every leaf $x \in dom(t)$ we have both $r(x0) \in F$ and $r(x1) \in F$, then the run r is said to be accepting. Automaton $\mathcal{M}$ *accepts* the tree t, if there is a run of $\mathcal{M}$ on t which is accepting. The *language* of the tree automaton $\mathcal{M}$ is

$$L(\mathcal{M}) = \{t \in T(\Sigma) \mid t \text{ is accepted by } \mathcal{M}\}.$$

These languages are called *tree-automata recognisable* or equivalently *regular*.

The word and tree languages are subsets of the underlying domains $\Sigma^\star$ and $T(\Sigma)$, respectively. As such they are unary relations on these domains. Our goal now is to define automata recognizable relations on these domains. For this, we need one technical notation.

Let $t_0, \ldots, t_{n-1}$ be Σ-trees. For $x \in dom(t_0) \cup \ldots \cup dom(t_{n-1})$ and $i < n$, we set $t_i'(x) = t_i(x)$ if $x \in dom(t_i)$, and $t_i'(x) = \Box$ if $x \notin dom(t_i)$. *The convolution* of the trees $t_0, \ldots, t_{n-1}$ is then the tree given by a mapping $conv(t_0, \ldots, t_{n-1})$ from $dom(t_0) \cup \ldots \cup dom(t_{n-1})$ to $(\Sigma \cup \{\Box\})^n$ which satisfies for all $x \in dom(t_0) \cup \ldots \cup dom(t_{n-1})$ that

$$conv(t_0, \ldots, t_{n-1})(x) = (t_0'(x), \ldots, t_{n-1}'(x)).$$

Thus a convolution of an n-tuple of trees $t_0, \ldots, t_{n-1}$ is a tree over a larger finite alphabet. We often identify the convoluted tree $conv(t_0, \ldots, t_{n-1})$ with the tuple $(t_0, \ldots, t_{n-1})$.

Definition 3. *We say that an n-ary relation R on $T(\Sigma)$ is* tree-automatic *if its convolution $conv(R) = \{conv(t_0, \ldots, t_{n-1}) \mid (t_0, \ldots, t_{n-1}) \in R\}$ is a tree-automata recognisable language.*

The reader can easily modify the convolution operation for finite strings, and hence define finite automata recognisable relations on the set $\Sigma^\star$. This allows us to give the following definition central to this paper.

Definition 4. *A structure $\mathcal{A} = (A; R_1, \ldots, R_n)$ is* tree-automatic (word-automatic) *if domain A and the atomic relations R_1, ..., R_n are all tree-automata (finite automata) recognisable. For a structure $\mathcal{B}$, if $\mathcal{B}$ is isomorphic to the structure $\mathcal{A}$ then we say that $\mathcal{A}$ is a* tree-automatic (word-automatic) presentation *of $\mathcal{B}$. We often refer to tree and word automatic structures and presentations as automatic structures and presentations, respectively.*

Automatic presentations $\mathcal{A}$ of a structure $\mathcal{B}$ is thus a finite collections of automata for the domain and atomic relations of $\mathcal{A}$. For the reader familiar with the second order logic, we note that the definition of automata presentability is a Σ^1_1-definition in arithmetic. This is because automata presentability of $\mathcal{B}$ requires a search for an isomorphism from automatic structures $\mathcal{A}$ to $\mathcal{B}$. However, we often abuse our definition and refer to automata presentable structures as automatic structures. We also remark that the type of automaticity used (tree automata or finite automata) in the text will be clear from the content.

Examples of word-automatic structures are the following. (1) The structure $(1^\star; S, \leq))$, where $S(1^n) = 1^{n+1}$ and $1^n \leq 1^m$ iff $n \leq m$ for $n, m \in \mathbb{Z}$. (2) The structure $(\{0,1\}^\star; \leq_{lex}, \leq_{pref}, \leq_{llex})$, where the relations are the lexicographical, prefix, and length-lecocographical orders on strings. (3) the small ordinals ω^n, where $n \in \mathbb{N}$. (4) The structure $(Base_k; Add_k)$, where $Base_k = \{0, 1, \ldots, k-1\}^\star \cdot \{1, \ldots, k-1\}$. In this example each word $w = x_0 \ldots x_n \in Base_k$ is identified with the number

$$val_k(w) = \sum_{i=0}^{n} x_i k^i.$$

This gives the least significant digit first (LSDF) base k representation of natural numbers. The predicate Add_k is the graph of the k-base addition of natural numbers, that is $Add_k = \{(u, v, w) \mid val_k(u) + val_k(v) = val_k(w)\}$. This structure is isomorphic to the natural numbers with addition $(\mathbb{N}, +)$ known as Presburger arithmetic.

Examples of tree-automatic structures are: (1) all word-automatic structures; (2) Skolem arithmetic $(\mathbb{N}; \times)$, (3) term algebras with infinitely many generators, (4) the ordinal ω^ω, (5) the countable atomless Boolean algebra. The structures in Examples (2)-(5) are not word-automatic structures. These require proofs, some non-trivial, see for instance [4] [19] [20].

3 Decidability and Definability Theorem

Closure properties for both finite and tree automata imply that automatic structures are closed under first order interpretations. Furthermore, the decidability

of the emptiness problem for automata can be used to prove that the first order theory of every automatic structure is decidable. We state an extended version of this fact in the theorem below, and outline a proof.

Theorem 5 ([5,14,17]). *Let $FO + \exists^\omega$–logic be the extension of the first-order logic with the "there are infinitely many" quantifier $\exists^\omega$.*

1. *There is an algorithm that, given an automatic presentation of $\mathcal{A}$ and a formula $\phi(x_1, \ldots, x_n)$ in $FO + \exists^\omega$–logic, builds an automaton $\mathcal{M}_\phi$ that recognizes all n-tuples in $\mathcal{A}$ that satisfy the formula.*
2. *There is an algorithm, that given an automatic presentation of $\mathcal{A}$ and a sentence ϕ in $FO + \exists^\omega$–logic, decides if ϕ is true in $\mathcal{A}$. Hence, the first-order theory of any automatic structure is decidable.*

Proof. We sketch the proof for Part 1. Verification details are left to the reader. The proof is by induction on complexity of the formula $\phi(x_1, \ldots, x_n)$.

If $\phi(x_1, \ldots, x_n)$ is an atomic formula, the definition of automaticity implies the proof. Assume that ϕ is a disjunction of two formulas, say $\phi_1(x_1, \ldots, x_n)$ and $\phi_2(x_1, \ldots, x_n)$. Since automata recognizable languages are closed under the union operation, using the inductive hypothesis, one constructs an automaton $\mathcal{M}_\phi$ from the automaton $\mathcal{M}_{\phi_1}$ and $\mathcal{M}_{\phi_2}$ that recognizes all tuples that make $\phi(x_1, \ldots, x_n)$ true. The conjunction case is proved similarly. Assume that $\phi(x_1, \ldots, x_n)$ is a negation of the formula $\psi(x_1, \ldots, x_n)$. Then one can construct the automaton $\mathcal{M}_\phi$ for the complement of the language recognized by $\mathcal{M}_\psi$. Now assume that $\phi(x_1, \ldots, x_n)$ is of the form $\exists x_{n+1}\psi(x_1, \ldots, x_n, x_{n+1})$. The automaton $\mathcal{M}_\phi$ is then built from $\mathcal{M}_\psi$ by "forgetting" the last components of the labels in the transition table of $\mathcal{M}_\psi$. The resulting automaton is non-determinitsic and accepts exactly those tuples that make $\phi(x_1, \ldots, x_n)$ true.

Now consider the formula $\phi(\bar{x})$ of the form $\exists^{<\omega} y\psi(\bar{x}, y)$. Assuming that there is an automaton $\mathcal{M}_\psi$ for $\psi(\bar{x}, y)$, we would like to construct an automaton for $\phi(\bar{x})$. If the underlying structure $\mathcal{A}$ is word-automatic then the formula $\exists^{<\omega} y\psi(\bar{x}, y)$ is equivalent to the following formula $\exists z \forall y(y \leq_{pref} z \vee \neg\psi(\bar{x}, y))$. Since $(\mathcal{A}, \leq_{pref})$ is word-automatic, the former formula is the formula of the first order language. The reasoning above shows that there exists an automaton that accepts exactly those tuples $\bar{a}$ that make $\phi(\bar{x})$ true. Assume that $\mathcal{A}$ is a tree-automatic structure. Then the formula $\exists^{<\omega} y\psi(\bar{x}, y)$ is equivalent to the following formula $\exists z \forall y(dom(y) \subseteq dom(z) \vee \neg\psi(\bar{x}, y))$. Note that there exists a tree automata that given two Σ-trees y and z recognizes if $dom(y) \subseteq dom(z)$. Therefore the structure $(\mathcal{A}, \subseteq)$ is tree-automatic structures. As above, we conclude that there exists a tree automaton that accepts exactly those tuples $\bar{a}$ that make $\phi(\bar{x})$ true.

For the second part of the theorem, consider the automatic structure $\mathcal{A}$ and a sentence ϕ. Let $\mathcal{M}_\phi$ be an automaton constructed for ϕ as above. Then ϕ is true in $\mathcal{A}$ if and only if $L(\mathcal{M}_\phi) \neq \emptyset$. Since, the emptiness problem for automata is decidable, we can decide whether or not ϕ is true in $\mathcal{A}$. This proves the theorem.

One can ask several questions concerning generalizations of Theorem 5. For instance, we would like to know for what extension of the $FO + \exists^\omega$-logic and for

which classes of automatic structures the theorem can be preserved. The reader is referred to [22] for this and related questions. We also refer the reader to [23] for a discussion of decision problems on automatic structures.

4 Domain Dependency

Here is an observation that has interesting consequences. Consider the unary domain $1^\star$ (all unary strings over the alphabet $\{1\}$). All linearly ordered word-automatic sets with domain $1^\star$ are finite unions of ω (the order of natural numbers), $\mathbb{Z}$ (the order of integers), ω^- (the order of negative integers), and finite orders [27]. In particular, infinite well ordered sets with domain $1^\star$ are exactly those that are strictly less than ω^2. Now let us change the domain. Say, instead of $1^\star$ we consider the domain $0^\star 1^\star$. One can prove that an infinite well-ordered set with domain $0^\star 1^\star$ is word-automatic if and only if it is strictly less than ω^3. Furthermore, every automatic linearly ordered set with domain $1^\star$ is isomorphic to an automatic linear order with domain $0^\star 1^\star$. This observation tells us that various automata recognizable domains might realize different automatic structures. The definition below takes this observation into account, refines the definition of automaticity and places an emphasis on the underlying automata recognisable domains of the structures.

For the definition, we fix a class of structures K, where structures are identified up to isomorphism. For instance, K can be the class of well-ordered sets, undirected graphs of bounded degree, trees, Abelian groups and so on.

Definition 6. *For a FA recognizable language X,* the algebraic spectrum of X with respect to the class K, *denoted by $AlgSpec_K(X)$, is the class of all structures $\mathcal{B} \in K$ such that there exists a word-automatic structure $\mathcal{A}$ with domain X isomorphic to $\mathcal{B}$. If $\mathcal{B} \in AlgSpec_K(X)$ then we say that* the set X admits (the isomorphism type of) the structure $\mathcal{B}$. *The spectrum $AlgSpec_K(X)$ for tree automata recognisable languages X is defined similarly.*

For example, no tree-automata (or finite automata) recognisable language admits a structure with undecidable first-order theory. The results in [27] show that $0^\star$ admits a well-order α if and only if $\alpha < \omega^2$. In [20] it is proven that if X is regular and X admits an ordinal α then $\alpha < \omega^\omega$. Generally, Khoussainov and Minnes [15] showed that if X is FA recognizable and X admits a well-founded partial order $\mathcal{A}$ then the height of $\mathcal{A}$ is below ω^ω. Another nice example is a recent result by Tsankov [29] showing that no regular language admits the structure $(\mathbb{Q}; +)$, the additive group of rational numbers. The last three examples are simply non-automaticity results (and are therefore domain independent). However, we stress that Definition 6 calls for a refined analysis of automaticity. Proving that a certain structure (e.g. a well order of type ω^n) is not admitted by a given regular or tree-automata recognisable language requires a deep analysis of underlying automata and understanding algebraic and model-theoretic properties of underlying structures of interest.

In order to show examples of domain dependency results, in this section the class K from Definition 6 will be the class of linearly ordered sets. Recall that a structure $\mathcal{A} = (A, \leq)$ is a *linearly ordered set* if $\leq$ is a partial order on A such that for all $x, y \in A$ we have either $x \leq y$ or $y \leq x$. A linearly ordered set $\mathcal{A} = (A, \leq)$ is a *well-order* if every non-empty subset of A has a $\leq$-minimal element. A linearly ordered set $\mathcal{A} = (A, \leq)$ is called *scattered* if no embedding exists from the natural order of the set of all rational numbers into $\mathcal{A}$. There is an equivalent definition of scatteredness in terms of Cantor-Bendixson ranks that we will explain below.

Cantor-Bendixson ranks (CB-ranks for short) are ordinals assigned to linearly ordered sets. Let $\mathcal{L} = (L; \leq)$ be a linearly ordered set. Say that $x, y \in L$ are $\sim$–equivalent if the interval between x and y is a finite set. The relation $\sim$ is an equivalence relation. The order $\leq$ naturally induces a linear order on the quotient set $\mathcal{L}/\sim$. Denote the resulting order by $\mathcal{L}'$. This new linearly ordered set $\mathcal{L}'$ is called *the derivative of* $\mathcal{L}$. We iterate this process of taking derivatives and produce the sequence of derivatives as follows: $\mathcal{L}_0 = \mathcal{L}$, $\mathcal{L}_1 = \mathcal{L}'_0$, $\mathcal{L}_{\alpha+1} = \mathcal{L}_\alpha{}'$ and $\mathcal{L}_\beta$ is the limit of all $\mathcal{L}_\gamma$ with $\gamma < \beta$ for limit ordinals β.

Definition 7. *We say that a linearly ordered set $\mathcal{L}$ is* very discrete *if there exists an α such that $\mathcal{L}_\alpha$ is a finite linearly ordered set. The least ordinal α for which $\mathcal{L}_\alpha$ is finite is called the* Cantor-Bendixson rank *of $\mathcal{L}$. We denote it by* CB-rank($\mathcal{L}$).

It is well-known that $\mathcal{L}$ is very discrete if and only it is scattered $\mathcal{L}$ [26]. Examples of scattered linearly ordered sets are the order of integers, well-ordered sets and their finite sums and products.

Theorem 8. *If $\mathcal{L} = (L; \leq)$ is a word-automatic linearly ordered set with at least one infinite $\sim$-class then the set $\Sigma^\star$ admits $\mathcal{L}$.*

Proof. Consider an infinite $\sim$-equivalence class $[x] = \{y \mid x \sim y \text{ in } \mathcal{L}\}$ that exists by the assumption. This class $[x]$ is a finite automata recognizable language. This follows from the fact that the relation $\sim$ is definable in $FO + \exists^\omega$-logic and Theorem 5. Consider the following regular language:

$$C = [x] \cup (\Sigma^\star \setminus L).$$

The linearly ordered set $([x]; \leq)$, where $\leq$ is the order in $\mathcal{L}$, is isomorphic to either the positive integers or the negative integers or all integers. Assume, without loss of generality, that $([x]; \leq)$ is isomorphic to the positive integers, that is, to the ordinal ω. We now write $\Sigma^\star$ as follows:

$$L_{[x]} \cup C \cup R_{[x]},$$

where $L_{[x]} = \{z \in L \mid z < x \text{ and } z \notin [x]\}$ and $R_{[x]} = \{z \in L \mid z > x \text{ and } z \notin [x]\}$. The languages $L_{[x]}$ and $R_{[x]}$ both are FA recognizable. Define the following linear order $\leq_{new}$. The order $\leq_{new}$ preserves the old order $\leq$ on the sets $L_{[x]}$ and $R_{[x]}$, orders the strings in $[x] \cup (\Sigma^\star \setminus L)$ length-lexicographically, and declares all the elements in C be greater than all elements in $L_{[x]}$, and all the elements in C be

less than all elements in $R_{[x]}$. The linear order $\leq_{new}$ is clearly finite automata recognisable.

It is easy to see that $\leq_{new}$ is a linear order on $\Sigma^\star$. In addition, the original word-automatic linearly ordered set $\mathcal{L}$ is isomorphic to $(\Sigma^\star; \leq_{new})$. Hence, $\Sigma^\star$ admits $\mathcal{L}$. We have proved the theorem. □

Since every word automatic infinite scattered linearly ordered set satisfies the hypothesis of the theorem above, we have the following corollary. The corollary states that the algebraic spectrum of $\Sigma^\star$ in the class of linearly ordered sets contains all word-automatic scattered linear orders.

Theorem 9 (Scaterdness Theorem (with Jain and Stephan)). *The set $\Sigma^\star$ admits every word automatic infinite scattered linearly ordered set.* □

We do not know if the theorem above is true for all infinite word-automatic linear orders.

We would now like to consider the case of tree-automatic scattered linear orders. To contrast the situation with word-automatic case, instead of the set of all finite strings $\Sigma^\star$, consider the set of all Σ-trees $T(\Sigma)$. A natural question arises if the set $T(\Sigma)$ can admit a tree-automatic scattered linear order. It turns out that the situation here differs dramatically from finite automata case. For instance, using (1) Gurevich and Shelah's theorem stating that no monadic second-order definable choice function exists on the infinite binary tree T_2 [13] and (2) finite-set-interpretablity of tree-automatic structures on T_2 [8], one can show that no tree-automatic well-order exists on the set of all Σ-trees $T(\Sigma)$. The theorem below greatly extends this to scattered linearly ordered sets. The proof of the theorem consists of a delicate analysis of tree automatic linear orders on the set $T(\Sigma)$ of all finite Σ-trees.

Theorem 10 (Non-scaterdness Theorem (with Jain and Stephan)). *The set $T(\Sigma)$ does not admit a tree-automatic scattered linear order.* □

Naturally, one wonders if $T(\Sigma)$ admits at least one tree-automatic linear order. Below we prove that $T(\Sigma)$ admits a tree-automatic linear order of the type of rational numbers. We assume that Σ is not a unary alphabet, say $\Sigma = \{a, b\}$ with $a < b$.

Proposition 11 (with Sanjay and Stephan). *The language $T(\Sigma)$ admits a tree-automatic linear order of the type of rational numbers.*

Proof. For any given two trees $p, q \in T(\Sigma)$ such that $p \neq q$, consider the left-most node $x_{(p,q)}$ in the convolution tree $conv(p, q)$ for which $p'(x_{(p,q)}) \neq q'(x_{(p,q)})$, where p' and q' are defined in the definition of convolution operation for trees. Here the left-most node is taken with respect to the pre-order on the nodes of the tree $conv(p, q)$. Now we define the relation $\sqsubseteq$ on $T(\Sigma)$ as follows. For trees $p, q \in T(\Sigma)$ declare $p \sqsubseteq q$ if and only if either $p = q$ or $conv(p, q)(x_{(p,q)}) \in \{(a, b), (\Box, b), (a, \Box)\}$.

We now claim that the relation $\sqsubseteq$ is the desired one. A tree automaton recognising this relation can be described as follows. On input $conv(p, q)$ the automaton non-deterministically selects a path leading to $x_{(p,q)}$. At all nodes v left of $x_{(p,q)}$ the automaton verifies that $p(v) = q(v)$. Once the node $x_{(p,q)}$ is reached the automaton accepts the tree. If node $x_{(p,q)}$ does not exist then the automaton fails along the non-deterministically chosen path that searches for $x_{(p,q)}$.

It is not hard to verify that the relation $\sqsubseteq$ is a linear order on $T(\Sigma)$. We need to show that $\sqsubseteq$ is dense and has no end-points. To show that the relation has no endpoints, let us take a tree $p \in T(\Sigma)$. Let v be any leaf of p; thus $x_{(p,q)}$ is a prefix of v . We now extend p to p_1 such that $p_1(v0) = a$ and $p_1(v1) = b$, and we extend p to p_2 such that $p_2(v0) = b$ and $p_2(v1) = b$. In this way we have $p_1 \sqsubseteq p \sqsubseteq p_2$.

Let p, q be such that $p \neq q$ and $p \sqsubseteq q$. Consider $x_{(p,q)}$. Assume that $p(x_{(p,q)}) = a$ and either $q(x_{(p,q)}) = b$ or $q(x_{(p,q)}) = \Box$. Let v be a leaf of p above $x_{(p,q)}$. Extend p to p_2 (as above) using v. Then $p \sqsubseteq p_2 \sqsubseteq q$. Assume that $p(x_{(p,q)}) = \Box$ and $q(x_{(p,q)}) = b$. Let w be a leaf of q above $x_{(p,q)}$. Extend q to q_1 using the node w. Then $p \sqsubseteq q_1 \sqsubseteq q$. Hence the linear order $\sqsubseteq$ is dense.

Finally, we point out that Definition 6 implies the following partial order on the set of all finite automata (tree automata) recognizable languages over the alphabet Σ. For two infinite (tree automata) finite automata recognizable sets X and Y we write

$$X \leq_K Y \text{ if and only if } AlgSpec_K(X) \subseteq AlgSpec_K(Y).$$

There are several interesting questions about this partial order. For instance, does it have a maximal and minimal elements? What is the height and the width of this partial order? Is the partial order semi-lattice? Can the order of rational numbers be embedded into this partial order? Is there an ω-chain in the partial order? Of course, all these depend on the class K selected. These questions are of interest since they call for the investigation of interactions between automata and properties of abstract mathematical structures. Interesting cases for selecting the class K are the classes of all structures, linearly ordered sets, trees, and various algebraic structures such as Boolean algebras, groups, and lattices.

5 Automaticity in Groups

In this section we introduce automaticity in groups through their Cayley graphs. Our definition naturally extends the known definition of automaticity given by Thurston and his collaborators [11]. We start by considering a labeled directed graph $\Gamma = (V, E)$. The labels of the graph are taken from a finite set Σ of labels. Let $\sigma_1, \ldots, \sigma_n$ be all labels from Σ.

Definition 12. *We view the graph Γ as the following structure:*

$$(V, E_{\sigma_1}, \ldots, E_{\sigma_n}),$$

where $E_\sigma = \{(x,y) \mid (x,y) \in E$ *and the label of* (x,y) *is* $\sigma\}$ *for* $\sigma \in \Sigma$. *We say that the graph* Γ *is* **word-automatic** *if the structure* $(V, E_{\sigma_1}, \ldots, E_{\sigma_n})$ *is word-automatic.*

Below we give two examples.

Example 1. Let T be a Turing machine. The configuration space of T is the graph $(Conf(T), E_T)$, where the set $Conf(T)$ is the set of all configurations of T, and the set E_T of edges consists of all pairs (c_1, c_2) of configurations such that T has an instruction that transforms c_1 to c_2. The structure $(Conf(T), E_T)$ is clearly an automatic directed graph since the transitions $(c_1, c_2) \in E_T$ can be detected by finite automata.

Example 2. Consider the n-dimensional grid $\mathbb{Z}^n$ as a labeled graph, where the labels are $e_1, \ldots, e_n$. Identify each e_i with the vector $(0, \ldots, 0, 1, 0, \ldots, 0)$, whose all components are 0 except at position i. For any two vectors v and w in $\mathbb{Z}^n$, put an edge from v to w and label it with e_i if $v + e_i = w$. We represent each vector $v \in \mathbb{Z}^n$ as an n-tuple $(x_1, \ldots, x_n)$ of integers each written in a binary (or decimal) notation. Under this coding, the edge relation

$$E_i = \{(v, w) \mid v + e_i = w\}$$

is clearly finite automata recognizable. Hence, the labeled graph $\mathbb{Z}^n$ is word-automatic.

Let G be a group generated by a finite set X. We always assume that X is closed under the inversion operation, that is $x^{-1} \in X$ for all $x \in X$. We also assume that the group G is infinite. The group G and X determine the following graph, called *Cayley graph of* G and denoted by $\Gamma(G, X)$. The vertices of the graph are the elements of the group. For each vertex g we put a directed edge from g to gx, where $x \in X$, and label the edge by x. Thus, $\Gamma(G, X)$ is a labeled directed graph. The proof of the lemma below is standard:

Lemma 13. *The Cayley graph* $\Gamma(G, X)$ *satisfies the following properties:*

1. *The graph is strongly connected.*
2. *The out-degree and in-degree of each node is bounded by* $|X|$.
3. *The graph is transitive, that is, for any two vertices* g_1 *and* g_2 *of the graph there exists an automorphism* α *such that* $\alpha(g_1) = g_2$.
4. *The group of automorphisms of* $\Gamma(G, X)$ *is isomorphic to* G. □

Our definition of automaticity for groups is now the following:

Definition 14. *Let* G *be a group generated by a finite set* X *of generators. We say that the group* G *is* **Cayley automatic** *if the graph* $\Gamma(G, X)$ *is an automatic graph.*

We give several examples.

Example 3. Consider a finitely generated abelian group G. The group G can be written as $\mathbb{Z}^n \bigoplus A$, where A is a finite abelian group and $n \in \mathbb{N}$. The group

G is generated by A and the vectors $e_1, \ldots, e_n$ in $\mathbb{Z}^n$. Using the fact that the n-dimensional grid $\mathbb{Z}^n$ is automatic and that A is finite, it is easy to show that the group G is Cayley automatic.

Example 4. Consider the Heisenberg group $\mathcal{H}_3(\mathbb{Z})$ consisting of 3×3 matrices over $\mathbb{Z}$ whose entries below the diagonal are all 0 and entries at the diagonal are 1. We identify these matrices with 3-tuples (a, b, c), where a, b, c are integers written in binary. The multiplication in the group is given by the following rule:

$$(a, b, c) \cdot (x, y, z) = (a + x, b + y + az, c + z).$$

The group has 3 generators $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. The multiplication of group elements (a, b, c) by each of these generators gives the following equalities:

$$\begin{aligned}(a, b, c) \cdot (1, 0, 0) = (a + 1, b, c),\ (a, b, c) \cdot (0, 1, 0) \\ = (a, b + 1, c), (a, b, c) \cdot (0, 0, 1) = (a, b, c + 1).\end{aligned}$$

Clearly, each of these operations can be performed by finite automata. Thus, the group $\mathcal{H}_3(\mathbb{Z})$ is Cayley automatic.

Example 5. The example above can clearly be generalized to Heisenberg groups $\mathcal{H}_n(\mathbb{Z})$ consisting of all $n \times n$ matrices over $\mathbb{Z}$ such that they contain 1 on the diagonal, and all other entries are 0 apart from the entries at the first row or the last column.

Our goal is to show that Cayley automaticity is preserved under several group-theoretic constructions. The next proposition shows that the definition of Cayley automaticity does not depend on the generators.

Proposition 15 (with Kharlampovich and Miasnikov). *If G is a Cayley automatic group with respect to a generating set X then G is Cayley automatic with respect to any generating set Y of G.*

Proof. We start with the following easy which is readily proved through decomposition of finite automata:

Lemma 16. *Let G be Cayley automatic group over a generator set X. Then for all $x_1, x_2 \in X$ there exists a finite automaton $\mathcal{M}_{x_1x_2}$ such that for all $v, w \in \Gamma(G, X)$, the automaton $\mathcal{M}_{x_1x_2}$ detects if $v = wx_1x_2$.*

Consider now the automatic Cayley graph $\Gamma(G, X)$. Each $y \in Y$ can be written as a product $x_1^{k_1} \cdot \ldots \cdot x_n^{k_n}$ of elements of X. We write this product as $w(y)$. Since $\Gamma(G, X)$ is automatic there exists an automaton $\mathcal{M}_x$, $x \in X$, such that for all $v, w \in \Gamma(G, X)$, the automaton $\mathcal{M}_x$ detects if $v = w \cdot x$. By the lemma above, we can use the automata $\mathcal{M}_x$, $x \in X$, to build a finite automaton $M_{w(y)}$ that recognizes all $v_1, v_2 \in \Gamma(G, X)$ such that $v_1 = v_2 y$. This proves that $\Gamma(G, Y)$ is an automatic graph.

Let G be a group and H be a normal subgroup of G. We say that G is a *finite extension of H* if the quotient group G/H is finite. It turns out that finite extensions preserve Cayley automaticity:

Theorem 17 (with Kharlampovich and Miasnikov). *Finite extensions of Cayley automatic groups are Cayley automatic.*

Proof. Let H be a Cayley automatic group. Let $H \trianglelefteq G$ be a normal subgroup of a group G such that G/H is finite. Let

$$G/H = \{Hk_0, \ldots, Hk_{r-1}\}$$

There exists a finite function g such that for all $i, s \leq r-1$, we have the equality:

$$(\star) \qquad Hk_i \cdot Hk_s = Hk_{g(i,s)}.$$

Let $h_0, \ldots, h_{n-1}$ be a finite number of generators of H that also include the identity of the group. The equality $(\star)$ above implies that there are sequences $g_1(i,s)$, …, $g_x(i,s)$ and $u_1(i,s)$, …, $u_x(i,s)$ of integers such that we have

$$k_i k_s = h_{u_1(i,s)}^{g_1(i,s)}, \ldots h_{u_x(i,s)}^{g_x(i,s)} k_{g(i,s)},$$

where $i, s \leq r-1$ and all $u_1(i,s)$, …, $u_x(i,s)$ are non-negative integers all less than or equal to $n-1$. Similarly, there are sequences $f_1(i,j,s)$, …, $f_m(i,j,s)$ and $v_1(i,j,s)$,…, $v_m(i,j,s)$ of integers such that for all $i, s \leq r-1$ and $j \leq n-1$ we have the following equalities:

$$k_i h_j k_s = h_{v_1(i,j,s)}^{f_1(i,j,s)} \cdot \ldots \cdot h_{v_m(i,j,s)}^{f_m(i,j,s)} k_i k_s.$$

This implies that for all all $s, i \leq r-1$, $j \leq n-1$, and $h \in H$ we have the following equalities:

$$\begin{aligned} hk_i\ h_j k_s &= h\ h_{v_1(i,j,s)}^{f_1(i,j,s)} \cdot \ldots \cdot h_{v_m(i,j,s)}^{f_m(i,j,s)} k_i k_s \\ &= h\ h_{v_1(i,j,s)}^{f_1(i,j,s)} \cdot \ldots \cdot h_{v_m(i,j,s)}^{f_m(i,j,s)}\ h_{u_1(i,s)}^{g_1(i,s)}, \ldots h_{u_x(i,s)}^{g_x(i,s)}\ k_{g(i,s)}. \end{aligned}$$

Let $\bar{h}$ be the word representing the element $h \in H$ under a Cayley automatic presentation of H. We represent elements hk of the group G as words $\bar{h}k$. Here we need to assume that the alphabet of the presentation for H does not contain symbols k_0, …, k_{r-1}. The equalities above tell us that there are finite automata $M_{i,j}$ that for every k_i, h_j accept all pairs of words of the form $(\bar{h}k, w)$ such that the equality $w = hkk_ih_j$ is true in the group G. Note that to build the automata $M_{i,j}$ one needs to use (1) the original automata that represent the group H, (2) remember the sequences $g_1(i,s)$, …, $g_x(i,s)$ and $u_1(i,s)$, …, $u_x(i,s)$, (3) the sequences $f_1(i,j,s)$, …, $f_m(i,j,s)$ and $v_1(i,j,s), \ldots, v_m(i,j,s)$, (3) the function g, and (4) build automata for representing the multiplication by elements $h_v^{f(i,j,s)}$ and $h_u^{g(i,j)}$ in the group H. This shows that the group G is Cayley automatic. The theorem is proved.

For the next theorem we define the *restricted wreath product* of two groups A and B. Let A_b be an isomorphic copy of A for each $b \in B$. Consider the direct sum of groups A_b denoted by K. Thus,

$$K = \bigoplus_{b \in B} A_b,$$

where elements of K are functions $f : B \to A$ such that $f(b) = 1_A$ for almost all $b \in B$. We write the elements of K as (a_b). Each element $c \in B$ induces an automorphism α_c of K as follows:

$$\alpha_c(a_b) = (a_{bc}).$$

The wreath product of A by B consists of all pairs of the form (b, k), where $b \in B$ and $k \in K$, with multiplication defined by:

$$(b_1, k_1) \cdot (b_2, k_2) = (b_1 b_2, \alpha_{b_2}(k_1)k_2).$$

The following theorem gives examples of Cayley automatic groups.

Theorem 18 (with Nies). *For every finite group G, the restricted wreath product of G by $\mathbb{Z}$ is Cayley automatic.*

Proof. Consider the restricted wreath product of G by $\mathbb{Z}$. The elements of the wreath product are of the form

$$(i, (\ldots, g_{-n}, g_{-n+1}, \ldots, g_{-1}, g_0, g_1, \ldots, g_{m-1}, g_m, \ldots)),$$

where each $g_j \in G$ and $i \in Z$. We refer to g_0 as the element of G at position 0. Assuming that g_k is the identity 1_G of the group G for all $k > m$ or $k < -n$, and $g_{-n} \neq 1_G$, $g_m \neq 1_G$, we can represent the element above as the following string

$$conv(i, g_{-n} \ldots g_{-1}(g_0, \star)g_1 \ldots g_m),$$

where i is written in binary. Recall that *conv* represents the convoluted string that represents the tuple $(i, g_{-n} \ldots g_{-1}(g_0, \star)g_1 \ldots g_m)$ (See Section 2). The alphabet of these strings is finite since G is a finite group. The symbol $\star$ represents elements of G at position 0. The generators of the wreath product are elements represented by the strings $conv(0, g)$ and $conv(1, 1_G)$, where $g \in G$. Multiplication by these generators works as follows:

$$conv(i, g_{-n} \ldots g_{-1}(g_0, \star)g_1 \ldots g_m) \cdot conv(0, g) = conv(i, g_n \ldots g_{-1}(g_0 \cdot g, \star)g_1 \ldots g_m)$$

and

$$conv(i, g_n \ldots g_{-1}(g_0, \star)g_1 \ldots g_m) \cdot conv(1, 1_G) = conv(i + 1, g'_{n+1} \ldots (g'_0, \star)g'_1 \ldots g'_{m+1}),$$

where $g'_{j+1} = g_j$ for $j \in \{-n, \ldots, m\}$. These operations can clearly be performed by finite automata. The theorem is proved.

Next, we consider the amalgamated free product of groups. Let A and B be Cayley automatic groups. Assume that H is a subgroup of both A and B. The *amalgamated product* $A \star_H B$ is a group obtained from the free product $A \star B$ of groups A and B in which the subgroup H in A and B become identified. It turns out that the amalgamated product is also Cayley automatic if H satisfies some natural automata-theoretic condition. We say that the subgroup H is a **uniformly regular subgroup of A and B** if the equivalence relations

$\sim_H^A = \{(x,y) \mid x,y \in A \text{ and } xy^{-1} \in H\}$ and $\sim_H^B = \{(x,y) \mid x,y \in B \text{ and } xy^{-1} \in H\}$ are both FA recognizable.

We note that if H is a regular subgroup of a Cayley automatic group A then the cosets Ha, where $a \in A$, are all FA recognizable. However, this does not guarantee that $\sim_H^A$ is regular. On the other hand, strong regularity of H guarantees that the cosets Ha are all regular. The proof of the theorem below uses A-normal forms that represent the group elements of the amalgamated product (see for instance, [6]). The set of all A-normal forms is a FA recognizable set.

Theorem 19 (with Kharlampovich and Miasnikov). *If A, B are Cayley automatic groups and H is a uniformly regular subgroup of A and B then the amalgamated product $A \star_H B$ is Cayley automatic. In particular, the free product of Cayley automatic groups is Cayley automatic.* □

Much more can be said about Cayley automatic groups. For instance, we would like to know the geometry of Cayley graphs of such groups. However, we leave this to further investigations. We note that in [25], an amalgamated products of abelian automatic groups is studied. In the paper, automatic groups are the groups in which the graph of the group operation is FA recognizable.

6 Thurston Automaticity vs. Cayley Automaticity

We now define Thurston automatic groups and show that the class of Thurston automatic groups is properly contained in the class of Cayley automatic groups. So, let G be a finitely generated group generated by a finite set X of generators. We assume that X is closed under inverses, that is $x^{-1} \in X$ if and only if $x \in X$. Consider the free group $F(X)$ generated by X. The elements of $F(X)$ are in their reduced form (that is, they dont contain sub-words of the form xx^{-1}, where $x \in X$). There exists a natural homomorphism $v \to \bar{v}$ that associates the string v in the group $F(X)$ with its value $\bar{v}$ in the group G.

Below we define groups that are automatic in the sense of Thurston and studied by many experts in group theory, geometric group theory and topology (e.g. D. Epstein, J. W. Cannon, D. F. Holt, S. Levy, M. S. Paterson, and W. Thurston). We refer to these groups as Thurston-automatic groups. There is a wide variety of finitely generated groups that are Thurston automatic. These include all virtually abelian groups, hyperbolic groups, braid groups, and fundamental groups of large variety of 3-manifolds.

Definition 20. *The group G is Thurston-automatic if there exist a regular language $L \subseteq X^\star$ of reduced words and finite automata $\mathcal{M}_x$, $x \in X$, such that the following properties are satisfied:*

1. *The mapping $f : L \to G(X)$ defined as $f(v) = \bar{v}$, with $v \in L$, is a bijection from L onto G.*
2. *For each $x \in X$, we have $L(\mathcal{M}_x) = \{(u,v) \mid u,v \in L \text{ and } \bar{u}x = \bar{v}\}$*

In the last two decades the study of Thurston-automatic groups has attracted the attention of many experts in group theory. See for instance [7] [11] [12] . We note that all Thurston-automatic groups are Cayley automatic.

Here we would like to consider nilpotent groups. For this we need to give several definitions and notations. Let G be a group. Recall the commutator of two elements $x, y \in G$ is given by

$$[x, y] = x^{-1}y^{-1}xy.$$

Let H, K be subsets of the group $\mathcal{G}$ whose unit element is denoted by e. Define the set $[H, K] = \{[h, k] \mid h \in H, k \in K\}$. If H and K are normal subgroups of $\mathcal{G}$ then so is $[H, K]$. Now define the following two sequence:

$$\gamma_0(G) = G, and\ \gamma_{k+1}(G) = [\gamma_k(G), G]\ for k > 0.$$

We say that $\mathcal{G}$ is *nilpotent* if $\gamma_r(G) = \{e\}$ for some $r \in \omega$. The least t such $\gamma_t(G) = \{e\}$ is called the *nilpotency class* of the group G. Examples of nilpotent groups are Heisenberg groups $\mathcal{H}_n(\mathbb{Z})$ given in Examples 5, 6.

It turns out that nilpotent groups that are Thurston-automatic are virtually abelian groups [11]; virtually abelian groups are those groups that have abelian normal subgroups of finite index. The Heisenberg group $\mathcal{H}_3(\mathbb{Z})$ in Example 5 is nilpotent and not a virtually abelian group. This group is Cayley automatic. Hence the class of all Cayley automatic groups contains the class of Thurston-automatic groups. In fact, the next theorem shows that Cayley automatic groups contain a large class of nilpotent groups. The proof of the theorem uses the polycyclic nature of nilpotent groups, and revisits matrix representations of nilpotent groups over integers:

Theorem 21 (with Kharlampovich and Miasnikov). *Every finitely generated group of nilpotency class at most two is Cayley automatic.* □

Thurston-automatic groups also satisfy the following nice property. They are all finitely presented. In contrast, the class of Cayley-automatic groups contains a large class of groups that are not finitely presented. Namely, the following is true:

Proposition 22 (with Nies). *There exist Cayley automatic but not finitely presented groups.*

Proof. The restricted wreath product of a non-trivial finite group G by $\mathbb{Z}$, by Theorem 18, is Cayley automatic. Now we use the following theorem by Baumslag [3]. For finitely presented groups A and B, the restricted wreath product of A by B is finitely presented if and only if either A is trivial or B is finite. Hence, for nontrivial finite group G, the restricted wreath product of G by Z is not finitely presented but Cayley automatic.

7 The Isomorphism Problem

We are often concerned with classifying structures up to isomorphism. This typically amounts to finding invariants of structures that describe their isomorphism types. To give an algorithmic spin to the isomorphism problem one would like to have finite descriptions of structures. For instance, automatic structures have finite descriptions, and these descriptions are automata that recognize the domain and relations of the structures. Similarly, finitely presented groups have finite descriptions given through finite set of generators X and finite set of relators R. From an algorithmic perspective, the isomorphism problem asks if there exists an algorithm that given finite descriptions of two structures decides whether the structures are isomorphic. A classical example here is the isomorphism problem asked by Dehn already at the start of the twentieth century [9] [10]. Dehn asked to design algorithms that given two finite presentations of groups establishes if the groups are isomorphic. In the context of automatic structures, the isomorphism problem is formulated as follows.

Let K be a class of automatic structures. Design an algorithm that given two automatic structures $\mathcal{A}$ and $\mathcal{B}$ from the class K decides if $\mathcal{A}$ and $\mathcal{B}$ are isomorphic. The point here is that both the structures $\mathcal{A}$ and $\mathcal{B}$ are given by automata that represent their domains and atomic relations.

It turns out that the isomorphism problem for automatic structures is undecidable. This was first noted by Blumensath in [4]. The precise complexity of the problem was studied in [15] [16] [19]. We informally explain here how hard it is to find out if two automatic structures are isomorphic. Consider the following set (known as the Halting set for Turing machines):

$$H = \{M \mid \text{ The Turing machine } M \text{ halts at some input}\}.$$

No algorithm exists that computes this set. We can now iterate this process using ordinals as follows. Set $H^1 = H$. For a successor ordinal $\alpha = \beta + 1$, consider

$$H^\alpha = \{M \mid \text{The Turing machine } M, \text{ that has access to an oracle for } H^\beta, \text{ halts at some input}\}.$$

No algorithm exists that computes H^α even if one allows to use an oracle that knows the set H^β. For α limit ordinal, we set

$$H^\alpha = \bigoplus_{\beta<\alpha} H^\beta,$$

where $\bigoplus$ represents a disjoint union. As above, no algorithm exists that computes H^α even if one allows to use an oracle that knows the set H^β for any given β. The set H^α is called the *α-jump* of the halting set H. The following theorem is implicit in [15] [16]:

Theorem 23. *No algorithm exists that, given two word-automatic structures $\mathcal{A}$ and $\mathcal{B}$, decides if $\mathcal{A}$ and $\mathcal{B}$ are isomorphic even if the algorithm uses an oracle for the α-jump of the halting set H, where α is any given computable ordinal.* □

In spite the fact that the isomorphism problem for automatic structures is highly undecidable, there are some natural classes of structures where the isomorphism problem can be decided. These include the classes of word-automatic well-ordered sets and Boolean algebras.

As an example, we give here an algorithm that solves the isomorphism problem for word-automatic ordinals. Recall that by Cantor's normal form theorem if α is an ordinal then it can be uniquely decomposed as $n_1\omega^{\alpha_1} + n_2\omega^{\alpha_2} + \ldots + n_k\omega^{\alpha_k}$, where $\alpha_1, \alpha_2, \ldots, \alpha_k$ are ordinals satisfying $\alpha_1 > \alpha_2 > \ldots > \alpha_k$ and $k, n_1, n_2, \cdots, n_k$ are natural numbers. Our proof of deciding the isomorphism problem for word-automatic ordinals is based on the following two facts: (1) An ordinal is word-automatic if and only if it is strictly less that ω^ω [20]; (2) The Cantor's normal form of a word-automatic ordinal α can be extracted from its word-automatic presentations. The first fact is equivalent to saying that the Cantor-Bendixson rank of α is finite. Below we provide an algorithm that given word-automatic ordinal α computes its Cantor normal form.

Assume that we have a word-automatic presentation for an ordinal α. The presentation is given by a regular set $R \subseteq \Sigma^*$ for some alphabet Σ and an automaton for the ordering $\leq_{ord}$ on R. Recall that the ordinal α represented by $(R, \leq_{ord})$ is of the form

$$\alpha = n_m\omega^m + n_{m-1}\omega^{m-1} + \ldots + n_2\omega^2 + n_1\omega + n_0$$

where $m, n_m, n_{m-1}, \ldots, n_1, n_0$ are natural numbers. The algorithm below computes compute the integers $m, n_0, n_1, \ldots$:

1. **Input** the presentation $(R, \leq_{ord})$.
2. Let $D = R$, $m = 0$, $n_m = 0$.
3. **While** $D \neq \emptyset$ **Do**
4. **If** D has a maximum u
   Then Let $n_m = n_m + 1$, let $D = D - \{u\}$.
   Else Let $L \subseteq D$ be the subset of limit ordinals in D; that is L is the set of all $x \in D$ with no predecessor in D. Replace D by L, let $m = m + 1$, let $n_m = 0$.
5. **End While**
6. **Output** the formula

$$n_m\omega^m + n_{m-1}\omega^{m-1} + \ldots + n_2\omega^2 + n_1\omega + n_0$$

   using the current values of $m, n_0, \ldots, n_m$.

Each step in the algorithm is computable by Theorem 5. Removing the maximal element from D reduces the ordinal represented of D by 1 while the corresponding n_m is increased by 1. Replacing D by the set of its limit ordinals preserves automaticity. This can be proved from Theorem 5 using the fact that $\sim$-equivalence relation is FA recognizable. Thus, the algorithm computes the coefficients $n_0, n_1, \ldots$ in this order. The algorithm eventually terminates since m is bounded by the Cantor-Bendixson rank of α. The following corollary is immediate.

Theorem 24. *[20] The isomorphism problem for automatic ordinals is decidable.*

Proof. Given two automatic presentations $\mathcal{A}$ and $\mathcal{B}$ of ordinals α and β, we extract the Cantor normal form for both these ordinals. Then the ordinals are isomorphic if and only if their Cantor normal forms are identical.

We point out that given an automatic linearly ordered set, one can effectively decide if the linear order is a well-order. Therefore, the theorem above can be strengthen. Namely, given two automatic structures, one can effectively decide if these two automatic structures are isomorphic ordinals [20].

It had been a long standing question if the isomorphism problem for word-automatic linearly ordered sets is decidable. If so, this would greatly extend the theorem above. However, Kuske, Liu and Lohrey have recently proved that the isomorphism problem for word-automatic linear orders is undecidable [21]. There still remain many open questions on solving the isomorphism problem for various classes of word and tree-automatic structures. For instance, these include automatic groups.

The author would like to thank Andre Nies for proof-reading and comments on this paper.

References

1. Bartholdi, L., Grigorchuk, R.I., Šuniḱ, Z.: Branch groups. In: Handbook of Algebra, vol. 3, pp. 989–1112. North-Holland, Amsterdam (2003)
2. Bartholdi, L., Henriques, A.G., Nekrashevych, V.V.: Automata, groups, limit spaces, and tilings. J. Algebra 305(2), 629–663 (2004)
3. Baumslag, G.: Wreath products and finitely presented groups. Math. Z. 75, 22–28 (1960/1961)
4. Blumensath, A.: Automatic structures. Diploma thesis, RWTH-Aachen (1999)
5. Blumensath, A., Grädel, E.: Automatic structures. In: 15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, CA, pp. 51–62. IEEE Computer Society Press, Los Alamitos (2000)
6. Bogopolski, O.: Introduction to group theory. EMS Textbooks in Mathematics. European Mathematical Society (EMS), Zürich (2008); x+177 Translated, revised and expanded from the 2002 Russian original
7. Choffrut, C.: A short introduction to automatic group theory. In: Semigroups, Algorithms, Automata and Languages, Coimbra, 2001, pp. 133–154. World Sci., River Edge (2002)
8. Colcombet, T., Löding, C.: Transforming structures by set interpretations. Logical Methods in Computer Science 3(2:4), 1–36 (2007)
9. Dehn, M.: Über unendliche diskontinuierliche Gruppen. Math. Ann. 71(1), 116–144 (1911)
10. Dehn, M.: Transformation der Kurven auf zweiseitigen flächen. Math. Ann. 72(3), 413–421 (1912)
11. Epstein, D.B.A., Cannon, J.W., Holt, D.F., Levy, S.V.F., Paterson, M.S., Thurston, W.P.: Word Processing in Groups. Jones and Bartlett, Boston (1992)

12. Gersten, S.: Introduction to hyperbolic and automatic groups. In: Summer School in Group Theory in Banff, 1996. CRM Proc. Lecture Notes, vol. 17, pp. 45–70. Amer. Math. Soc., Providence (1999)
13. Gurevich, Y., Shelah, S.: Rabin's uniformization problem. J. Symb. Log. 48(4), 1105–1119 (1983)
14. Hodgson, B.R.: On direct products of automaton decidable theories. Theoretical Computer Science 19(3), 331–335 (1982)
15. Khoussainov, B., Minnes, M.: Model-theoretic complexity of automatic structures. Ann. Pure Appl. Logic 161(3), 416–426 (2009)
16. Khoussainov, B., Minnes, M.: Three lectures on automatic structures. In: Proceedings of Logic Colloquium 2007. Lecture Notes in Logic, vol. 35, pp. 132–176 (2010)
17. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995)
18. Khoussainov, B., Nerode, A.: Open questions in the theory of automatic structures. Bull. Eur. Assoc. Theor. Comput. Sci. EATCS (94), 181–204 (2008)
19. Khoussainov, B., Nies, A., Rubin, S., Stephan, F.: Automatic structures: Richness and limitations. In: 19th IEEE Symposium on Logic in Computer Science, LICS 2004, Turku, Finland, July 14-17. Logical Methods in Computer Science, vol. 3(2), pp. 44–53. IEEE Computer Society, Los Alamitos (2007)
20. Khoussainov, B., Rubin, S., Stephan, F.: Automatic linear orders and trees. ACM Trans. Comput. Log 6(4), 675–700 (2005)
21. Kuske, D., Liu, J., Lohrey, M.: The isomorphism problem on classes of automatic structures. In: LICS, pp. 160–169. IEEE Computer Society, Los Alamitos (2010)
22. Kuske, D., Lohrey, M.: First-order and counting theories of ω-automatic structures. J. Symb. Log 73(1), 129–150 (2008)
23. Kuske, D., Lohrey, M.: Some natural decision problems in automatic graphs. J. Symb. Log 75(2), 678–710 (2010)
24. Nekrashevych, V.: Self-similar groups. In: Mathematical Surveys and Monographs, vol. 117. American Mathematical Society, Providence (2005)
25. Nies, A., Semukhin, P.: Finite automata presentable abelian groups. Ann. Pure Appl. Logic 161(3), 458–467 (2009)
26. Rosenstein, J.G.: Linear orderings. Academic Press, New York (1981)
27. Rubin, S.: Automatic Structures. PhD dissertation, The University of Auckland (2004)
28. Rubin, S.: Automata presenting structures: A survey of the finite string case. Bulletin of Symbolic Logic 14(2), 169–209 (2008)
29. Tsankov, T.: The additive group of the rationals does not have an automatic presentation. Journal for Symbolic Logic (to appear)

Vector Addition System Reachability Problem: A Short Self-contained Proof*

Jérôme Leroux

LaBRI, Université de Bordeaux, CNRS, France
leroux@labri.fr

Abstract. A central problem of net theory is the reachability problem for Vector Addition Systems (VASs). The general problem is known to be decidable by algorithms exclusively based on the classical Kosaraju-Lambert-Mayr-Sacerdote-Tenney decomposition (KLMTS decomposition). Recently from this decomposition, we deduced that a final configuration is not reachable from an initial one if and only if there exists a Presburger inductive invariant that contains the initial configuration but not the final one. Since we can decide if a Preburger formula denotes an inductive invariant, we deduce from this result that there exist checkable certificates of non-reachability in the Presburger arithmetic. In particular, there exists a simple algorithm for deciding the general VAS reachability problem based on two semi-algorithms. A first one that tries to prove the reachability by enumerating finite sequences of actions and a second one that tries to prove the non-reachability by enumerating Presburger formulas. In this paper we provide the first proof of the VAS reachability problem that is not based on the KLMST decomposition. The proof is based on the notion of production relations, inspired from Hauschildt, that directly proves the existence of Presburger inductive invariants.

1 Introduction

Vector Addition Systems (VASs) or equivalently Petri Nets are one of the most popular formal methods for the representation and the analysis of parallel processes [1]. Their reachability problem is central since many computational problems (even outside the realm of parallel processes) reduce to the reachability problem. Sacerdote and Tenney provided in [9] a partial proof of decidability of this problem. The proof was completed in 1981 by Mayr [7] and simplified by Kosaraju [4] from [9,7]. Ten years later [5], Lambert provided a further simplified version based on [4]. This last proof still remains difficult and the upper-bound complexity of the corresponding algorithm is just known to be non-primitive recursive. Nowadays, the exact complexity of the reachability problem for VASs is

* This version extends the POPL'2011 paper with additional figures and examples. Some classes of sets get more intuitive names like the polytope conic sets, the polytope periodic sets, and the Petri sets that are now called the definable conic sets, the asymptotically definable periodic sets, and the almost semilinear sets.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 41–64, 2011.

still an open-problem. Even the existence of an elementary upper-bound complexity is open. In fact, the known general reachability algorithms are exclusively based on the Kosaraju-Lambert-Mayr-Sacerdote-Tenney (KLMST) decomposition.

Recently [6] we proved thanks to the KLMST decomposition that Parikh images of languages accepted by VASs are semi-pseudo-linear, a class that extends the Presburger sets. An application of this result was provided; we proved that a final configuration is not reachable from an initial one if and only if there exists a forward inductive invariant definable in the Presburger arithmetic that contains the initial configuration but not the final one. Since we can decide if a Presburger formula denotes a forward inductive invariant, we deduce that there exist checkable certificates of non-reachability in the Presburger arithmetic. In particular, there exists a simple algorithm for deciding the general VAS reachability problem based on two semi-algorithms. A first one that tries to prove the reachability by enumerating finite sequences of actions and a second one that tries to prove the non-reachability by enumerating Presburger formulas.

In this paper we provide a new proof of the reachability problem that is not based on the KLMST decomposition. The proof is based on the *production relations* inspired by Hauschildt [3] and it proves directly that reachability sets are *almost semilinear*, a class of sets introduced in this paper that extend the class of Presburger sets and contained in the class of semi-pseudo-linear sets. In particular this paper provides a more precise characterization of the reachability sets of VASs.

Outline of the paper: Section 2 provides notations and classical definitions. Section 3 and Section 4 introduce classes of sets used in the sequel : *definable conic sets* and *vector spaces* in the first one and *asymptotically definable periodic* sets, *Presburger* sets, and *almost semilinear* sets in the second one. Section 5 and Section 6 show that is sufficient to prove that the reachability relation of a Vector Addition system is an almost semilinear relation in order to deduce the existence of forward inductive invariants definable in the Presburger arithmetic proving the non-reachability. In Section 7 we introduce the class of Vector Addition Systems and the central notion of production relations. We show in the next Section 8 that these relations are asymptotically definable periodic. In Section 9 we prove that the reachability relation of a Vector Addition System is an almost semilinear relation. Finally in Section 10 we combine all the previous results to deduce the decidability of the Vector Addition System reachability problem based on Presburger inductive invariants.

2 Notations

We introduce in this section notations and classical definitions used in this paper.

We denote by $\mathbb{N}, \mathbb{N}_{>0}, \mathbb{Z}, \mathbb{Q}, \mathbb{Q}_{\geq 0}, \mathbb{Q}_{>0}$ the set of *natural numbers*, *positive integers*, *integers*, *rational numbers*, *non negative rational numbers*, and *positive rational numbers*. *Vectors* and *sets of vectors* are denoted in bold face. The ith *component* of a vector $\mathbf{v} \in \mathbb{Q}^d$ is denoted by $\mathbf{v}(i)$. We introduce

$||\mathbf{v}||_\infty = \max_{1\leq i\leq d} |\mathbf{v}(i)|$ where $|\mathbf{v}(i)|$ is the *absolute value* of $\mathbf{v}(i)$. The total order $\leq$ over $\mathbb{Q}$ is extended component-wise into an order $\leq$ over the set of vectors $\mathbb{Q}^d$. The addition function $+$ is also extended component-wise over $\mathbb{Q}^d$. Given two sets $\mathbf{V}_1, \mathbf{V}_2 \subseteq \mathbb{Q}^d$ we denote by $\mathbf{V}_1 + \mathbf{V}_2$ the set $\{\mathbf{v}_1 + \mathbf{v}_2 \mid (\mathbf{v}_1, \mathbf{v}_2) \in \mathbf{V}_1 \times \mathbf{V}_2\}$, and we denote by $\mathbf{V}_1 - \mathbf{V}_2$ the set $\{\mathbf{v}_1 - \mathbf{v}_2 \mid (\mathbf{v}_1, \mathbf{v}_2) \in \mathbf{V}_1 \times \mathbf{V}_2\}$. In the same way given $T \subseteq \mathbb{Q}$ and $\mathbf{V} \subseteq \mathbb{Q}^d$ we let $T\mathbf{V} = \{t\mathbf{v} \mid (t, \mathbf{v}) \in T \times \mathbf{V}\}$. We also denote by $\mathbf{v}_1 + \mathbf{V}_2$ and $\mathbf{V}_1 + \mathbf{v}_2$ the sets $\{\mathbf{v}_1\} + \mathbf{V}_2$ and $\mathbf{V}_1 + \{\mathbf{v}_2\}$, and we denote by $t\mathbf{V}$ and $T\mathbf{v}$ the sets $\{t\}\mathbf{V}$ and $T\{\mathbf{v}\}$. In the sequel, an empty sum of sets included in $\mathbb{Q}^d$ denotes the set reduced to the zero vector $\{\mathbf{0}\}$.

A *(binary) relation* R over $\mathbb{Q}^d$ is a subset $R \subseteq \mathbb{Q}^d \times \mathbb{Q}^d$. The *composition* of two relations R and S is the relation denoted by $R \circ S$ and defined as usual by the following equality:

$$R \circ S = \bigcup_{\mathbf{y}\in\mathbb{Q}^d} \big\{(\mathbf{x}, \mathbf{z}) \in \mathbb{Q}^d \times \mathbb{Q}^d \mid (\mathbf{x}, \mathbf{y}) \in R \ \wedge \ (\mathbf{y}, \mathbf{z}) \in S\big\}$$

The *reflexive and transitive closure* of a relation R is denoted by R^*. In this paper, notions introduced over the sets are transposed over the relations by identifying $\mathbb{Q}^d \times \mathbb{Q}^d$ with $\mathbb{Q}^{2d}$.

An order $\sqsubseteq$ over a set S is said to be *well* if for every sequence $(s_n)_{n\in\mathbb{N}}$ of elements $s_n \in S$ we can extract a sub-sequence that is non-decreasing for $\sqsubseteq$, i.e. there exists a strictly increasing sequence $(n_k)_{k\in\mathbb{N}}$ of natural numbers in $(\mathbb{N}, \leq)$ such that $(s_{n_k})_{k\in\mathbb{N}}$ is non decreasing for $\sqsubseteq$. A *minimal element* of an ordered set $(S, \sqsubseteq)$ is an element $s \in S$ such that for every $t \in T$ the relation $t \sqsubseteq s$ implies $s = t$. Given a set $Y \subseteq S$ we denote by $\min_\sqsubseteq(Y)$ the *set of minimal elements* of the ordered set $(Y, \sqsubseteq)$. Let us recall that if $(S, \sqsubseteq)$ is well ordered then $X = \min_\sqsubseteq(Y)$ is finite and for every $y \in Y$ there exists $x \in X$ such that $x \sqsubseteq y$.

Let us consider an order $\sqsubseteq$ over a set S. We introduce the component-wise extension of $\sqsubseteq$ over the set of vectors S^d defined by $\mathbf{s} \sqsubseteq \mathbf{t}$ if $\mathbf{s}(i) \sqsubseteq \mathbf{t}(i)$ for every $i \in \{1, \ldots, d\}$.

Lemma 2.1 (Dickson's Lemma). *The ordered set $(S^d, \sqsubseteq)$ is well for every well ordered set $(S, \sqsubseteq)$.*

Example 2.2. The set $(\mathbb{N}, \leq)$ is well ordered. Hence $(\mathbb{N}^d, \leq)$ is also well ordered. The set $(\mathbb{Z}, \leq)$ is not well ordered.

3 Definable Conic Sets

A *conic set* is a set $\mathbf{C} \subseteq \mathbb{Q}^d$ such that $\mathbf{0} \in \mathbf{C}$, $\mathbf{C} + \mathbf{C} \subseteq \mathbf{C}$ and such that $\mathbb{Q}_{\geq 0}\mathbf{C} \subseteq \mathbf{C}$. A conic set $\mathbf{C}$ is said to be *finitely generated* if there exists a finite sequence $\mathbf{c}_1, \ldots, \mathbf{c}_k$ of vectors $\mathbf{c}_j \in \mathbf{C}$ such that $\mathbf{C} = \mathbb{Q}_{\geq 0}\mathbf{c}_1 + \cdots + \mathbb{Q}_{\geq 0}\mathbf{c}_k$.

Definition 3.1. *We say that a conic set $\mathbf{C}$ is* definable *if it is definable in* FO $(\mathbb{Q}, +, \leq, 0)$.

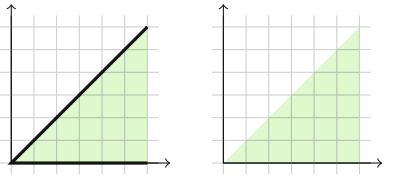


Fig. 1. The finitely generated conic set $\mathbb{Q}_{\geq 0}(1,1) + \mathbb{Q}_{\geq 0}(1,0)$ and the definable conic set $\{(0,0)\} \cup \{(c_1,c_2) \in \mathbb{Q}^2_{>0} \mid c_2 \leq c_1\}$

In this section definable conic sets are geometrically characterized thanks to the *vector spaces* and the *topological closure*.

Example 3.2. Fig. 1 depicts examples of finitely generated conic sets and (non finitely generated) definable conic sets. The conic set $\mathbf{C} = \{(c_1,c_2) \in \mathbb{Q}^2_{\geq 0} \mid \sqrt{2}c_2 \leq c_1\}$ is not definable.

A *vector space* is a set $\mathbf{V} \subseteq \mathbb{Q}^d$ such that $\mathbf{0} \in \mathbf{V}$, $\mathbf{V} + \mathbf{V} \subseteq \mathbf{V}$ and such that $\mathbb{Q}\mathbf{V} \subseteq \mathbf{V}$. Let $\mathbf{X} \subseteq \mathbb{Q}^d$. The following set is a vector space called the *vector space generated* by $\mathbf{X}$.

$$\mathbf{V} = \left\{ \sum_{j=1}^{k} \lambda_j \mathbf{x}_j \mid k \in \mathbb{N} \text{ and } (\lambda_j, \mathbf{x}_j) \in \mathbb{Q} \times \mathbf{X} \right\}$$

This vector space is the minimal for inclusion among the vector space that contains $\mathbf{X}$. Note that the vector space $\mathbf{V}$ generated by a conic set $\mathbf{C}$ satisfies the equality $\mathbf{V} = \mathbf{C} - \mathbf{C}$. Let us recall that every vector space $\mathbf{V}$ is generated by a finite set $\mathbf{X}$ with at most d vectors. The *rank* $\text{rank}(\mathbf{V})$ of a vector space $\mathbf{V}$ is the minimal natural number $r \in \{0,\ldots,d\}$ such that there exists a finite set $\mathbf{X}$ with r vectors that generates $\mathbf{V}$. Note that $\text{rank}(\mathbf{V}) \leq \text{rank}(\mathbf{W})$ for every pair of vector spaces $\mathbf{V} \subseteq \mathbf{W}$. Moreover, if $\mathbf{V}$ is strictly included in $\mathbf{W}$ then $\text{rank}(\mathbf{V}) < \text{rank}(\mathbf{W})$.

Example 3.3. Vector spaces $\mathbf{V}$ included in $\mathbb{Q}^2$ satisfy $\text{rank}(\mathbf{V}) \in \{0,1,2\}$. Moreover these vectors spaces can be classified as follows : $\text{rank}(\mathbf{V}) = 0$ if and only if $\mathbf{V} = \{\mathbf{0}\}$, $\text{rank}(\mathbf{V}) = 1$ if and only if $\mathbf{V} = \mathbb{Q}\mathbf{v}$ with $\mathbf{v} \in \mathbb{Q}^2 \backslash \{\mathbf{0}\}$, and $\text{rank}(\mathbf{V}) = 2$ if and only if $\mathbf{V} = \mathbb{Q}^2$.

The *(topological) closure* of a set $\mathbf{X} \subseteq \mathbb{Q}^d$ is the set $\overline{\mathbf{X}}$ of vectors $\mathbf{r} \in \mathbb{Q}^d$ such that for every $\epsilon \in \mathbb{Q}_{>0}$ there exists $\mathbf{x} \in \mathbf{X}$ satisfying $||\mathbf{r} - \mathbf{x}||_\infty < \epsilon$. A set $\mathbf{X}$ is said to be *closed* if $\overline{\mathbf{X}} = \mathbf{X}$. Note that $\overline{\mathbf{X}}$ is closed and this set is the minimal for inclusion among the closed sets that contain $\mathbf{X}$. Let us recall that a vector space $\mathbf{V}$ is closed and the closure of a conic set is a conic set. Since the classical topological interior of a conic set $\mathbf{C}$ is empty when the vector space generated by $\mathbf{C}$ is not equal to $\mathbb{Q}^d$ (the conic set is *degenerated*), we introduce the notion of interior of $\mathbf{C}$ relatively to the vector space $\mathbf{V} = \mathbf{C} - \mathbf{C}$. More precisely, a

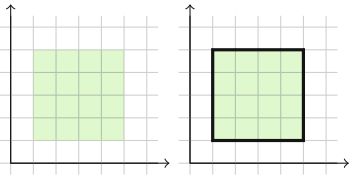


Fig. 2. Sets $\mathbf{X} = (1,5) \times (1,5)$ and $\overline{\mathbf{X}} = [1,5] \times [1,5]$

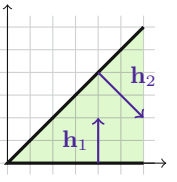


Fig. 3. A picture of the duality lemma 3.5

vector $\mathbf{c} \in \mathbf{C}$ is said to be in the *interior* of $\mathbf{C}$ if there exists $\epsilon \in \mathbb{Q}_{>0}$ such that $\mathbf{c}+\mathbf{v} \in \mathbf{C}$ for every $\mathbf{v} \in \mathbf{C}-\mathbf{C}$ satisfying $||\mathbf{v}||_\infty < \epsilon$. We denote by $\text{int}(\mathbf{C})$ the set of *interior vectors* of $\mathbf{C}$. Let us recall that $\text{int}(\mathbf{C})$ is non empty for every conic set $\mathbf{C}$, and $\overline{\mathbf{C}_1} = \overline{\mathbf{C}_2}$ if and only if $\text{int}(\mathbf{C}_1) = \text{int}(\mathbf{C}_2)$ for every conic sets $\mathbf{C}_1, \mathbf{C}_2$.

Example 3.4. Let $\mathbf{X} = (1,5) \times (1,5)$. Then $\overline{\mathbf{X}} = [1,5] \times [1,5]$ (see Fig. 2).

The following lemma characterizes the finitely generated cones.

Lemma 3.5 (Duality). *Let $\mathbf{V} \subseteq \mathbb{Q}^d$ be a vector space. A conic set $\mathbf{C} \subseteq \mathbf{V}$ is finitely generated if and only if there exists a sequence $(\mathbf{h}_j)_{1 \leq j \leq k}$ of vectors $\mathbf{h}_j \in \mathbf{V} \backslash \{\mathbf{0}\}$ such that:*

$$\mathbf{C} = \bigcap_{j=1}^{k} \left\{ \mathbf{v} \in \mathbf{V} \mid \sum_{i=1}^{d} \mathbf{h}_j(i)\mathbf{v}(i) \geq 0 \right\}$$

Moreover in this case the following equality holds if and only if $\mathbf{V}$ is the vector space generated by $\mathbf{C}$:

$$\text{int}(\mathbf{C}) = \bigcap_{j=1}^{k} \left\{ \mathbf{v} \in \mathbf{V} \mid \sum_{i=1}^{d} \mathbf{h}_j(i)\mathbf{v}(i) > 0 \right\}$$

Proof. This is a classical result of duality [10]. □

Example 3.6. Let us introduce the whole vector space $\mathbf{V} = \mathbb{Q}^2$ and the finitely generated conic set $\mathbf{C} = \mathbb{Q}_{\geq 0}(1,1) + \mathbb{Q}_{\geq 0}(1,0)$. Fig. 3 shows that $\mathbf{C} = \bigcap_{j \in \{1,2\}} \{\mathbf{v} \in \mathbf{V} \mid \sum_{i=1}^{d} \mathbf{h}_j(i)\mathbf{v}(i) \geq 0\}$ where $\mathbf{h}_1 = (0,2)$ and $\mathbf{h}_2 = (2,-2)$.

Lemma 3.7. *The topological closure of a set definable in* $\text{FO}(\mathbb{Q}, +, \leq, 0)$ *is a finite union of finitely generated conic sets.*

Proof. Let $\mathbf{X} \subseteq \mathbb{Q}^d$ be a set definable in FO $(\mathbb{Q}, +, \leq, 0)$. Since this logic admits quantification elimination we deduce that there exists a quantifier free formula in this logic that denotes $\mathbf{X}$. Hence there exists a finite sequence $(A_j)_{1 \leq j \leq k}$ of finite sets $A_j \subseteq \mathbb{Q}^d \times \{>, \geq\}$ such that $\mathbf{X} = \bigcup_{j=1}^{k} \mathbf{X}_j$ where:

$$\mathbf{X}_j = \bigcap_{(\mathbf{h},\#) \in A_j} \left\{ \mathbf{x} \in \mathbb{Q}^d \mid \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{x}(i) \# 0 \right\}$$

We can assume without loss of generality that $\mathbf{X}_j$ is non empty. Moreover if $k = 0$ the proof is immediate since $\overline{\mathbf{X}} = \emptyset$. So we can assume that $k \geq 1$. Let us introduce the following set $\mathbf{R}_j$:

$$\mathbf{R}_j = \bigcap_{(\mathbf{h},\#) \in A_j} \left\{ \mathbf{x} \in \mathbb{Q}^d \mid \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{x}(i) \geq 0 \right\}$$

Lemma 3.5 shows that $\mathbf{R}_j$ is finitely generated. Thanks to Lemma 3.5, we deduce that $\mathbf{R} = \bigcup_{j=1}^{k} \mathbf{R}_j$ is closed. We are going to prove that $\overline{\mathbf{X}} = \mathbf{R}$. Since $\mathbf{X}_j \subseteq \mathbf{R}_j$ we get $\mathbf{X} \subseteq \mathbf{R}$. As $\mathbf{R}$ is closed we deduce that $\overline{\mathbf{X}} \subseteq \mathbf{R}$. Let us prove the converse inclusion. Let $\mathbf{r} \in \mathbf{R}$. There exists $j \in \{1, \ldots, k\}$ such that $\mathbf{r} \in \mathbf{R}_j$. Since $\mathbf{X}_j$ is non empty, there exists $\mathbf{x}_j \in \mathbf{X}_j$. As $\mathbf{r}_j \in \mathbf{R}_j$ and $\mathbf{x}_j \in \mathbf{X}_j$ we deduce that $\mathbf{r}_j + \mathbb{Q}_{>0}\mathbf{x}_j \subseteq \mathbf{X}_j$. Hence $\mathbf{r}_j \in \overline{\mathbf{X}_j}$ and we have proved the other inclusion $\mathbf{R} \subseteq \overline{\mathbf{X}}$. Therefore $\overline{\mathbf{X}}$ is a finite union of finitely generated conic sets since it is equal to $\mathbf{R}$. □

Theorem 3.8. *A conic set $\mathbf{C} \subseteq \mathbb{Q}^d$ is definable if and only if the conic set $\overline{\mathbf{C} \cap \mathbf{V}}$ is finitely generated for every vector space $\mathbf{V} \subseteq \mathbb{Q}^d$.*

Proof. Let us first consider a definable conic set $\mathbf{C} \subseteq \mathbb{Q}^d$, let $\mathbf{V}$ be a vector space, and let us prove that $\overline{\mathbf{X}}$ is finitely generated where $\mathbf{X} = \mathbf{C} \cap \mathbf{V}$. Since $\mathbf{X}$ is definable in FO $(\mathbb{Q}, +, \leq, 0)$, Lemma 3.7 shows that $\overline{\mathbf{X}} = \bigcup_{j=1}^{k} \mathbf{C}_j$ where $\mathbf{C}_j$ is a finitely generated conic sets. Moreover, as $\mathbf{X}$ is non empty we deduce that $k \geq 1$. As $\overline{\mathbf{X}}$ is a conic set we deduce that $\sum_{j=1}^{k} \mathbf{C}_j \subseteq \overline{\mathbf{X}}$. Moreover, as $\mathbf{0} \in \mathbf{C}_j$ for every j, we deduce that $\mathbf{C}_j \subseteq \sum_{j=1}^{k} \mathbf{C}_j$ for every j. Thus $\overline{\mathbf{X}} = \sum_{j=1}^{k} \mathbf{C}_j$ and we have proved that $\overline{\mathbf{X}}$ is finitely generated.

Conversely, we prove by induction over r that the conic sets $\mathbf{C} \subseteq \mathbb{Q}^d$ such that $\text{rank}(\mathbf{C} - \mathbf{C}) \leq r$ and such that the conic set $\overline{\mathbf{C} \cap \mathbf{V}}$ is finitely generated for every vector space $\mathbf{V} \subseteq \mathbb{Q}^d$ are definable. The case $r = 0$ is immediate since in this case $\mathbf{C} = \{\mathbf{0}\}$. Let us assume the induction proved for an integer $r \in \mathbb{N}$ and let us consider a conic set $\mathbf{C} \subseteq \mathbb{Q}^d$ such that $\text{rank}(\mathbf{C} - \mathbf{C}) \leq r + 1$ and such that the conic set $\overline{\mathbf{C} \cap \mathbf{V}}$ is finitely generated for every vector space $\mathbf{V} \subseteq \mathbb{Q}^d$. We introduce the vector space $\mathbf{W} = \mathbf{C} - \mathbf{C}$. Since $\overline{\mathbf{C}} = \overline{\mathbf{C} \cap \mathbf{V}}$ with $\mathbf{V} = \mathbb{Q}^d$, we deduce that $\overline{\mathbf{C}}$ is finitely generated. Lemma 3.5 shows that there exists a finite sequence $(\mathbf{h}_j)_{1 \leq j \leq k}$ of vectors $\mathbf{h}_j \in \mathbf{W} \backslash \{\mathbf{0}\}$ such that the following equality holds:

$$\overline{\mathbf{C}} = \bigcap_{j=1}^{k} \left\{ \mathbf{x} \in \mathbf{W} \mid \sum_{i=1}^{d} \mathbf{h}_j(i)\mathbf{x}(i) \geq 0 \right\}$$

Since $\text{int}(\mathbf{C}) = \text{int}(\overline{\mathbf{C}})$ we get the following equality:

$$\text{int}(\mathbf{C}) = \bigcap_{j=1}^{k} \left\{ \mathbf{x} \in \mathbf{W} \mid \sum_{i=1}^{d} \mathbf{h}_j(i)\mathbf{x}(i) > 0 \right\}$$

In particular $\text{int}(\mathbf{C})$ is definable in $\text{FO}(\mathbb{Q}, +, \leq, 0, 1)$. As $\text{int}(\mathbf{C}) \subseteq \mathbf{C} \subseteq \overline{\mathbf{C}}$ we deduce the following decomposition where $\mathbf{W}_j = \{\mathbf{w} \in \mathbf{W} \mid \sum_{i=1}^{d} \mathbf{h}_j(i)\mathbf{w}(i) = 0\}$:

$$\mathbf{C} = \text{int}(\mathbf{C}) \cup \bigcup_{j=1}^{k} (\mathbf{C} \cap \mathbf{W}_j)$$

Observe that $\mathbf{h}_j \in \mathbf{W} \backslash \mathbf{W}_j$ and in particular $\mathbf{W}_j$ is strictly included in $\mathbf{W}$. Thus $\text{rank}(\mathbf{W}_j) < \text{rank}(\mathbf{W}) \leq r + 1$. Note that $\mathbf{C}_j = \mathbf{C} \cap \mathbf{W}_j$ is a conic set such that $\text{rank}(\mathbf{C}_j - \mathbf{C}_j) \leq \text{rank}(\mathbf{W}_j) \leq r$ and such that $\overline{\mathbf{C}_j \cap \mathbf{V}}$ is a finitely generated conic set for every vector space $\mathbf{V}$. Thus by induction $\mathbf{C}_j$ is definable in $\text{FO}(\mathbb{Q}, +, \leq, 0, 1)$. We deduce that $\mathbf{C}$ is definable. We have proved the induction. □

Example 3.9. Observe that the conic set $\mathbf{C} = \{(c_1, c_2) \in \mathbb{Q}^2_{\geq 0} \mid \sqrt{2}c_2 \leq c_1\}$ is not finitely generated. Let us consider $\mathbf{V} = \mathbb{Q}^2$ and observe that $\mathbf{C} \cap \mathbf{V} = \mathbf{C}$ and since $\overline{\mathbf{C}} = \mathbf{C}$ we deduce that $\overline{\mathbf{C} \cap \mathbf{V}}$ is not finitely generated. Theorem 3.8 shows that $\mathbf{C}$ is not definable.

4 Presburger Sets and almost Semilinear Sets

In this section we introduce the *Presburger* sets and the *almost semilinear* sets.

A *periodic set* is a subset $\mathbf{P} \subseteq \mathbb{Z}^d$ such that $\mathbf{0} \in \mathbf{P}$ and such that $\mathbf{P} + \mathbf{P} \subseteq \mathbf{P}$. A periodic set $\mathbf{P}$ is said to be *finitely generated* if there exists a finite sequence $\mathbf{p}_1, \ldots, \mathbf{p}_k$ of vectors $\mathbf{p}_j \in \mathbf{P}$ such that $\mathbf{P} = \mathbb{N}\mathbf{p}_1 + \cdots + \mathbb{N}\mathbf{p}_k$ (see Fig. 4). A subset $\mathbf{S} \subseteq \mathbb{Z}^d$ is called a *Presburger set* if it can be denoted by a formula in the Presburger arithmetic $\text{FO}(\mathbb{Z}, +, \leq, 0, 1)$. Let us recall [2] that a subset $\mathbf{S} \subseteq \mathbb{Z}^d$ is Presburger if and only if it is *semilinear*, i.e. a finite union of sets $\mathbf{b} + \mathbf{P}$ where $\mathbf{b} \in \mathbb{Z}^d$ and $\mathbf{P} \subseteq \mathbb{Z}^d$ is a finitely generated periodic set. The class of almost semilinear sets is obtained by weakening the finiteness property of the periodic sets $\mathbf{P}$.

Definition 4.1. *A periodic set* $\mathbf{P}$ *is said to be* asymptotically definable *if the conic set* $\mathbb{Q}_{\geq 0}\mathbf{P}$ *is definable.*

Remark 4.2. Every finitely generated periodic set $\mathbf{P}$ is asymptotically definable since in this case $\mathbb{Q}_{\geq 0}\mathbf{P}$ is a finitely generated conic set and in particular a definable conic set.

Example 4.3. The periodic set $\mathbf{P} = \{(p_1, p_2) \in \mathbb{N}^2 \mid \sqrt{2}p_2 \leq p_1\}$ is not asymptotically definable since $\mathbb{Q}_{\geq 0}\mathbf{P} = \{(c_1, c_2) \in \mathbb{N}^2 \mid \sqrt{2}c_2 \leq c_1\}$ is not definable (see example 3.9).

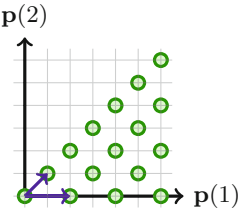


Fig. 4. The finitely generated periodic set $\mathbf{P} = \mathbb{N}(1,1) + \mathbb{N}(2,0)$

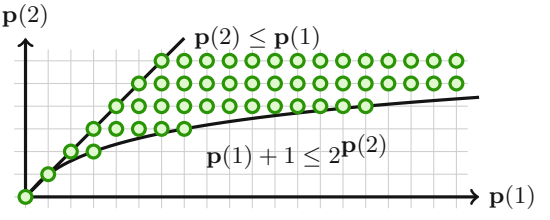


Fig. 5. An asymptotically definable periodic set

Example 4.4. The periodic set $\mathbf{P} = \{\mathbf{p} \in \mathbb{N}^2 \mid \mathbf{p}(2) \leq \mathbf{p}(1) \leq 2^{\mathbf{p}(2)} - 1\}$ is represented in Figure 5. Observe that $\mathbb{Q}_{\geq 0}\mathbf{P} = \{\mathbf{0}\} \cup \{\mathbf{c} \in \mathbb{Q}^2_{>0} \mid \mathbf{p}(2) \leq \mathbf{p}(1)\}$ is a definable conic set. Thus $\mathbf{P}$ is an asymptotically definable periodic set.

The following lemma shows that the class of asymptotically definable periodic sets is stable by finite intersections.

Lemma 4.5. *We have* $(\mathbb{Q}_{\geq 0}\mathbf{P}_1) \cap (\mathbb{Q}_{\geq 0}\mathbf{P}_2) = \mathbb{Q}_{\geq 0}(\mathbf{P}_1 \cap \mathbf{P}_2)$ *for every periodic sets* $\mathbf{P}_1, \mathbf{P}_2 \subseteq \mathbb{Z}^d$.

Proof. Observe that $\mathbf{P}_1 \subseteq \mathbb{Q}_{\geq 0}\mathbf{P}_1$ and $\mathbf{P}_2 \subseteq \mathbb{Q}_{\geq 0}\mathbf{P}_2$. Hence $\mathbf{P}_1 \cap \mathbf{P}_2 \subseteq \mathbf{C}$ where $\mathbf{C} = (\mathbb{Q}_{\geq 0}\mathbf{P}_1) \cap (\mathbb{Q}_{\geq 0}\mathbf{P}_2)$. As $\mathbf{C}$ is a conic set we deduce that $\mathbb{Q}_{\geq 0}(\mathbf{P}_1 \cap \mathbf{P}_2) \subseteq \mathbf{C}$. For the converse inclusion. Let $\mathbf{c} \in \mathbf{C}$. Since $\mathbf{c} \in \mathbb{Q}_{\geq 0}\mathbf{P}_1$, there exists $\lambda_1 \in \mathbb{Q}_{\geq 0}$ such that $\mathbf{c} \in \lambda_1\mathbf{P}_1$. Symmetrically there exists $\lambda_2 \in \mathbb{Q}_{\geq 0}$ such that $\mathbf{c} \in \lambda_2\mathbf{P}_2$. Let $n_1, n_2 \in \mathbb{N}_{>0}$ such that $n_1\lambda_1 \in \mathbb{N}$ and $n_2\lambda_2 \in \mathbb{N}$. Let $n = n_1n_2$ and observe that $n\mathbf{c} \in n_2(n_1\lambda_1)\mathbf{P}_1 \subseteq \mathbf{P}_1$ since $\mathbf{P}_1$ is a periodic set. Symmetrically $n\mathbf{c} \in \mathbf{P}_2$. We have proved that $n\mathbf{c} \in \mathbf{P}_1 \cap \mathbf{P}_2$. Thus $\mathbf{c} \in \mathbb{Q}_{\geq 0}(\mathbf{P}_1 \cap \mathbf{P}_2)$ and we get the other inclusion. □

Definition 4.6. *An* almost semilinear set *is a subset* $\mathbf{X} \subseteq \mathbb{Z}^d$ *such that for every Presburger set* $\mathbf{S} \subseteq \mathbb{Z}^d$ *the set* $\mathbf{X} \cap \mathbf{S}$ *is a finite union of sets* $\mathbf{b} + \mathbf{P}$ *where* $\mathbf{b} \in \mathbb{Z}^d$ *and* $\mathbf{P} \subseteq \mathbb{Z}^d$ *is an asymptotically definable periodic set.*

Example 4.7. Let us consider the periodic set $\mathbf{P} = \{(0,0)\} \cup \{(2^n, 1) \mid n \in \mathbb{N}\} \cup ((1,2) + \mathbb{N}^2)$ depicted in Fig 6. Observe that $\mathbb{Q}_{\geq 0}\mathbf{P}$ is the definable conic set $\{(0,0)\} \cup \mathbb{Q}_{\geq 0} \times \mathbb{Q}_{>0}$. Note that $\mathbf{P}$ is not almost semilinear since $\mathbf{P} \cap (\mathbb{N} \times \{1\}) = \{(2^n, 1) \mid n \in \mathbb{N}\}$ can not be decomposed as a finite union of sets $\mathbf{b} + \mathbf{P}$ where $\mathbf{b} \in \mathbb{Z}^d$ and $\mathbf{P} \subseteq \mathbb{Z}^d$ is an asymptotically definable periodic set.

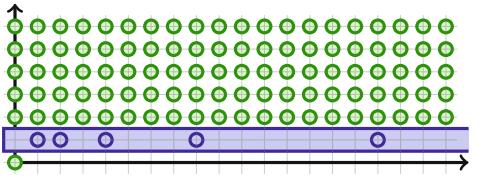


Fig. 6. An asymptotically definable periodic set that is not almost semilinear

The class of almost semilinear sets is included in the class of Presburger sets. The strict inclusion will be proved strict as a direct consequence of a stronger result proved in this paper. In fact the *reachability relation of a Vector Addition System is proved to be almost semilinear* and we know that in general such a relation is not Presburger.

5 Linearizations

The linearization of a periodic set $\mathbf{P} \subseteq \mathbb{Z}^d$ is the periodic set $\mathrm{lin}(\mathbf{P})$ defined by the following equality:

$$\mathrm{lin}(\mathbf{P}) = (\mathbf{P} - \mathbf{P}) \cap \overline{\mathbb{Q}_{\geq 0}\mathbf{P}}$$

Lemma 5.1. *The linearization of an asymptotically definable periodic set is finitely generated.*

Proof. Let $\mathbf{V}$ be the vector space generated by $\mathbf{P}$ and let us introduce the conic set $\mathbf{C} = \overline{\mathbb{Q}_{\geq 0}\mathbf{P}}$. Note that $\mathbb{Q}_{\geq 0}\mathbf{P} \subseteq \mathbf{V}$ and since $\mathbf{V}$ is closed we get $\mathbf{C} \subseteq \mathbf{V}$. As $\mathbb{Q}_{\geq 0}\mathbf{P}$ is a definable conic set we deduce that $\mathbf{C}$ is finitely generated. Hence there exists $\mathbf{c}_1, \ldots, \mathbf{c}_k \in \mathbf{C}$ such that $\mathbf{C} = \mathbb{Q}_{\geq 0}\mathbf{c}_1 + \cdots + \mathbb{Q}_{\geq 0}\mathbf{c}_k$. As $\mathbf{c}_j \in \mathbf{C} \subseteq \mathbf{V} = \mathbb{Q}_{\geq 0}\mathbf{P} - \mathbb{Q}_{\geq 0}\mathbf{P}$, by replacing $\mathbf{c}_j$ by a vector in $\mathbb{N}_{>0}\mathbf{c}_j$ we can assume that $\mathbf{c}_j \in \mathbf{P} - \mathbf{P}$ for every $j \in \{1, \ldots, k\}$.

We introduce the following set $\mathbf{R}$:

$$\mathbf{R} = \left\{ \mathbf{r} \in \mathbf{P} - \mathbf{P} \mid \mathbf{r} = \sum_{j=1}^{k} \lambda_j \mathbf{c}_j \quad \lambda_j \in \mathbb{Q} \quad 0 \leq \lambda_j < 1 \right\}$$

We observe that every vector $\mathbf{r} \in \mathbf{R}$ satisfies $||\mathbf{r}||_\infty \leq s$ where $s = \sum_{j=1}^{k} ||\mathbf{c}_j||_\infty$. Hence $\mathbf{R} \subseteq \{-s, \ldots, s\}^d$ and we deduce that $\mathbf{R}$ is finite.

Let $\mathbf{L}$ be the periodic set generated by the finite set $\mathbf{R} \cup \{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$. Since this finite set is included in $\mathrm{lin}(\mathbf{P})$ we deduce that $\mathbf{L} \subseteq \mathrm{lin}(\mathbf{P})$. Let us prove the converse inclusion. Let $\mathbf{x} \in \mathrm{lin}(\mathbf{P})$. Since $\mathbf{x} \in \mathbf{C}$, there exists a sequence $(\mu_j)_{1 \leq j \leq k}$ of rational elements $\mu_j \in \mathbb{Q}_{\geq 0}$ such that $\mathbf{x} = \sum_{j=1}^{k} \mu_j \mathbf{c}_j$. Let us introduce $n_j \in \mathbb{N}$ such that $\lambda_j = \mu_j - n_j$ satisfies $0 \leq \lambda_j < 1$. Let $\mathbf{r} = \sum_{j=1}^{k} \lambda_j \mathbf{c}_j$. As $\mathbf{r} = \mathbf{x} - \sum_{j=1}^{k} n_j \mathbf{c}_j$ we get $\mathbf{r} \in \mathbf{P} - \mathbf{P}$. Thus $\mathbf{r} \in \mathbf{R}$. From $\mathbf{x} = \mathbf{r} + \sum_{j=1}^{k} n_j \mathbf{c}_j$ we get $\mathbf{x} \in \mathbf{L}$. We have proved that $\mathrm{lin}(\mathbf{P})$ is the finitely generated periodic set $\mathbf{L}$. □

We observe that if the intersection $(\mathbf{b}_1+\mathbf{P}_1)\cap(\mathbf{b}_2+\mathbf{P}_2)$ is empty where $\mathbf{b}_1,\mathbf{b}_2\in\mathbb{Z}^d$ and $\mathbf{P}_1,\mathbf{P}_2\subseteq\mathbb{Z}^d$ are two asymptotically definable periodic sets then the intersection $(\mathbf{b}_1+\mathrm{lin}(\mathbf{P}_1))\cap(\mathbf{b}_2+\mathrm{lin}(\mathbf{P}_2))$ may be non empty (see Example 5.3). In this section we show that a dimension is strictly decreasing.

Let us first introduce our definition of dimension. The *dimension* $\dim(\mathbf{X})$ of a non-empty set $\mathbf{X}\subseteq\mathbb{Z}^d$ is the minimal integer $r\in\{0,\ldots,d\}$ such that there exists $k\in\mathbb{N}_{>0}$, a sequence $(\mathbf{b}_j)_{1\leq j\leq k}$ of vectors $\mathbf{b}_j\in\mathbb{Z}^d$, and a sequence $(\mathbf{V}_j)_{1\leq j\leq k}$ of vector spaces $\mathbf{V}_j\subseteq\mathbb{Q}^d$ such that $\mathrm{rank}(\mathbf{V}_j)\leq r$ and such that $\mathbf{X}\subseteq\bigcup_{j=1}^k \mathbf{b}_j+\mathbf{V}_j$. The dimension of the empty set is defined by $\dim(\emptyset)=-1$.

In the reminder of this section we prove the following Theorem 5.2. All the other results or definitions introduced in this section are not used in the sequel.

Theorem 5.2. *Let $\mathbf{b}_1,\mathbf{b}_2\in\mathbb{Z}^d$ and let $\mathbf{P}_1,\mathbf{P}_2$ be two asymptotically definable periodic sets such that the intersection $(\mathbf{b}_1+\mathbf{P}_1)\cap(\mathbf{b}_2+\mathbf{P}_2)$ is empty. The intersection $\mathbf{X}=(\mathbf{b}_1+\mathrm{lin}(\mathbf{P}_1))\cap(\mathbf{b}_2+\mathrm{lin}(\mathbf{P}_2))$ satisfies:*

$$\dim(\mathbf{X})<\max\{\dim(\mathbf{b}_1+\mathbf{P}_1),\dim(\mathbf{b}_2+\mathbf{P}_2)\}$$

Example 5.3. Sets introduced in this example are depicted in Fig. 7. Let us introduce the asymptotically definable periodic sets $\mathbf{P}_1=\{\mathbf{p}\in\mathbb{N}^2\mid\mathbf{p}(2)\leq\mathbf{p}(1)\leq 2^{\mathbf{p}(2)}-1\}$ and $\mathbf{P}_2=\mathbb{N}(1,0)+\mathbb{N}(3,-1)$. We consider $\mathbf{b}_1=(0,0)$ and $\mathbf{b}_2=(7,2)$. We observe that the intersection of $\mathbf{b}_1+\mathbf{P}_1$ and $\mathbf{b}_2+\mathbf{P}_2$ is empty. Note that the intersection $\mathbf{X}$ of $\mathbf{b}_1+\mathrm{lin}(\mathbf{P}_1)$ and $\mathbf{b}_2+\mathrm{lin}(\mathbf{P}_2)$ satisfies $\mathbf{X}=\{(7,2),(10,1),(13,0)\}+\mathbb{N}(1,0)$. In particular we have $\dim(\mathbf{X})=1$ whereas $\dim(\mathbf{b}_1+\mathrm{lin}(\mathbf{P}_1))=\dim(\mathbf{b}_2+\mathrm{lin}(\mathbf{P}_2))=2$.

We first characterize the dimension of a periodic set.

Lemma 5.4. *Let $\mathbf{V}$ be the vector space generated by a periodic set $\mathbf{P}$. Then* $\mathrm{rank}(\mathbf{V})=\dim(\mathbf{P})$.

Proof. Let $\mathbf{P}$ be a periodic set and let us first prove by induction over $k\in\mathbb{N}_{>0}$ that for every sequence $(\mathbf{V}_j)_{1\leq j\leq k}$ of vector spaces $\mathbf{V}_j\subseteq\mathbb{Q}^d$, the inclusion $\mathbf{P}\subseteq\bigcup_{j=1}^k\mathbf{V}_j$ implies that there exists $j\in\{1,\ldots,k\}$ such that $\mathbf{P}\subseteq\mathbf{V}_j$. The case $k=1$ is immediate. Assume the property proved for an integer $k\in\mathbb{N}_{>0}$

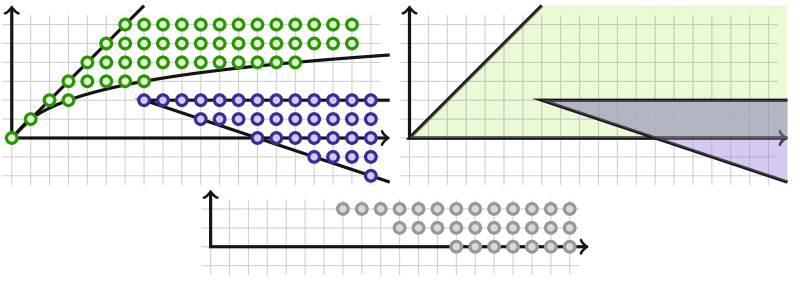


Fig. 7. A figure for Theorem 5.2 and Example 5.3

and let us assume that $\mathbf{P} \subseteq \bigcup_{j=1}^{k+1} \mathbf{V}_j$. If $\mathbf{P} \subseteq \mathbf{V}_{k+1}$ the property is proved. So we can assume that there exists $\mathbf{p} \in \mathbf{P} \backslash \mathbf{V}_{k+1}$. Let us prove that $\mathbf{P} \subseteq \bigcup_{j=1}^{k} \mathbf{V}_j$. We consider $\mathbf{x} \in \mathbf{P}$. Observe that if $\mathbf{x} \notin \mathbf{V}_{k+1}$ then $\mathbf{x} \in \bigcup_{j=1}^{k} \mathbf{V}_j$. So we can assume that $\mathbf{x} \in \mathbf{V}_{k+1}$. We observe that $\mathbf{p} + n\mathbf{x} \in \mathbf{P}$ for every $n \in \mathbb{N}$ since the set $\mathbf{P}$ is periodic. We deduce that there exists $j \in \{1, \ldots, k+1\}$ such that $\mathbf{p} + n\mathbf{x} \in \mathbf{V}_j$. Naturally this integer j depends on n. However, since $\{1, \ldots, k+1\}$ is finite whereas $\mathbb{N}$ is infinite, there exists $j \in \{1, \ldots, k+1\}$ and $n < n'$ in $\mathbb{N}$ such that $\mathbf{p} + n\mathbf{x}$ and $\mathbf{p} + n'\mathbf{x}$ are both in $\mathbf{V}_j$. As $\mathbf{V}_j$ is a vector space, we deduce that $n'(\mathbf{p} + n\mathbf{x}) - n(\mathbf{p} + n'\mathbf{x})$ is in $\mathbf{V}_j$. Hence $\mathbf{p} \in \mathbf{V}_j$. As $\mathbf{p} \notin \mathbf{V}_{k+1}$ we deduce that $j \neq k+1$. As $\mathbf{V}_j$ is a vector space we deduce that $(\mathbf{p} + n'\mathbf{x}) - (\mathbf{p} + n\mathbf{x}) \in \mathbf{V}_j$. Hence $\mathbf{x} \in \mathbf{V}_j$. We have proved that $\mathbf{x} \in \bigcup_{j=1}^{k} \mathbf{V}_j$. Thus $\mathbf{P} \subseteq \bigcup_{j=1}^{k} \mathbf{V}_j$ and by induction there exists $j \in \{1, \ldots, k\}$ such that $\mathbf{P} \subseteq \mathbf{V}_j$. We have proved the induction.

Now, let us prove the lemma. We consider a periodic set $\mathbf{P}$ and we let $\mathbf{V}$ be the vector space generated by this set. Since $\mathbf{P} \subseteq \mathbf{V}$ we deduce that $\dim(\mathbf{P}) \leq \mathrm{rank}(\mathbf{V})$. For the converse inclusion, since $\mathbf{P}$ is non empty we deduce that $\mathbf{P} \subseteq \bigcup_{j=1}^{k} \mathbf{b}_j + \mathbf{V}_j$ where $k \in \mathbb{N}_{>0}$, $\mathbf{b}_j \in \mathbb{Z}^d$ and $\mathbf{V}_j \subseteq \mathbb{Q}^d$ is a vector space such that $\mathrm{rank}(\mathbf{V}_j) \leq \dim(\mathbf{P})$. Let us consider the set $J = \{j \in \{1, \ldots, k\} \mid \mathbf{b}_j \in \mathbf{V}_j\}$ and let us prove that $\mathbf{P} \subseteq \bigcup_{j \in J} \mathbf{V}_j$. Let $\mathbf{p} \in \mathbf{P}$ and $n \in \mathbb{N}$. Since $n\mathbf{p} \in \mathbf{P}$ there exists $j \in \{1, \ldots, k\}$ such that $n\mathbf{p} \in \mathbf{b}_j + \mathbf{V}_j$. Hence there exists $j \in \{1, \ldots, k\}$ and $n < n'$ in $\mathbb{N}$ such that $n\mathbf{p}$ and $n'\mathbf{p}$ are both in $\mathbf{b}_j + \mathbf{V}_j$. As $\mathbf{V}_j$ is a vector space we deduce that $n'\mathbf{p} - n\mathbf{p} \in \mathbf{V}_j$. Thus $\mathbf{p} \in \mathbf{V}_j$. Moreover as $\mathbf{b}_j \in n\mathbf{p} - \mathbf{V}_j \subseteq \mathbf{V}_j$ we deduce that $j \in J$. We have prove the inclusion $\mathbf{P} \subseteq \bigcup_{j \in J} \mathbf{V}_j$. From the previous paragraph we deduce that there exists $j \in J$ such that $\mathbf{P} \subseteq \mathbf{V}_j$. By minimality of the vector space generated by $\mathbf{P}$ we get $\mathbf{V} \subseteq \mathbf{V}_j$. Hence $\mathrm{rank}(\mathbf{V}) \leq \mathrm{rank}(\mathbf{V}_j)$. Since $\mathrm{rank}(\mathbf{V}_j) \leq \dim(\mathbf{P})$ we have proved the inequality $\mathrm{rank}(\mathbf{V}) \leq \dim(\mathbf{P})$. □

Next we prove a separation property.

Lemma 5.5. *Let $\mathbf{C}_{\leq}$ and $\mathbf{C}_{\geq}$ be two finitely generated conic sets that generates the same vector space $\mathbf{V}$ and such that the vector space generated by $\mathbf{C}_{\leq} \cap \mathbf{C}_{\geq}$ is strictly included in $\mathbf{V}$. Then there exists a vector $\mathbf{h} \in \mathbf{V} \backslash \{\mathbf{0}\}$ such that for every $\# \in \{\leq, \geq\}$, we have:*

$$\mathbf{C}_{\#} \subseteq \left\{ \mathbf{v} \in \mathbf{V} \mid \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{v}(i) \# 0 \right\}$$

Proof. Lemma 3.5 shows that there exists two finite sets $\mathbf{H}_{\leq}, \mathbf{H}_{\geq}$ included in $\mathbf{V} \backslash \{\mathbf{0}\}$ such that:

$$\mathbf{C}_{\#} = \bigcap_{\mathbf{h} \in \mathbf{H}_{\#}} \left\{ \mathbf{v} \in \mathbf{V} \mid \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{v}(i) \geq 0 \right\}$$

$$\mathrm{int}(\mathbf{C}_{\#}) = \bigcap_{\mathbf{h} \in \mathbf{H}_{\#}} \left\{ \mathbf{v} \in \mathbf{V} \mid \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{v}(i) > 0 \right\}$$

Assume by contradiction that the intersection $\text{int}(\mathbf{C}_{\leq}) \cap \text{int}(\mathbf{C}_{\geq})$ is non empty and let $\mathbf{c}$ be a vector in this set. Observe that there exists $\epsilon \in \mathbb{Q}_{>0}$ such that $\mathbf{c} + \mathbf{v} \in \mathbf{C}_{\leq} \cap \mathbf{C}_{\geq}$ for every $\mathbf{v} \in \mathbf{V}$ such that $||\mathbf{v}||_\infty < \epsilon$. We deduce that the vector space generated by $\mathbf{C}_{\leq} \cap \mathbf{C}_{\geq}$ contains $\mathbf{V}$ and we get a contradiction.

We deduce that the following intersection is empty where $\mathbf{H} = \mathbf{H}_{\leq} \cup \mathbf{H}_{\geq}$

$$\bigcap_{\mathbf{h} \in \mathbf{H}} \left\{ \mathbf{v} \in \mathbf{V} \mid \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{v}(i) > 0 \right\}$$

Farkas's Lemma [10] shows that there exists a non-zero function $f : \mathbf{H} \to \mathbb{Q}_{\geq 0}$ such that $\sum_{\mathbf{h} \in \mathbf{H}} f(\mathbf{h})\mathbf{h} = \mathbf{0}$. Let us introduce $\mathbf{a} = \sum_{\mathbf{h} \in \mathbf{H}_{\geq}} f(\mathbf{h})\mathbf{h}$ and $\mathbf{b} = \sum_{\mathbf{h} \in \mathbf{H} \setminus \mathbf{H}_{\geq}} f(\mathbf{h})\mathbf{h}$. Assume by contradiction that $\mathbf{a} = \mathbf{0}$. Since $\mathbf{a}+\mathbf{b} = \mathbf{0}$ we deduce that $\mathbf{b} = \mathbf{0}$. As f is not the zero function, there exists $\mathbf{h} \in \mathbf{H}$ such that $f(\mathbf{h}) \neq 0$. Note that either $\mathbf{h} \in \mathbf{H}_{\geq}$ or $\mathbf{h} \in \mathbf{H} \setminus \mathbf{H}_{\geq}$. In the first case we deduce that $\text{int}(\mathbf{C}_{\geq})$ is empty and in the second case we deduce that $\text{int}(\mathbf{C}_{\leq})$ is empty. Since both cases are impossible we get a contradiction. Thus $\mathbf{a} \neq \mathbf{0}$. For every $\mathbf{c} \in \text{int}(\mathbf{C}_{\geq})$ we have $\sum_{i=1}^{d} \mathbf{a}(i)\mathbf{c}(i) \geq 0$. Since the set $\{\mathbf{c} \in \mathbb{Q}^d \mid \sum_{i=1}^{d} \mathbf{a}(i)\mathbf{c}(i) \geq 0\}$ is closed we deduce that for every $\mathbf{c} \in \overline{\text{int}(\mathbf{C}_{\geq})} = \mathbf{C}_{\geq}$ the same inequality holds. Now let us consider $\mathbf{c} \in \text{int}(\mathbf{C}_{\leq})$. In this case $\sum_{i=1}^{d} \mathbf{b}(i)\mathbf{c}(i) \geq 0$. Since $\mathbf{a} + \mathbf{b} = \mathbf{0}$ we get $\sum_{i=1}^{d} \mathbf{a}(i)\mathbf{c}(i) \leq 0$. We deduce that this inequality holds for every $\mathbf{c} \in \mathbf{C}_{\leq}$. □

Remark 5.6. The previous Lemma 5.5 is wrong if we remove the finitely generated condition on the conic sets $\mathbf{C}_{\leq}$ and $\mathbf{C}_{\geq}$. In fact let us consider the conic sets $\mathbf{C}_{\leq} = \{\mathbf{x} \in \mathbb{Q}^2_{\geq 0} \mid \mathbf{x}(1) \leq \sqrt{2}\mathbf{x}(2)\}$ and $\mathbf{C}_{\geq} = \{\mathbf{x} \in \mathbb{Q}^2_{\geq 0} \mid \mathbf{x}(2) \geq \sqrt{2}\mathbf{x}(2)\}$. Observe that $\mathbf{C}_{\leq} \cap \mathbf{C}_{\geq} = \{\mathbf{0}\}$. Hence the vector space generated by the intersection is strictly included in $\mathbb{Q}^2$. However there does not exist a vector $\mathbf{h} \in \mathbb{Q}^2 \setminus \{\mathbf{0}\}$ satisfying the separation property required by Lemma 5.5. This problem can be overcome by introducing the vector spaces of $\mathbb{R}^d$. We do not introduce this extension to simplify the presentation.

We can now provide a proof for Theorem 5.2. We consider two vectors $\mathbf{b}_1, \mathbf{b}_2 \in \mathbb{Z}^d$ and two periodic sets $\mathbf{P}_1, \mathbf{P}_2 \subseteq \mathbb{Z}^d$ such that $(\mathbf{b}_1 + \mathbf{P}_1) \cap (\mathbf{b}_2 + \mathbf{P}_2) = \emptyset$. We introduce the intersection $\mathbf{X} = (\mathbf{b}_1 + \text{lin}(\mathbf{P}_1)) \cap (\mathbf{b}_2 + \text{lin}(\mathbf{P}_2))$. Observe that if $\mathbf{X}$ is empty the theorem is proved. So we can assume that there exists a vector $\mathbf{b}$ in this intersection. Let us denote by $\mathbf{V}_1$ and $\mathbf{V}_2$ the vector spaces generated by $\mathbf{P}_1$ and $\mathbf{P}_2$. Lemma 5.4 shows that $\text{rank}(\mathbf{V}_j) = \dim(\mathbf{P}_j)$ and from $\dim(\mathbf{b}_j + \mathbf{P}_j) = \dim(\mathbf{P}_j)$ we deduce that $\dim(\mathbf{b}_j + \mathbf{P}_j) = \text{rank}(\mathbf{V}_j)$. As $\mathbf{X}$ is included in $\mathbf{b}+\mathbf{V}$ where $\mathbf{V} = \mathbf{V}_1 \cap \mathbf{V}_2$, we deduce that if $\mathbf{V}$ is strictly included in $\mathbf{V}_j$ for one $j \in \{1, 2\}$ then $\dim(\mathbf{X}) \leq \text{rank}(\mathbf{V}) < \text{rank}(\mathbf{V}_j) = \dim(\mathbf{b}_j + \mathbf{P}_j)$ and the theorem is proved. So we can assume that $\mathbf{V}_1 = \mathbf{V}_2 = \mathbf{V}$. Let us consider the conic sets $\mathbf{C}_1 = \overline{\mathbb{Q}_{\geq 0}\mathbf{P}_1}$ and $\mathbf{C}_2 = \overline{\mathbb{Q}_{\geq 0}\mathbf{P}_2}$. Since $\mathbf{P}_1$ and $\mathbf{P}_2$ are asymptotically definable periodic sets, we deduce that $\mathbf{C}_1$ and $\mathbf{C}_2$ are finitely generated conic sets. Note that $\mathbf{C}_1, \mathbf{C}_2 \subseteq \mathbf{V}$. We introduce the intersection $\mathbf{C} = \mathbf{C}_1 \cap \mathbf{C}_2$.

Assume by contradiction that the vector space generated by $\mathbf{C}$ is equal to $\mathbf{V}$. Let us consider a vector $\mathbf{c}$ in the interior of $\mathbf{C}$. The characterization given by Lemma 3.5 shows that in this case $\text{int}(\mathbf{C}) = \text{int}(\mathbf{C}_1) \cap \text{int}(\mathbf{C}_2)$. Since $\text{int}(\mathbf{C}_j) =$

$\text{int}(\mathbb{Q}_{\geq 0}\mathbf{P}_j)$ we deduce that $\mathbf{c} \in (\mathbb{Q}_{\geq 0}\mathbf{P}_1) \cap (\mathbb{Q}_{\geq 0}\mathbf{P}_2)$. Lemma 4.5 shows that $\mathbf{c} \in \mathbb{Q}_{\geq 0}(\mathbf{P}_1 \cap \mathbf{P}_2)$. By replacing $\mathbf{c}$ be a vector in $\mathbb{N}_{>0}\mathbf{c}$ we can assume that $\mathbf{c} \in \mathbf{P}_1 \cap \mathbf{P}_2$.

Let us prove that there exists $k_1 \in \mathbb{N}$ such that $\mathbf{b} + k_1\mathbf{c} \in \mathbf{b}_1 + \mathbf{P}_1$. From $\mathbf{b} \in \mathbf{b}_1 + \text{lin}(\mathbf{P}_1)$ we deduce that there exists $\mathbf{p}_1, \mathbf{p}'_1 \in \mathbf{P}_1$ such that $\mathbf{b} = \mathbf{b}_1 + \mathbf{p}_1 - \mathbf{p}'_1$. Since $-\mathbf{p}'_1$ is in the vector space generated by $\mathbf{C}$ and $\mathbf{c}$ is in the interior of $\mathbf{C}$, there exists $n_1 \in \mathbb{N}$ large enough such that $n_1\mathbf{c} + (-\mathbf{p}'_1) \in \mathbf{C}_1$. Hence there exists $n'_1 \in \mathbb{N}_{>0}$ such that $n_1n'_1\mathbf{c} - n'_1\mathbf{p}'_1 \in \mathbf{P}_1$. Thus $n_1n'_1\mathbf{c} - \mathbf{p}'_1 \in (n'_1 - 1)\mathbf{p}'_1 + \mathbf{P}_1 \subseteq \mathbf{P}_1$. Hence $\mathbf{b} + k_1\mathbf{c} \in \mathbf{b}_1 + \mathbf{P}_1$ with $k_1 = n_1n'_1$.

Symmetrically we deduce that there exists $k_2 \in \mathbb{N}$ such that $\mathbf{b} + k_2\mathbf{c} \in \mathbf{b}_2 + \mathbf{P}_2$. We have proved that $\mathbf{b} + (k_1 + k_2)\mathbf{c} \in (\mathbf{b}_1 + \mathbf{P}_1) \cap (\mathbf{b}_2 + \mathbf{P}_2)$ and we get a contradiction since this intersection is supposed to be empty.

We deduce that the vector space generated by $\mathbf{C}$ is strictly included in $\mathbf{V}$. Lemma 5.5 shows that there exists a vector $\mathbf{h} \in \mathbf{V} \backslash \{\mathbf{0}\}$ such that:

$$\mathbf{C}_1 \subseteq \left\{ \mathbf{v} \in \mathbf{V} \mid \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{v}(i) \geq 0 \right\}$$

$$\mathbf{C}_2 \subseteq \left\{ \mathbf{v} \in \mathbf{V} \mid \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{v}(i) \leq 0 \right\}$$

By replacing $\mathbf{h}$ by a vector in $\mathbb{N}_{>0}\mathbf{h}$ we can assume that $\mathbf{h} \in \mathbb{Z}^d$. Now let us consider $\mathbf{x} \in \mathbf{X}$. Since $\mathbf{x} - \mathbf{b}_1 \in \mathbf{C}_1$ we deduce that $\sum_{i=1}^{d} \mathbf{h}(i)(\mathbf{x}(i) - \mathbf{b}_1(i)) \geq 0$ and since $\mathbf{x} - \mathbf{b}_2 \in \mathbf{C}_2$ we deduce that $\sum_{i=1}^{d} \mathbf{h}(i)(\mathbf{x}(i) - \mathbf{b}_2(i)) \leq 0$. We introduce the integers $z_1 = \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{b}_1(i)$ and $z_2 = \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{b}_2(i)$. We have proved that $\mathbf{X}$ can be decomposed into a finite union of slices $\mathbf{X} = \bigcup_{z=z_1}^{z_2} \mathbf{X}_z$ where:

$$\mathbf{X}_z = \left\{ \mathbf{x} \in \mathbf{X} \mid \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{x}(i) = z \right\}$$

Let us prove that $\dim(\mathbf{X}_z) < \text{rank}(\mathbf{V})$. If $\mathbf{X}_z$ is empty the relation is immediate. If $\mathbf{X}_z$ is non empty let us consider $\mathbf{x} \in \mathbf{X}_z$ and observe that $\mathbf{X}_z \subseteq \mathbf{x} + \mathbf{W}$ where:

$$\mathbf{W} = \left\{ \mathbf{v} \in \mathbf{V} \mid \sum_{i=1}^{d} \mathbf{h}(i)\mathbf{v}(i) = 0 \right\}$$

Note that $\mathbf{h} \in \mathbf{V} \backslash \mathbf{W}$. We deduce that $\mathbf{W}$ is strictly included in $\mathbf{V}$ and in particular $\text{rank}(\mathbf{W}) < \text{rank}(\mathbf{V})$. Hence $\dim(\mathbf{X}_z) < \text{rank}(\mathbf{V})$.

From $\mathbf{X} = \bigcup_{z=z_1}^{z_2} \mathbf{X}_z$ and $\dim(\mathbf{X}_z) < \text{rank}(\mathbf{V})$ for every z, we deduce that $\dim(\mathbf{X}) < \text{rank}(\mathbf{V})$ and the theorem is proved.

6 Presburger Invariants

Given a relation R over $\mathbb{Z}^d$ and two sets $\mathbf{X}, \mathbf{Y} \subseteq \mathbb{Z}^d$ we introduce the *forward image* $\text{post}_R(\mathbf{X})$ and the *backward image* $\text{pre}_R(\mathbf{Y})$ defined by the following equalities:

$$\begin{cases} \text{post}_R(\mathbf{X}) & = \bigcup_{\mathbf{x} \in \mathbf{X}} \{\mathbf{y} \in \mathbb{Z}^d \mid (\mathbf{x}, \mathbf{y}) \in R\} \\ \text{pre}_R(\mathbf{Y}) & = \bigcup_{\mathbf{y} \in \mathbf{Y}} \{\mathbf{x} \in \mathbb{Z}^d \mid (\mathbf{x}, \mathbf{y}) \in R\} \end{cases}$$

We say that a set $\mathbf{X} \subseteq \mathbb{Z}^d$ is a *forward invariant* for R if $\text{post}_R(\mathbf{X}) \subseteq \mathbf{X}$ and we say that a set $\mathbf{Y} \subseteq \mathbb{Z}^d$ is a *backward invariant* for R if $\text{pre}_R(\mathbf{Y}) \subseteq \mathbf{Y}$. In the reminder of this section we prove the following Theorem 6.1. All the other results or definitions introduced in this section are not used in the sequel.

Theorem 6.1. *Let R^* be a reflexive and transitive almost semilinear relation over $\mathbb{Z}^d$ and let $\mathbf{X}, \mathbf{Y} \subseteq \mathbb{Z}^d$ be two Presburger sets such that $R^* \cap (\mathbf{X} \times \mathbf{Y})$ is empty. There exists a partition of $\mathbb{Z}^d$ into a Presburger forward invariant that contains $\mathbf{X}$ and a Presburger backward invariant that contains $\mathbf{Y}$.*

We first prove the following lemma.

Lemma 6.2. *The sets $\text{post}_R(\mathbf{X})$ and $\text{pre}_R(\mathbf{Y})$ are almost semilinear for every almost semilinear relation $R \subseteq \mathbb{Z}^d \times \mathbb{Z}^d$ and for every Presburger sets $\mathbf{X}, \mathbf{Y} \subseteq \mathbb{Z}^d$*

Proof. Let us first prove that $\text{post}_R(\mathbf{X})$ is an almost semilinear set. We consider a Presburger set $\mathbf{S} \subseteq \mathbb{Z}^d$. Observe that $\mathbf{X} \times \mathbf{S}$ is a Presburger relation. Since R is an almost semilinear relation we deduce that $R \cap (\mathbf{X} \times \mathbf{S})$ can be decomposed into a finite union $\bigcup_{j=1}^{k} (\mathbf{a}_j, \mathbf{b}_j) + R_j$ with $k \in \mathbb{N}$, $(\mathbf{a}_j, \mathbf{b}_j) \in \mathbb{Z}^d \times \mathbb{Z}^d$ and R_j is an asymptotically definable periodic relation. We deduce that $\text{post}_R(\mathbf{X}) \cap \mathbf{S} = \bigcup_{j=1}^{k} \mathbf{b}_j + \mathbf{P}_j$ where $\mathbf{P}_j = \{\mathbf{v} \in \mathbb{Z}^d \mid \exists (\mathbf{u}, \mathbf{v}) \in R_j\}$. Since R_j is a periodic relation we deduce that $\mathbf{P}_j$ is a periodic set. Moreover since $\mathbb{Q}_{\geq 0} R_j$ is definable we deduce that $\mathbf{C}_j = \{\mathbf{v} \in \mathbb{Q}^d \mid \exists (\mathbf{u}, \mathbf{v}) \in \mathbb{Q}_{\geq 0} R_j\}$ is definable. Let us prove that $\mathbb{Q}_{\geq 0} \mathbf{P}_j = \mathbf{C}_j$. By construction we have $\mathbf{P}_j \subseteq \mathbf{C}_j$. Since $\mathbf{C}_j$ is conic we deduce that $\mathbb{Q}_{\geq 0} \mathbf{P}_j \subseteq \mathbf{C}_j$. For the converse inclusion let $\mathbf{v} \in \mathbf{C}_j$. There exists $\mathbf{u} \in \mathbb{Q}^d$ such that $(\mathbf{u}, \mathbf{v}) \in \mathbb{Q}_{\geq 0} R_j$. Hence there exists $\lambda \in \mathbb{Q}_{\geq 0}$ such that $(\mathbf{u}, \mathbf{v}) \in \lambda \mathbf{R}_j$. Let us consider $n \in \mathbb{N}_{>0}$ such that $n\lambda_j \in \mathbb{N}$ and observe that $(n\mathbf{u}, n\mathbf{v}) \in (n\lambda) R_j \subseteq R_j$ since R_j is periodic. Thus $n\mathbf{v} \in \mathbf{P}_j$ and we have proved that $\mathbf{v} \in \mathbb{Q}_{\geq 0} \mathbf{P}_j$. Hence $\mathbb{Q}_{\geq 0} \mathbf{P}_j = \mathbf{C}_j$ is a definable conic set and we have proved that $\text{post}_R(\mathbf{X})$ is an almost semilinear set. From $\text{pre}_R(\mathbf{Y}) = \text{post}_{R^{-1}}(\mathbf{Y})$ with $R^{-1} = \{(\mathbf{y}, \mathbf{x}) \mid (\mathbf{x}, \mathbf{y}) \in R\}$ we deduce that $\text{pre}_R(\mathbf{Y})$ is an almost semilinear set. □

Now, let us prove Theorem 6.1. We consider a reflexive and transitive almost semilinear relation R^*. We introduce the notion of *separators*. A *separator* is a couple $(\mathbf{X}, \mathbf{Y})$ of Presburger sets such that the intersection $R^* \cap (\mathbf{X} \times \mathbf{Y})$ is empty. Since R^* is reflexive, the intersection $\mathbf{X} \cap \mathbf{Y}$ is empty. The Presburger set $\mathbf{D} = \mathbb{Z}^d \setminus (\mathbf{X} \cup \mathbf{Y})$ is called the *domain* of $(\mathbf{X}, \mathbf{Y})$. We observe that a separator $(\mathbf{X}, \mathbf{Y})$ with an empty domain is a partition of $\mathbb{Z}^d$ such that $\mathbf{X}$ is a Presburger forward invariant and $\mathbf{Y}$ is a Presburger backward invariant. In particular Theorem 6.1 is obtained thanks to the following Lemma 6.3 with an immediate induction.

Lemma 6.3. *Let $(\mathbf{X}_0, \mathbf{Y}_0)$ be a separator with a non-empty domain $\mathbf{D}_0$. There exists a separator $(\mathbf{X}, \mathbf{Y})$ with a domain $\mathbf{D}$ such that $\mathbf{X}_0 \subseteq \mathbf{X}$, $\mathbf{Y}_0 \subseteq \mathbf{Y}$ and $\dim(\mathbf{D}) < \dim(\mathbf{D}_0)$.*

Proof. We first observe that a couple $(\mathbf{X}, \mathbf{Y})$ of Presburger sets is a separator if and only if $\text{post}_{R^*}(\mathbf{X}) \cap \text{pre}_{R^*}(\mathbf{Y}) = \emptyset$ if and only if $\text{post}_{R^*}(\mathbf{X}) \cap \mathbf{Y} = \emptyset$ if and only if $\text{pre}_{R^*}(\mathbf{Y}) \cap \mathbf{X} = \emptyset$.

Since R^* is an almost semilinear relation we deduce that $\text{post}_{R^*}(\mathbf{X}_0)$ is an almost semilinear set. As $\mathbf{D}_0$ is a Presburger set, we deduce that $\text{post}_{R^*}(\mathbf{X}_0) \cap \mathbf{D}_0 = \bigcup_{j=1}^{k} \mathbf{b}_j + \mathbf{P}_j$ where $\mathbf{b}_j \in \mathbb{Z}^d$ and $\mathbf{P}_j \subseteq \mathbb{Z}^d$ is an asymptotically definable periodic set. We introduce the following Presburger set:

$$\mathbf{S} = \bigcup_{j=1}^{k} \mathbf{b}_j + \text{lin}(\mathbf{P}_j)$$

Observe that $\text{post}_{R^*}(\mathbf{X}_0) \cap \mathbf{D}_0 \subseteq \mathbf{S}$. We deduce that the set $\mathbf{Y} = \mathbf{Y}_0 \cup (\mathbf{D}_0 \backslash \mathbf{S})$ is such that $\text{post}_{R^*}(\mathbf{X}_0) \cap \mathbf{Y} = \emptyset$. Hence $(\mathbf{X}_0, \mathbf{Y})$ is a separator.

Symmetrically, since R^* is an almost semilinear relation we deduce that $\text{pre}_{R^*}(\mathbf{Y})$ is an almost semilinear set. As $\mathbf{D}_0$ is a Presburger set, we deduce that $\text{pre}_{R^*}(\mathbf{Y}) \cap \mathbf{D}_0 = \bigcup_{l=1}^{n} \mathbf{c}_l + \mathbf{Q}_l$ where $\mathbf{c}_l \in \mathbb{Z}^d$ and $\mathbf{Q}_l \subseteq \mathbb{Z}^d$ is an asymptotically definable periodic set. We introduce the following Presburger set:

$$\mathbf{T} = \bigcup_{l=1}^{n} \mathbf{c}_l + \text{lin}(\mathbf{Q}_l)$$

Observe that $\text{pre}_{R^*}(\mathbf{Y}) \cap \mathbf{D}_0 \subseteq \mathbf{T}$. We deduce that the set $\mathbf{X} = \mathbf{X}_0 \cup (\mathbf{D}_0 \backslash \mathbf{T})$ is such that $\text{pre}_{R^*}(\mathbf{Y}) \cap \mathbf{X} = \emptyset$. Hence $(\mathbf{X}, \mathbf{Y})$ is a separator.

Let us introduce the domain $\mathbf{D}$ of $(\mathbf{X}, \mathbf{Y})$. We have the following equality where $\mathbf{Z}_{j,l} = (\mathbf{b}_j + \text{lin}(\mathbf{P}_j)) \cap (\mathbf{c}_l + \text{lin}(\mathbf{Q}_l))$:

$$\mathbf{D} = \mathbf{D}_0 \cap \Big(\bigcup_{\substack{1 \leq j \leq k \\ 1 \leq l \leq n}} \mathbf{Z}_{j,l} \Big)$$

As $(\mathbf{X}, \mathbf{Y})$ is a separator we deduce that $\text{post}_{R^*}(\mathbf{X}) \cap \text{pre}_{R^*}(\mathbf{Y})$ is empty. As $\mathbf{b}_j + \mathbf{P}_j \subseteq \text{post}_{R^*}(\mathbf{X}_0) \subseteq \text{post}_{R^*}(\mathbf{X})$ and $\mathbf{c}_l + \mathbf{Q}_l \subseteq \text{pre}_{R^*}(\mathbf{Y})$ we deduce that the intersection $(\mathbf{b}_j + \mathbf{P}_j) \cap (\mathbf{c}_l + \mathbf{Q}_l)$ is empty. Theorem 5.2 shows that $\dim(\mathbf{Z}_{j,l}) < \max\{\dim(\mathbf{b}_j + \mathbf{P}_j), \dim(\mathbf{c}_l + \mathbf{Q}_l)\}$. Since $\mathbf{b}_j + \mathbf{P}_j \subseteq \mathbf{D}_0$ and $\mathbf{c}_l + \mathbf{Q}_l \subseteq \mathbf{D}_0$ we deduce that $\dim(\mathbf{b}_j + \mathbf{P}_j) \leq \dim(\mathbf{D}_0)$ and $\dim(\mathbf{c}_l + \mathbf{Q}_l) \leq \dim(\mathbf{D}_0)$. We have proved that $\dim(\mathbf{D}) < \dim(\mathbf{D}_0)$. □

7 Vector Addition Systems

In this section we introduce the *Vector Addition Systems*, the *production relations* and a well order over the set of *runs* of Vector Addition Systems.

A *Vector Addition System (VAS)* is a finite subset $\mathbf{A} \subseteq \mathbb{Z}^d$. A *marking* is a vector $\mathbf{m} \in \mathbb{N}^d$. The semantics of vector addition systems is obtained by introducing for every word $w = \mathbf{a}_1 \dots \mathbf{a}_k$ of vectors $\mathbf{a}_j \in \mathbf{A}$ the relation $\xrightarrow{w}$ over the set of markings defined by $\mathbf{x} \xrightarrow{w} \mathbf{y}$ if there exists a word $\rho = \mathbf{m}_0 \dots \mathbf{m}_k$ of markings $\mathbf{m}_j \in \mathbb{N}^d$ such that $(\mathbf{x}, \mathbf{y}) = (\mathbf{m}_0, \mathbf{m}_k)$ and $\mathbf{m}_j = \mathbf{m}_{j-1} + \mathbf{a}_j$ for every $j \in \{1, \dots, k\}$. The word ρ is unique and it is called the *run* from $\mathbf{x}$ to $\mathbf{y}$ labeled by w. The marking $\mathbf{x}$ is called the *source* of ρ and it is denoted by $\text{src}(\rho)$, and

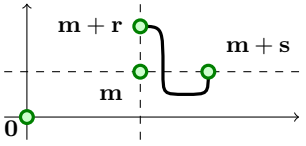


Fig. 8. The production relation of a marking $\mathbf{m}$

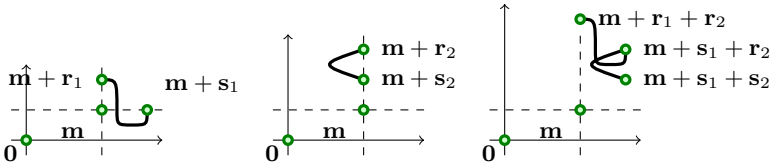


Fig. 9. Production relations are periodic

the marking $\mathbf{y}$ is called the *target* of ρ and it is denoted by $\mathrm{tgt}(\rho)$. The set of *runs* is denoted by Ω.

The *reachability relation* is the relation denoted by $\xrightarrow{*}$ over the set of markings defined by $\mathbf{x} \xrightarrow{*} \mathbf{y}$ if there exists a word $w \in \mathbf{A}^*$ such that $\mathbf{x} \xrightarrow{w} \mathbf{y}$. In the sequel we often used the fact that $\mathbf{x} \xrightarrow{w} \mathbf{y}$ implies $\mathbf{x} + \mathbf{v} \xrightarrow{w} \mathbf{y} + \mathbf{v}$ for every $\mathbf{v} \in \mathbb{N}^d$.

The *production relation* of a marking $\mathbf{m} \in \mathbb{N}^d$ (see Fig. 8) is the relation $\xrightarrow{*}_{\mathbf{m}}$ over $\mathbb{N}^d$ defined by $\mathbf{r} \xrightarrow{*}_{\mathbf{m}} \mathbf{s}$ if $\mathbf{m} + \mathbf{r} \xrightarrow{*} \mathbf{m} + \mathbf{s}$. The *production relation* of a run $\rho = \mathbf{m}_0 \dots \mathbf{m}_k$ is the relation $\xrightarrow{*}_{\rho}$ defined by the following composition:

$$\xrightarrow{*}_{\rho} = \xrightarrow{*}_{\mathbf{m}_0} \circ \cdots \circ \xrightarrow{*}_{\mathbf{m}_k}$$

Example 7.1. The production relation $\xrightarrow{*}_{\mathbf{m}}$ with $\mathbf{m} = \mathbf{0}$ is the reachability relation.

The following Lemma 7.2 shows that $\xrightarrow{*}_{\rho}$ seens as a subset of $\mathbb{Z}^{2d}$ is periodic for every run ρ as a composition of periodic relations (see Fig. 9). Note that in Section 8 we prove that these periodic relations are asymptotically definable.

Lemma 7.2. *The relation $\xrightarrow{*}_{\mathbf{m}}$ is periodic.*

Proof. Let us assume that $\mathbf{r}_1 \xrightarrow{*}_{\mathbf{m}} \mathbf{s}_1$ and $\mathbf{r}_2 \xrightarrow{*}_{\mathbf{m}} \mathbf{s}_2$. Since $\mathbf{r}_1 \xrightarrow{*}_{\mathbf{m}} \mathbf{s}_1$ we deduce that $\mathbf{r}_1 + \mathbf{r}_2 \xrightarrow{*}_{\mathbf{m}} \mathbf{s}_1 + \mathbf{r}_2$. Moreover, since $\mathbf{r}_2 \xrightarrow{*}_{\mathbf{m}} \mathbf{s}_2$ we deduce that $\mathbf{r}_2 + \mathbf{s}_1 \xrightarrow{*}_{\mathbf{m}} \mathbf{s}_2 + \mathbf{s}_1$. Therefore $\mathbf{r}_1 + \mathbf{r}_2 \xrightarrow{*}_{\mathbf{m}} \mathbf{s}_1 + \mathbf{s}_2$. □

We introduce a well order over the set of runs based on the following Lemma 7.3.

Lemma 7.3. *The following inclusion holds for every run ρ:*

$$(\mathrm{src}(\rho), \mathrm{tgt}(\rho))+ \xrightarrow{*}_{\rho} \quad \subseteq \quad \xrightarrow{*}$$

Proof. Assume that $\rho = \mathbf{m}_0 \dots \mathbf{m}_k$ with $\mathbf{m}_j \in \mathbb{N}^d$, and let $(\mathbf{r}, \mathbf{s})$ be a couple in the production relation $\xrightarrow{*}_{\rho}$. Since this relation is defined as a composition,

there exists a sequence $(\mathbf{v}_j)_{0\leq j\leq k+1}$ of vectors $\mathbf{v}_j \in \mathbb{N}^d$ satisfying the following relations with $\mathbf{v}_0 = \mathbf{r}$ and $\mathbf{v}_{k+1} = \mathbf{s}$:

$$\mathbf{v}_0 \xrightarrow[\mathbf{m}_0]{*} \mathbf{v}_1 \cdots \mathbf{v}_k \xrightarrow[\mathbf{m}_k]{*} \mathbf{v}_{k+1}$$

We introduce the vector $\mathbf{a}_j = \mathbf{m}_j - \mathbf{m}_{j-1}$ for every $j \in \{1, \ldots, k\}$. Since $\mathbf{m}_{j-1} \xrightarrow{\mathbf{a}_j} \mathbf{m}_j$ we deduce that $\mathbf{m}_{j-1} + \mathbf{v}_j \xrightarrow{\mathbf{a}_j} \mathbf{m}_j + \mathbf{v}_j$. Moreover, as $\mathbf{v}_j \xrightarrow[\mathbf{m}_j]{*} \mathbf{v}_{j+1}$, there exists a word $w_j \in \mathbf{A}^*$ such that $\mathbf{m}_j + \mathbf{v}_j \xrightarrow{w_j} \mathbf{m}_j + \mathbf{v}_{j+1}$. We deduce that the following relation holds:

$$\mathbf{m}_0 + \mathbf{v}_0 \xrightarrow{w_0\mathbf{a}_1 w_1 \ldots \mathbf{a}_k w_k} \mathbf{m}_k + \mathbf{v}_{k+1}$$

Therefore $(\mathbf{m}_0, \mathbf{m}_k) + (\mathbf{v}_0, \mathbf{v}_{k+1})$ is in the reachability relation. □

We introduce the order $\preceq$ over the set of runs defined by $\rho \preceq \rho'$ if the following inclusion holds:

$$(\mathrm{src}(\rho'), \mathrm{tgt}(\rho'))+ \xrightarrow[\rho']{*} \quad \subseteq \quad (\mathrm{src}(\rho), \mathrm{tgt}(\rho))+ \xrightarrow[\rho]{*}$$

In the reminder of this section we prove the following theorem. All the other results or definitions introduced in this section are not used in the sequel.

Theorem 7.4. *The order $\preceq$ is well.*

The order $\preceq$ is proved well thanks to the *Higmann's Lemma*. We first recall this lemma. Let us consider an order $\sqsubseteq$ over a set S. We introduce the order $\sqsubseteq^*$ over the set of words over S defined by $u \sqsubseteq^* v$ where $u = s_1 \ldots s_k$ with $s_j \in S$ if there exists a sequence $(t_j)_{1\leq j\leq k}$ with $t_j \in S$ and $s_j \sqsubseteq t_j$ and a sequence $(w_j)_{0\leq j\leq k}$ of words $w_j \in S^*$ such that $v = w_0 t_1 w_1 \ldots t_k w_k$.

Lemma 7.5 (Higmann's Lemma). *The ordered set $(S^*, \sqsubseteq^*)$ is well for every well ordered set $(S, \sqsubseteq)$.*

We associate to every run $\rho = \mathbf{m}_0 \ldots \mathbf{m}_k$ the word $\alpha(\rho) = (\mathbf{a}_1, \mathbf{m}_1) \ldots (\mathbf{a}_k, \mathbf{m}_k)$ where $\mathbf{a}_j = \mathbf{m}_j - \mathbf{m}_{j-1}$. Note that $\alpha(\rho)$ is a word over the alphabet $S = \mathbf{A} \times \mathbb{N}^d$. We introduce the order $\sqsubseteq$ over this alphabet by $(\mathbf{a}, \mathbf{m}) \sqsubseteq (\mathbf{a}', \mathbf{m}')$ if $\mathbf{a} = \mathbf{a}'$ and $\mathbf{m} \leq \mathbf{m}'$. Since $\mathbf{A}$ is a finite set and $\leq$ is a well order over $\mathbb{N}^d$, we deduce that $\sqsubseteq$ is a well order over S. From the Higmann's lemma, the order $\sqsubseteq^*$ is well over S^*. We introduce the well order $\trianglelefteq$ over the set of runs defined by $\rho \trianglelefteq \rho'$ if $\alpha(\rho) \sqsubseteq^* \alpha(\rho')$, $\mathrm{src}(\rho) \leq \mathrm{src}(\rho')$ and $\mathrm{tgt}(\rho) \leq \mathrm{tgt}(\rho')$. The following lemma provides a useful characterization of this order.

Lemma 7.6. *Let $\rho = \mathbf{m}_0 \ldots \mathbf{m}_k$ be a run and let ρ' be another run. We have $\rho \trianglelefteq \rho'$ if and only if there exists a sequence $(\mathbf{v}_j)_{0\leq j\leq k+1}$ of vectors in $\mathbb{N}^d$ such that $\rho' = \rho'_0 \ldots \rho'_k$ where ρ'_j is a run from $\mathbf{m}_j + \mathbf{v}_j$ to $\mathbf{m}_j + \mathbf{v}_{j+1}$.*

Proof. We introduce the sequence $(\mathbf{a}_j)_{1\leq j\leq k}$ defined by $\mathbf{a}_j = \mathbf{m}_j - \mathbf{m}_{j-1}$.

Assume first that $\rho \trianglelefteq \rho'$.

Since $\alpha(\rho) \sqsubseteq^* \alpha(\rho')$ we get $\alpha(\rho') = w_0(\mathbf{a}_1, \mathbf{m}'_1)w_1 \dots (\mathbf{a}_k, \mathbf{m}'_k)w_k$ where $w_j \in S^*$ and $\mathbf{m}'_j \geq \mathbf{m}_j$. We introduce the sequence $(\mathbf{v}_j)_{0 \leq j \leq k+1}$ defined by $\mathbf{v}_0 = \mathrm{src}(\rho') - \mathrm{src}(\rho)$, $\mathbf{v}_{k+1} = \mathrm{tgt}(\rho') - \mathrm{tgt}(\rho)$ and $\mathbf{v}_j = \mathbf{m}'_j - \mathbf{m}_j$ for every $j \in \{1, \dots, k\}$. Observe that $\mathbf{v}_j \in \mathbb{N}^d$ for every $j \in \{0, \dots, k+1\}$. We deduce that ρ' can be decomposed into $\rho' = \rho'_0 \dots \rho'_k$ where ρ'_j is the run from $\mathbf{m}_j + \mathbf{v}_j$ to $\mathbf{m}_j + \mathbf{v}_{j+1}$ such that $\alpha(\rho'_j) = w_j$.

Conversely let $(\mathbf{v}_j)_{0 \leq j \leq k+1}$ be a sequence of vectors in $\mathbb{N}^d$ such that $\rho' = \rho'_0 \dots \rho'_k$ where ρ'_j is a run from $\mathbf{m}_j + \mathbf{v}_j$ to $\mathbf{m}_j + \mathbf{v}_{j+1}$. We deduce that we have the following equality where $\mathbf{m}'_j = \mathbf{m}_j + \mathbf{v}_j$ and $\mathbf{a}'_j \in \mathbf{A}$:

$$\alpha(\rho') = \alpha(\rho'_0)(\mathbf{a}'_1, \mathbf{m}'_1)\alpha(\rho'_1) \dots (\mathbf{a}'_k, \mathbf{m}'_k)\alpha(\rho'_k)$$

Observe that $\mathbf{a}'_j = \mathrm{tgt}(\rho'_{j-1}) - \mathbf{m}'_j = (\mathbf{m}_j + \mathbf{v}_j) - (\mathbf{m}_{j-1} + \mathbf{v}_j)$ and in particular $\mathbf{a}'_j = \mathbf{a}_j$. We deduce that $\alpha(\rho) \sqsubseteq^* \alpha(\rho')$. Moreover, since $\mathrm{src}(\rho) \leq \mathrm{src}(\rho')$ and $\mathrm{tgt}(\rho) \leq \mathrm{tgt}(\rho')$ we deduce that $\rho \trianglelefteq \rho'$. □

Since $\trianglelefteq$ is a well order, the following lemma shows that $\preceq$ is a well order. We have proved Theorem 7.4.

Lemma 7.7. *$\rho \trianglelefteq \rho'$ implies $\rho \preceq \rho'$.*

Proof. Assume that $\rho = \mathbf{m}_0 \dots \mathbf{m}_k$. Lemma 7.6 shows that there exists a sequence $(\mathbf{v}_j)_{0 \leq j \leq k+1}$ of vectors in $\mathbb{N}^d$ such that $\rho' = \rho'_0 \dots \rho'_k$ where ρ'_j is a run from $\mathbf{m}_j + \mathbf{v}_j$ to $\mathbf{m}_j + \mathbf{v}_{j+1}$. Lemma 7.3 shows that $(\mathrm{src}(\rho'_j), \mathrm{tgt}(\rho'_j)) + \xrightarrow{*}_{\rho'_j} \subseteq \xrightarrow{*}$. Hence $(\mathbf{v}_j, \mathbf{v}_{j+1}) + \xrightarrow{*}_{\rho'_j} \subseteq \xrightarrow{*}_{\mathbf{m}_j}$. We deduce that $(\mathbf{v}_0, \mathbf{v}_{k+1}) + \xrightarrow{*}_{\rho'} \subseteq \xrightarrow{*}_{\rho}$ by composition. Since $(\mathrm{src}(\rho'), \mathrm{tgt}(\rho')) = (\mathrm{src}(\rho), \mathrm{tgt}(\rho)) + (\mathbf{v}_0, \mathbf{v}_{k+1})$ we get $\rho \preceq \rho'$ from the previous inclusion. □

8 Asymptotically Definable Production Relations

In this section we prove that production relations are asymptotically definable (Theorem 8.1). All the other results or definitions introduced in the section are not used in the sequel.

Theorem 8.1. *Production relations are asymptotically definable.*

The following lemma shows that asymptotically definable periodic relations are stable by composition. In particular it is sufficient to prove that production relations $\xrightarrow{*}_{\mathbf{m}}$ are asymptotically definable for every marking $\mathbf{m} \in \mathbb{N}^d$ in order to deduce that production relations $\xrightarrow{*}_{\rho}$ are asymptotically definable for every run ρ.

Lemma 8.2. *We have $\mathbb{Q}_{\geq 0}(R_1 \circ R_2) = (\mathbb{Q}_{\geq 0}R_1) \circ (\mathbb{Q}_{\geq 0}R_2)$ for every periodic relations over $\mathbb{Z}^d$.*

Proof. We have $R_1 \subseteq \mathbb{Q}_{\geq 0}R_1$ and $R_2 \subseteq \mathbb{Q}_{\geq 0}R_2$. Thus $R_1 \circ R_2 \subseteq C$ where $C = (\mathbb{Q}_{\geq 0}R_1) \circ (\mathbb{Q}_{\geq 0}R_2)$. As C is a conic set we get $\mathbb{Q}_{\geq 0}(R_1 \circ R_2) \subseteq C$. For

the converse inclusion, let us consider $(\mathbf{x},\mathbf{z}) \in C$. There exists $\mathbf{y} \in \mathbb{Q}^d$ such that $(\mathbf{x},\mathbf{y}) \in \mathbb{Q}_{\geq 0}R_1$ and $(\mathbf{y},\mathbf{z}) \in \mathbb{Q}_{\geq 0}R_2$. There exists $\lambda_1,\lambda_2 \in \mathbb{Q}_{\geq 0}$ such that $(\mathbf{x},\mathbf{y}) \in \lambda_1 R_1$ and $(\mathbf{y},\mathbf{z}) \in \lambda_2 R_2$. We introduce $n_1,n_2 \in \mathbb{N}_{>0}$ such that $n_1\lambda_1 \in \mathbb{N}$ and $n_2\lambda_2 \in \mathbb{N}$ and we deduce that $n(\mathbf{x},\mathbf{y}) \in R_1$ and $n(\mathbf{y},\mathbf{z}) \in R_2$ with $n = n_1n_2$. Hence $n(\mathbf{x},\mathbf{z}) \in R_1 \circ R_2$. We deduce that $(\mathbf{x},\mathbf{z}) \in \mathbb{Q}_{\geq 0}(R_1 \circ R_2)$. □

Theorem 3.8 shows that the conic set $\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m}}$ is definable if and only if the following conic set is finitely generated for every vector space $V \subseteq \mathbb{Q}^d \times \mathbb{Q}^d$:

$$\overline{(\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m}}) \cap V}$$

We introduce the periodic relation $\xrightarrow{*}_{\mathbf{m},V}$ defined as the intersection $\xrightarrow{*}_{\mathbf{m}} \cap V$. Let us observe that $\overline{(\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m}}) \cap V}$ is equal to $\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m},V}$. So, we just have to prove that the conic set $\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m},V}$ is finitely generated for every $\mathbf{m} \in \mathbb{N}^d$ and for every vector space $V \subseteq \mathbb{Q}^d \times \mathbb{Q}^d$.

We introduce the set $\Omega_{\mathbf{m},V}$ of runs ρ such that $(\mathrm{src}(\rho),\mathrm{tgt}(\rho)) - (\mathbf{m},\mathbf{m})$ is in $(\mathbb{N}^d \times \mathbb{N}^d) \cap V$. Note that a couple $(\mathbf{r},\mathbf{s}) \in \mathbb{N}^d \times \mathbb{N}^d$ satisfies $\mathbf{r} \xrightarrow{*}_{\mathbf{m},V} \mathbf{s}$ if and only if there exists a run $\rho \in \Omega_{\mathbf{m},V}$ such that $\mathrm{src}(\rho) = \mathbf{m}+\mathbf{r}$ and $\mathrm{tgt}(\rho) = \mathbf{m}+\mathbf{s}$. We introduce the set $\mathbf{Q}_{\mathbf{m},V}$ of markings $\mathbf{q}$ that occurs in at least one run $\rho \in \Omega_{\mathbf{m},V}$. In general the set $\mathbf{Q}_{\mathbf{m},V}$ is infinite. We consider the set $I_{\mathbf{m},V}$ of $i \in \{1,\ldots,d\}$ such that $\{\mathbf{q}(i) \mid \mathbf{q} \in \mathbf{Q}_{\mathbf{m},V}\}$ is infinite. We observe that if $i \in I_{\mathbf{m},V}$ there exists a sequence of markings in $\mathbf{Q}_{\mathbf{m},V}$ such that the ith component is strictly increasing. We are going to prove that there exists a sequence of markings in $\mathbf{Q}_{\mathbf{m},V}$ such that every component in $I_{\mathbf{m},V}$ is strictly increasing. This property is proved by introducing the intraproductions. An *intraproduction* for $(\mathbf{m},V)$ is a triple $(\mathbf{r},\mathbf{x},\mathbf{s})$ such that $\mathbf{x} \in \mathbb{N}^d$, $(\mathbf{r},\mathbf{s}) \in (\mathbb{N}^d \times \mathbb{N}^d) \cap V$ and such that:

$$\mathbf{r} \xrightarrow{*}_{\mathbf{m}} \mathbf{x} \xrightarrow{*}_{\mathbf{m}} \mathbf{s}$$

Since $\xrightarrow{*}_{\mathbf{m}}$ is a periodic relation we deduce that the set of intraproductions is stable by addition. In particular $\mathbf{m}+n\mathbf{x}$ occurs in at least one run of $\Omega_{\mathbf{m},V}$ for every intraproduction $(\mathbf{r},\mathbf{x},\mathbf{s})$ and for every $n \in \mathbb{N}$. Hence, if $\mathbf{x}(i) > 0$ then $i \in I_{\mathbf{m},V}$. An *intraproduction* for $(\mathbf{m},V)$ is said to be *total* if $\mathbf{x}(i) > 0$ for every $i \in I_{\mathbf{m},V}$.

Lemma 8.3. *There exists a total intraproduction for* $(\mathbf{m},V)$.

Proof. Since finite sums of intraproductions are intraproductions, it is sufficient to prove that for every $i \in I_{\mathbf{m},V}$ there exists an intraproduction $(\mathbf{r},\mathbf{x},\mathbf{s})$ for $(\mathbf{m},V)$ such that $\mathbf{x}(i) > 0$. We fix $i \in I$.

Let us first prove that there exists $\mathbf{q} \leq \mathbf{q}'$ in $\mathbf{Q}_{\mathbf{m},V}$ such that $\mathbf{q}(i) < \mathbf{q}'(i)$. Since $i \in I$ there exists a sequence $(\mathbf{q}_n)_{n\in\mathbb{N}}$ of markings $\mathbf{q}_n \in \mathbf{Q}_{\mathbf{m},V}$ such that $(\mathbf{q}_n(i))_{n\in\mathbb{N}}$ is strictly increasing. Since $(\mathbb{N}^d,\leq)$ is well ordered, we can extract for this sequence a subsequence that is non decreasing for $\leq$. We have proved that there exists $\mathbf{q} \leq \mathbf{q}'$ in $\mathbf{Q}_{\mathbf{m},V}$ such that $\mathbf{q}(i) < \mathbf{q}'(i)$.

As $\mathbf{q} \in \mathbf{Q}_{\mathbf{m},V}$ then $\mathbf{q}$ occurs in a run in $\Omega_{\mathbf{m},V}$. Hence there exists $(\mathbf{r},\mathbf{s}) \in (\mathbb{N}^d \times \mathbb{N}^d) \cap V$ such that:

$$\mathbf{m}+\mathbf{r} \xrightarrow{*} \mathbf{q} \xrightarrow{*} \mathbf{m}+\mathbf{s}$$

Symmetrically, as $\mathbf{q}' \in \mathbf{Q}_{\mathbf{m},V}$ there exists $(\mathbf{r}', \mathbf{s}') \in (\mathbb{N}^d \times \mathbb{N}^d) \cap V$ such that:

$$\mathbf{m} + \mathbf{r}' \xrightarrow{*} \mathbf{q}' \xrightarrow{*} \mathbf{m} + \mathbf{s}'$$

Let us introduce $\mathbf{v} = \mathbf{q}' - \mathbf{q}$. We deduce:

- $(\mathbf{m} + \mathbf{r}') + \mathbf{r} \xrightarrow{*} \mathbf{q}' + \mathbf{r}$ from $\mathbf{m} + \mathbf{r}' \xrightarrow{*} \mathbf{q}'$.
- $\mathbf{q} + (\mathbf{v} + \mathbf{r}) \xrightarrow{*} (\mathbf{m} + \mathbf{s}) + (\mathbf{v} + \mathbf{r})$ from $\mathbf{q} \xrightarrow{*} \mathbf{m} + \mathbf{s}$.
- $(\mathbf{m} + \mathbf{r}) + (\mathbf{v} + \mathbf{s}) \xrightarrow{*} \mathbf{q} + (\mathbf{v} + \mathbf{s})$ from $\mathbf{m} + \mathbf{r} \xrightarrow{*} \mathbf{q}$.
- $\mathbf{q}' + \mathbf{s} \xrightarrow{*} (\mathbf{m} + \mathbf{s}') + \mathbf{s}$ from $\mathbf{q}' \xrightarrow{*} \mathbf{m} + \mathbf{s}'$.

Since $\mathbf{q}' + \mathbf{r} = \mathbf{q} + \mathbf{v} + \mathbf{r}$ and $\mathbf{q} + \mathbf{v} + \mathbf{s} = \mathbf{q}' + \mathbf{s}$, we have proved the following relations where $\mathbf{x} = \mathbf{s} + \mathbf{v} + \mathbf{r}$:

$$\mathbf{r} + \mathbf{r}' \xrightarrow{*}_{\mathbf{m}} \mathbf{x} \xrightarrow{*}_{\mathbf{m}} \mathbf{s} + \mathbf{s}'$$

As $(\mathbf{r} + \mathbf{r}', \mathbf{s} + \mathbf{s}') \in (\mathbb{N}^d \times \mathbb{N}^d) \cap V$ we deduce that $(\mathbf{r} + \mathbf{r}', \mathbf{x}, \mathbf{s} + \mathbf{s}')$ is an intraproduction for $(\mathbf{m}, V)$. Since $\mathbf{x}(i) > 0$ we are done. □

Let us introduce an additional element $\infty \notin \mathbb{N}$ and let $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$. A vector in $\mathbb{N}_\infty^d$ is called an *extended marking* and the set $I = \{i \in \{1, \ldots, d\} \mid \mathbf{m}(i) = \infty\}$ is called the set of *relaxed components* of an extended marking $\mathbf{m}$. Given a finite set $I \subseteq \{1, \ldots, d\}$ and a marking $\mathbf{m} \in \mathbb{N}^d$, we denote by $\mathbf{m}^I$ the extended marking defined by $\mathbf{m}^I(i) = \infty$ if $i \in I$ and $\mathbf{m}^I(i) = \mathbf{m}(i)$ if $i \notin I$. Given a word $w = \mathbf{a}_1 \ldots \mathbf{a}_k$ of vectors $\mathbf{a}_j \in \mathbf{A}$, we extend the relation $\xrightarrow{w}$ over the set of extended markings relaxed over a set I by $\mathbf{x} \xrightarrow{w} \mathbf{y}$ if there exists a word $\rho = \mathbf{m}_0 \ldots \mathbf{m}_k$ of extended markings relaxed over I such that $(\mathbf{x}, \mathbf{y}) = (\mathbf{m}_0, \mathbf{m}_k)$ and $\mathbf{m}_j(i) = \mathbf{m}_{j-1}(i) + \mathbf{a}_j(i)$ for every $j \in \{1, \ldots, k\}$ and for every $i \in \{1, \ldots, d\} \backslash I$. The word ρ is unique and it is called the *run* from $\mathbf{x}$ to $\mathbf{y}$ labeled by w.

We introduce the finite graph $G_{\mathbf{m},V} = (\mathbf{Q}, \mathbf{A}, E)$ where $\mathbf{Q} = \{\mathbf{q}^{I_{\mathbf{m},V}} \mid \mathbf{q} \in \mathbf{Q}_{\mathbf{m},V}\}$ and where $E = \{(\mathbf{p}^{I_{\mathbf{m},V}}, \mathbf{a}, \mathbf{q}^{I_{\mathbf{m},V}}) \mid \mathbf{p}, \mathbf{q} \in \mathbf{Q}_{\mathbf{m},V} \wedge \mathbf{q} = \mathbf{p} + \mathbf{a}\}$. We introduce the periodic relation $R_{\mathbf{m},V}$ of couples $(\mathbf{r}, \mathbf{s}) \in (\mathbb{N}^d \times \mathbb{N}^d) \cap V$ such that $\mathbf{r}(i) = \mathbf{s}(i) = 0$ for every $i \in \{1, \ldots, d\} \backslash I_{\mathbf{m},V}$ and such that there exists a cycle in $G_{\mathbf{m},V}$ on the state $\mathbf{m}^{I_{\mathbf{m},V}}$ labeled by a word $\mathbf{a}_1 \ldots \mathbf{a}_k$ where $\mathbf{a}_j \in \mathbf{A}$ such that $\mathbf{r} + \sum_{j=1}^{k} \mathbf{a}_j = \mathbf{s}$.

Lemma 8.4. *The periodic relation $R_{\mathbf{m},V}$ is Presburger.*

Proof. This is a classical result based on the fact that the Parikh image of a regular language is Presburger. □

Lemma 8.5. *The following equality holds:*

$$\overline{\mathbb{Q}_{\geq 0} R_{\mathbf{m},V}} = \overline{\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m},V}}$$

Proof. Let us first prove the inclusion $\supseteq$. Let $(\mathbf{r}, \mathbf{s})$ such that $\mathbf{r} \xrightarrow{*}_{\mathbf{m},V} \mathbf{s}$. In this case there exists a word $w \in \mathbf{A}^*$ such that $\mathbf{m} + \mathbf{r} \xrightarrow{w} \mathbf{m} + \mathbf{s}$. Observe that $\mathbf{m} + n\mathbf{r}$ and $\mathbf{m} + n\mathbf{s}$ are in $\mathbf{Q}_{\mathbf{m},V}$ for every $n \in \mathbb{N}$. Hence $\mathbf{r}(i) > 0$ or $\mathbf{s}(i) > 0$ implies

$i \in I_{\mathbf{m},V}$ and we deduce that $\mathbf{m}^{I_{\mathbf{m},V}} \xrightarrow{w} \mathbf{m}^{I_{\mathbf{m},V}}$. Therefore w is the label of cycle in $G_{\mathbf{m},V}$ on $\mathbf{m}^{I_{\mathbf{m},V}}$. We have proved that $(\mathbf{r},\mathbf{s}) \in R_{\mathbf{m},V}$.

Now let us prove the inclusion $\subseteq$. We consider $(\mathbf{r},\mathbf{s}) \in R_{\mathbf{m},V}$. In this case $(\mathbf{r},\mathbf{s}) \in (\mathbb{N}^d \times \mathbb{N}^d) \cap V$ satisfies $\mathbf{r}(i) = \mathbf{s}(i) = 0$ for every $i \notin I_{\mathbf{m},V}$ and there exists a word $w = \mathbf{a}_1 \dots \mathbf{a}_k$ of vectors $\mathbf{a}_j \in \mathbf{A}$ that labels a cycle in $G_{\mathbf{m},V}$ on $\mathbf{m}^{I_{\mathbf{m},V}}$ and such that $\mathbf{m} + \mathbf{r} + \sum_{j=1}^{k} \mathbf{a}_j = \mathbf{m} + \mathbf{s}$. Let us consider a total intraproduction $(\mathbf{r}', \mathbf{x}, \mathbf{s}')$ for $(\mathbf{m}, V)$. Given $p \in \mathbb{N}$ and $j \in \{0, \dots, k\}$ we introduce the following vector $\mathbf{m}_{p,j}$:

$$\mathbf{m}_{p,j} = \mathbf{m} + \mathbf{r} + p\mathbf{x} + \mathbf{a}_1 + \cdots + \mathbf{a}_j$$

Let us first prove that there exists $p \in \mathbb{N}$ such that $\mathbf{m}_{p,j}(i) \in \mathbb{N}$ for every $i \in I_{\mathbf{m},V}$ and $j \in \{0, \dots, k\}$. Let $i \in I_{\mathbf{m},V}$ and $j \in \{0, \dots, k\}$, since $\mathbf{x}(i) > 0$, there exists $p_{i,j} \in \mathbb{N}$ such that $\mathbf{m}_{p,j}(i) \in \mathbb{N}$ for every $p \geq p_{i,j}$. We deduce that there exists $p \in \mathbb{N}$ such that $\mathbf{m}_{p,j}(i) \in \mathbb{N}$ for every $i \in I_{\mathbf{m},V}$ and $j \in \{0, \dots, k\}$.

Now we prove that $\mathbf{m}_{p,j}(i) \in \mathbb{N}$ for every $i \in \{1, \dots, d\} \backslash I_{\mathbf{m},V}$ and $j \in \{0, \dots, k\}$. Let $j \in \{0, \dots, k\}$. Since w is the label of a cycle on $\mathbf{m}^{I_{\mathbf{m},V}}$, there exists an extended marking $\mathbf{q}_j$ relaxed over $I_{\mathbf{m},V}$ such that the following relation holds:

$$\mathbf{m}^{I_{\mathbf{m},V}} \xrightarrow{\mathbf{a}_1 \dots \mathbf{a}_j} \mathbf{q}_j$$

We deduce that for every $i \in \{1, \dots, d\} \backslash I_{\mathbf{m},V}$ we have $\mathbf{m}(i) + \mathbf{a}_1(i) + \cdots + \mathbf{a}_j(i) = \mathbf{q}_j(i)$. Since $\mathbf{r}(i) = 0$ and $\mathbf{x}(i) = 0$ we get $\mathbf{m}_{p,j}(i) \in \mathbb{N}$.

We have proved that $\mathbf{m}_{p,j} \in \mathbb{N}^d$ for every $j \in \{0, \dots, k\}$. Since $\mathbf{m}_{p,j} - \mathbf{m}_{p,j-1} = \mathbf{a}_j$ we deduce that $\rho_p = \mathbf{m}_{p,0} \dots \mathbf{m}_{p,k}$ is a run. Note that $\mathbf{m}_{p,0} = \mathbf{m} + p\mathbf{x} + \mathbf{r}$ and $\mathbf{m}_{p,k} = \mathbf{m} + p\mathbf{x} + \mathbf{r} + \sum_{j=1}^{k} \mathbf{a}_j = \mathbf{m} + p\mathbf{x} + \mathbf{s}$. We have proved that the following relation holds:

$$\mathbf{m} + p\mathbf{x} + \mathbf{r} \xrightarrow{w} \mathbf{m} + p\mathbf{x} + \mathbf{s}$$

In particular $(\mathbf{r},\mathbf{s})$ is in the production relation $\xrightarrow{*}_{\mathbf{m}'}$ where $\mathbf{m}' = \mathbf{m} + p\mathbf{x}$. Since a production relation is periodic we get $\mathbf{m}' + n\mathbf{r} \xrightarrow{*} \mathbf{m}' + n\mathbf{s}$ for every $n \in \mathbb{N}$. As $(p\mathbf{r}', p\mathbf{x}, p\mathbf{s}')$ is an intraproduction for $(\mathbf{m}, V)$ we get $\mathbf{m} + p\mathbf{r}' \xrightarrow{*} \mathbf{m}' \xrightarrow{*} \mathbf{m} + p\mathbf{s}'$. We deduce the relation $(\mathbf{m} + p\mathbf{r}') + n\mathbf{r} \xrightarrow{*} \mathbf{m}' + n\mathbf{r}$ from $(\mathbf{m} + p\mathbf{r}') \xrightarrow{*} \mathbf{m}'$, and the relation $\mathbf{m}' + n\mathbf{s} \xrightarrow{*} (\mathbf{m} + p\mathbf{s}') + n\mathbf{s}$ from $\mathbf{m}' \xrightarrow{*} (\mathbf{m} + p\mathbf{s}')$. We deduce that the following relation holds for every $n \in \mathbb{N}$:

$$\mathbf{m} + p\mathbf{r}' + n\mathbf{r} \xrightarrow{*} \mathbf{m} + p\mathbf{s}' + n\mathbf{s}$$

Hence $p(\mathbf{r}', \mathbf{s}') + \mathbb{N}(\mathbf{r}, \mathbf{s}) \subseteq \xrightarrow{*}_{\mathbf{m},V}$. Thus $(\mathbf{r},\mathbf{s}) \in \overline{\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m},V}}$. From the inclusion $R_{\mathbf{m},V} \subseteq \mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m},V}$ we get the inclusion $\overline{\mathbb{Q}_{\geq 0} R_{\mathbf{m},V}} \subseteq \overline{\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m},V}}$. □

Lemma 8.6. *The conic set* $\overline{\mathbb{Q}_{\geq 0}\mathbf{P}}$ *is finitely generated for every Presburger periodic set* $\mathbf{P}$.

Proof. Let us consider a Presburger periodic set $\mathbf{P}$. Since $\mathbf{P}$ is Presburger then $\mathbf{P} = \bigcup_{j=1}^{k} \mathbf{b}_j + \mathbf{P}_j$ where $\mathbf{b}_j \in \mathbb{Z}^d$ and $\mathbf{P}_j \subseteq \mathbb{Z}^d$ is a finitely generated periodic set. We introduce the finitely generated conic set $\mathbf{C} = \sum_{j=1}^{k} (\mathbb{Q}_{\geq 0}\mathbf{b}_j + \mathbf{C}_j)$ where $\mathbf{C}_j$ is the finitely generated conic set $\mathbf{C}_j = \mathbb{Q}_{\geq 0}\mathbf{P}_j$. Since $\mathbf{P} \subseteq \mathbf{C}$ and $\mathbf{C}$ is a conic

set we deduce the inclusion $\mathbb{Q}_{\geq 0}\mathbf{P} \subseteq \mathbf{C}$. As $\mathbf{C}$ is finitely generated we deduce that $\mathbf{C}$ is closed. Hence $\overline{\mathbb{Q}_{\geq 0}\mathbf{P}} \subseteq \mathbf{C}$. For the other inclusion let $\mathbf{p} \in \mathbf{P}_j$. For every $n \in \mathbb{N}$ we have $\mathbf{b}_j + n\mathbf{p} \in \mathbf{P}$. Hence $\frac{1}{n}\mathbf{b}_j + \mathbf{p} \in \mathbb{Q}_{\geq 0}\mathbf{P}$ for every $n \in \mathbb{N}_{>0}$. We deduce that $\mathbf{p} \in \overline{\mathbb{Q}_{\geq 0}\mathbf{P}}$. Therefore $\mathbf{P}_j \subseteq \overline{\mathbb{Q}_{\geq 0}\mathbf{P}}$. We get $\mathbf{C}_j \subseteq \overline{\mathbb{Q}_{\geq 0}\mathbf{P}}$. As $\mathbb{Q}_{\geq 0}\mathbf{b}_j \subseteq \mathbb{Q}_{\geq 0}\mathbf{P} \subseteq \overline{\mathbb{Q}_{\geq 0}\mathbf{P}}$ we have proved the inclusion $\mathbf{C} \subseteq \overline{\mathbb{Q}_{\geq 0}\mathbf{P}}$. Hence the previous inclusion is in fact an equality. □

Now, we can prove Theorem 8.1. Lemma 8.4 shows that $R_{\mathbf{m},V}$ is a Presburger periodic relation. Lemma 8.6 proves that the conic set $\overline{\mathbb{Q}_{\geq 0}R_{\mathbf{m},V}}$ is finitely generated. Lemma 8.5 shows that $\overline{\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m},V}}$ is finitely generated. Hence $(\overline{\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m}}}) \cap V$ is a finitely generated conic set for every vector space $V \subseteq \mathbb{Q}^d \times \mathbb{Q}^d$. Theorem 3.8 shows that the conic relation $\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m}}$ is definable. Hence $\xrightarrow{*}_{\mathbf{m}}$ is an asymptotically definable periodic relation.

9 Almost Semilinear Reachability Relations

In this section we prove the following Theorem 9.1. All the other results or definitions introduced in this section are not used in the sequel.

Theorem 9.1. *The reachability relation of a Vector Addition System is an almost semilinear relation.*

We are interested in proving that $\xrightarrow{*}$ is an almost semilinear relation. We first inspect the intersection $\xrightarrow{*} \cap((\mathbf{m},\mathbf{n}) + P)$ where $(\mathbf{m},\mathbf{n}) \in \mathbb{N}^d \times \mathbb{N}^d$ and $P \subseteq \mathbb{N}^d \times \mathbb{N}^d$ is a finitely generated periodic relation. We introduce the order $\leq_P$ over P defined by $p \leq_P p'$ if $p' \in p + P$. Since P is finitely generated we deduce that $\leq_P$ is a well order over P (Dickson's Lemma). We introduce the set $\Omega_{\mathbf{m},P,\mathbf{n}}$ of runs ρ such that $(\mathrm{src}(\rho),\mathrm{tgt}(\rho)) \in (\mathbf{m},\mathbf{n}) + P$. This set is well ordered by the relation $\preceq_P$ defined by $\rho \preceq_P \rho'$ if $\rho \preceq \rho'$, $(\mathrm{src}(\rho),\mathrm{tgt}(\rho)) - (\mathbf{m},\mathbf{n}) \leq_P (\mathrm{src}(\rho'),\mathrm{tgt}(\rho')) - (\mathbf{m},\mathbf{n})$. We deduce that $\min_{\preceq_P}(\Omega_{\mathbf{m},P,\mathbf{n}})$ is finite.

Lemma 9.2. *The following equality holds:*

$$\xrightarrow{*} \cap((\mathbf{m},\mathbf{n}) + P) = \bigcup_{\rho \in \min_{\preceq_P}(\Omega_{\mathbf{m},P,\mathbf{n}})} (\mathrm{src}(\rho),\mathrm{tgt}(\rho)) + (\xrightarrow{*}_{\rho} \cap P)$$

Proof. Let us first prove $\supseteq$. Let $\rho \in \Omega_{\mathbf{m},P,\mathbf{n}}$. Lemma 7.3 shows that the inclusion $(\mathrm{src}(\rho),\mathrm{tgt}(\rho))+ \xrightarrow{*}_{\rho} \subseteq \xrightarrow{*}$ holds. Since $(\mathrm{src}(\rho),\mathrm{tgt}(\rho)) \in (\mathbf{m},\mathbf{n}) + P$ and P is periodic we deduce the inclusion $\supseteq$.

Let us prove $\subseteq$. Let $(\mathbf{x}',\mathbf{y}')$ in the intersection $\xrightarrow{*} \cap((\mathbf{m},\mathbf{n})+P)$. There exists a run $\rho' \in \Omega_{\mathbf{m},P,\mathbf{n}}$ such that $\mathbf{x}' = \mathrm{src}(\rho')$ and $\mathbf{y}' = \mathrm{tgt}(\rho')$. Since $\preceq_P$ is a well order, there exists $\rho \in \min_{\preceq_P}(\Omega_{\mathbf{m},P,\mathbf{n}})$ such that $\rho \preceq_P \rho'$. We deduce that $(\mathbf{x}',\mathbf{y}')$ is in $(\mathrm{src}(\rho),\mathrm{tgt}(\rho))+ \xrightarrow{*}_{\rho}$. We get $(\mathbf{x}',\mathbf{y}') \in (\mathrm{src}(\rho),\mathrm{tgt}(\rho)) + (\xrightarrow{*}_{\rho} \cap P)$ and we have proved the inclusion $\subseteq$. □

Theorem 8.1 shows $\xrightarrow{*}_\rho$ is an asymptotically definable periodic relation. Since P is a finitely generated periodic relation we deduce that P is asymptotically definable. Lemma 4.5 shows that the class of asymptotically definable periodic relations is stable by finite intersections. We deduce that $\xrightarrow{*}_\rho \cap P$ is asymptotically definable. Thanks to the previous lemma we have proved that $\xrightarrow{*}$ is almost semilinear and Theorem 9.1 is proved.

10 Conclusion

The reachability problem for Vector Additions Systems consists to decide for a triple $(\mathbf{m}, \mathbf{A}, \mathbf{n})$ where $\mathbf{m}, \mathbf{n}$ are two markings of a Vector Addition System $\mathbf{A}$ if there exists a word $w \in \mathbf{A}^*$ such that $\mathbf{m} \xrightarrow{w} \mathbf{n}$. The following algorithm decides this problem.

```
1  Reachability ( m , A , n )
2     k ← 0
3     repeat forever
4        for each word w ∈ A* of length k
5           if m -w-> n
6              return "reachable"
7        for each Presburger formula ψ of length k
8           if ψ(m) and ¬ψ(n) are true and
9              x ≥ 0 ∧ y ≥ 0 ∧ ψ(x) ∧ y ∈ x + A ∧ ¬ψ(y) unsat
10             return "unreachable"
11       k ← k + 1
```

The correctness is immediate since when the algorithm returns "reachable" we deduce that there exists a word $w \in \mathbf{A}^*$ such that $\mathbf{m} \xrightarrow{w} \mathbf{n}$ and when it returns "unreachable" we deduce a Presburger formula ψ that denotes a set $\mathbf{I}$ satisfying $\mathbf{m} \in \mathbf{I}$ (since $\psi(\mathbf{m})$ is true), $\mathbf{n} \notin \mathbf{I}$ (since $\neg\psi(\mathbf{n})$ is true), and such that $\mathbf{I}$ is a forward invariant (since $\mathbf{x} \geq \mathbf{0} \wedge \mathbf{y} \geq \mathbf{0} \wedge \psi(\mathbf{x}) \wedge \mathbf{y} \in \mathbf{x} + \mathbf{A} \wedge \neg\psi(\mathbf{y})$ is unsatisfiable). The termination is guaranteed by the following Theorem 10.1.

Theorem 10.1. *For every pair of markings* $(\mathbf{m}, \mathbf{n})$ *in the complement of the reachability relation of a Vector Addition System, there exists a partition of the set of markings into a Presburger forward invariant that contains* $\mathbf{m}$ *and a Presburger backward invariant that contains* $\mathbf{n}$.

Proof. Let us consider $\mathbf{X} = \{\mathbf{m}\}$ and $\mathbf{Y} = \{\mathbf{n}\}$ and let R^* be the reachability relation of the Vector addition system. Theorem 9.1 shows that R^* is an almost semilinear relation. Since R^* is reflexive and transitive and such that $(\mathbf{X} \times \mathbf{Y}) \cap R^* = \emptyset$, Theorem 6.1 shows that there exists a partition of the set of markings into a Presburger forward invariant set that contains $\mathbf{X}$ and a Presburger backward invariant set that contains $\mathbf{Y}$. □

This algorithm does not require the classical KLMST decomposition. Note however that the complexity of this algorithm is still open. In fact, the complexity

depends on the minimal size of a word $w \in \mathbf{A}^*$ such that $\mathbf{m} \xrightarrow{w} \mathbf{n}$ if $\mathbf{m} \xrightarrow{*} \mathbf{n}$, and the minimal size of a Presburger formula $\psi(\mathbf{x})$ denoting a forward invariant $\mathbf{I}$ such that $\mathbf{m} \in \mathbf{I}$ and $\mathbf{n} \notin \mathbf{I}$ otherwise. We left as an open question the problem of computing lower and upper bounds for these sizes. Note that the VAS exhibiting a large (Ackermann size) but finite reachability set given in [8] does not directly provide an Ackermann lower-bound for these sizes since Presburger forward invariants can over-approximate reachability sets.

As future work we are interested in providing complexity bounds on formulas in $\mathrm{FO}(\mathbb{Q}, +, \leq, 0, 1)$ denoting the definable conic sets $\mathbb{Q}_{\geq 0} \xrightarrow{*}_{\mathbf{m}}$.

References

1. Esparza, J., Nielsen, M.: Decidability issues for petri nets - a survey. Bulletin of the European Association for Theoretical Computer Science 52, 245–262 (1994)
2. Ginsburg, S., Spanier, E.H.: Semigroups, Presburger formulas and languages. Pacific Journal of Mathematics 16(2), 285–296 (1966)
3. Hauschildt, D.: Semilinearity of the Reachability Set is Decidable for Petri Nets. PhD thesis, University of Hamburg (1990)
4. Kosaraju, S.R.: Decidability of reachability in vector addition systems (preliminary version). In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC 1982), May 5-7, pp. 267–281. ACM, San Francisco (1982)
5. Lambert, J.L.: A structure to decide reachability in petri nets. Theoretical Computer Science 99(1), 79–104 (1992)
6. Leroux, J.: The general vector addition system reachability problem by Presburger inductive invariants. In: LICS 2009, pp. 4–13 (2009)
7. Mayr, E.W.: An algorithm for the general petri net reachability problem. In: Conference Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computation (STOC 1981), May 11-13, pp. 238–246. ACM, Milwaukee (1981)
8. Mayr, E.W., Meyer, A.R.: The complexity of the finite containment problem for petri nets. J. ACM 28(3), 561–576 (1981)
9. Sacerdote, G.S., Tenney, R.L.: The decidability of the reachability problem for vector addition systems (preliminary version). In: Conference Record of the Ninth Annual ACM Symposium on Theory of Computing, May 2-4, pp. 61–76. ACM, Boulder (1977)
10. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley and Sons, New York (1987)

Abstract Numeration Systems

Narad Rampersad

Institute of Mathematics, University of Liège
Grande Traverse 12 (B37), 4000 Liège, Belgium
nrampers@ulg.ac.be

Abstract. We survey various results concerning abstract numeration systems. We begin with the classical case of the integer base numeration systems, then examine the more general case of linear numeration systems. Finally we discuss abstract numeration systems, which generalize even further the two previous classes of numeration systems.

1 Introduction

The most general definition of a *numeration system* is simply a rule for representing numbers as words. We are most familiar with the classical integer base numeration systems, notably the decimal, or base-10 system, and, for computer scientists, the binary, or base-2 system. In many cases, the choice of base is somewhat arbitrary and there are many properties of numbers that are independent of their representations as words. However, there are also examples of rather non-trivial properties of the integers that can be characterized in terms of their representations in a certain base. For example, Gauss (see Nathanson [29]) proved that a number n is the sum of three squares if and only if n is not of the form $4^a(8m+7)$. This condition can be verified rather easily, given the binary represention of n. Hence the non-trivial property of a number being a sum of three squares depends very much on the form of its representation in base 2.

If X is a set of integers, the set of representations of the elements of X in our chosen numeration system forms a *language* L. Our general object of study is those sets X for which L can be *recognized* by the simplest possible computing device: a finite automaton. Of course, the notion of *recognizability* depends on our choice of numeration system. We will consider different classes of numeration systems in increasing order of generality. We begin with the *integer base numeration systems*, proceed to the more general class of *linear numeration systems*, and conclude by examining the class of *abstract numeration systems*, which contains the other two classes as special cases.

2 Integer Bases

2.1 Basic Definitions and Results

We begin by examining the familiar integer base numeration systems—the systems wherein we represent integers as sums of powers of a fixed base k. Let $k \geq 2$

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 65–79, 2011.

be an integer. For any non-negative integer n, we denote the base-k representation of n by $[n]_k$. Similarly, for any word w over the alphabet $\{0, 1, \ldots, k-1\}$, we denote the value of w, interpreted as an integer written in base k, by $\langle w \rangle_k$. A set $X \subseteq \mathbb{N}$ is *k-recognizable* (or *k-automatic*) if the language $[X]_k$ consisting of the base-k representations of the elements of X is accepted by a finite automaton.

Example 1. The set $\mathbb{N}$ is k-recognizable for all k, since $[\mathbb{N}]_k$ is the regular language

$$\{0, 1, \ldots, k-1\}^* \setminus (0\{0, 1, \ldots, k-1\}^*)$$

consisting of all words over $\{0, 1, \ldots, k-1\}$ that do not begin with a 0.

Example 2. The prototypical example of a 2-recognizable set is the *Thue–Morse set*

$$\{n \in \mathbb{N} : [n]_2 \text{ contains an odd number of 1's}\}.$$

This is clearly 2-recognizable as one can easily define a finite automaton that accepts those inputs over $\{0, 1\}$ that contain an odd number of 1's.

It is clear from the definition of k-recognizable, and from well-known closure properties of regular languages, that the class of k-recognizable sets is closed under the Boolean operations of union, intersection, and complement.

The following result gives certain properties of the growth of a k-recognizable set. Charlier and Rampersad [10] have recently given a more precise description of the growth of k-recognizable sets.

Theorem 1 (Eilenberg [15]). *Let $k \geq 2$ be an integer. A k-recognizable set $X = (x_n)_{n \geq 0}$ of non-negative integers satisfies either*

$$\limsup_{n \to \infty} (x_{n+1} - x_n) < \infty$$

or

$$\limsup_{n \to \infty} \frac{x_n + 1}{x_n} > 1.$$

This theorem can be used to show that certain sets are not k-recognizable for any k.

Example 3. The set $\{n^2 : n \in \mathbb{N}\}$ of squares is not k-recognizable for any k, since clearly

$$\limsup_{n \to \infty} ((n+1)^2 - n^2) = \infty$$

and

$$\limsup_{n \to \infty} \frac{(n+1)^2}{n^2} = 1.$$

Example 4. The set of prime numbers is not k-recognizable for any k. It is well-known that there can be arbitrarily large gaps between sucessive prime numbers. Moreover, if p_n is the n-th prime, the Prime Number Theorem implies that

$$p_{n+1}/p_n \to 1,$$

so neither condition of Theorem 1 is satisfied.

2.2 Cobham's Theorem and Periodicity

It is natural to wonder to what extent the recognizability of a set X depends on the choice of base k. Can one pass from one base to another without losing recognizability? A celebrated result of Cobham [13] answers this question. Cobham's Theorem characterizes the sets that are recognizable in all integer bases $k \geq 2$.

We first need the following definitions. Two numbers k and ℓ are *multiplicatively independent* if $k^m = \ell^n$ implies $m = n = 0$. A subset of the integers is *ultimately periodic* if it is a finite union of arithmetic progressions.

Theorem 2 (Cobham's Theorem [13]). *Let $k, \ell \geq 2$ be two multiplicatively independent integers and let $X \subseteq \mathbb{N}$. The set X is both k-recognizable and ℓ-recognizable if and only if it is ultimately periodic.*

Example 5. The set $P_2 = \{2^n : n \geq 0\}$ of powers of 2 is clearly 2-recognizable, since $[P_2]_2 = 10^*$ is a regular language. Clearly P_2 is not an ultimately periodic set. Cobham's Theorem therefore implies that P_2 is not 3-recognizable.

Cobham's Theorem shows that ultimately periodic sets are of particular interest among the k-recognizable sets. This leads to the following decidability question, known as the *periodicity problem*: Given an automaton accepting the base-k representations of some set X, determine if X is ultimately periodic.

Theorem 3 (Honkala [21]). *The periodicity problem is decidable for k-recognizable sets.*

This decidability result was subsequently reproved several times by various authors, such as Muchnik [28], Fagnot [17], and Allouche et al. [2]. Leroux [26] gave a polynomial time algorithm.

2.3 Alternative Characterizations

Next we present an alternative characterization of k-recognizable sets. We first need the following definitions. A map $h : \Sigma^* \to \Delta^*$ is called a *morphism* if h satisfies $h(xy) = h(x)h(y)$ for all $x, y \in \Sigma^*$. A morphism may be specified by providing the values $h(a)$ for all $a \in \Sigma$. This definition is easily extended to (one-sided) infinite words.

A morphism $h : \Sigma^* \to \Sigma^*$ such that $h(a) = ax$ for some $a \in \Sigma$ and $x \in \Sigma^*$ is said to be *prolongable on* a; we may then repeatedly iterate h to obtain the *fixed point*

$$h^\omega(a) = axh(x)h^2(x)h^3(x)\cdots .$$

A morphism is *k-uniform* if $h(a)$ has length k for all $a \in \Sigma$; it is *uniform* if it is k-uniform for some k. A morphism is a *coding* if it is 1-uniform.

Example 6. The *Thue–Morse morphism* is the 2-uniform morphism $\mu : \{0,1\}^* \to \{0,1\}^*$ defined by

$$0 \mapsto 01$$
$$1 \mapsto 10.$$

The fixed point

$$\mathbf{t} = \mu^\omega(0) = 0110100110010110\cdots$$

is known as the *Thue–Morse word*. Observe that this is precisely the characteristic sequence of the Thue–Morse set defined in Example 2.

We have the following alternative characterization of k-recognizable sets.

Theorem 4 (Cobham [14]). *Let $k \geq 2$. A set $X \subseteq \mathbb{N}$ is k-recognizable if and only if its characteristic sequence is of the form $g(h^\omega(a))$, where g is a coding and h is a k-uniform morphism prolongable on the letter a.*

Example 7. In Example 5 we observed that the set of powers of 2 is a 2-recognizable set. Its characteristic sequence can be generated using the 2-uniform morphism

$$h : a \mapsto ab, b \mapsto bc, c \mapsto cc$$

iterated on a and a coding $g : a, c \mapsto 0, b \mapsto 1$. We have

$$h^\omega(a) = abbcbcccbcccccccbcccccccccccccccbcc\cdots$$

and

$$g(h^\omega(a)) = 011010001000000010000000000000000100\cdots.$$

It is also possible to give a logical characterization of k-recognizable sets. The k-recognizable sets are precisely the sets definable in the first order theory $\langle \mathbb{N}, +, V_k \rangle$, where $V_k(n)$ is the largest power of k that divides n. For more information on this logical characterization of k-recognizable sets, see the survey by Bruyère et al. [9].

2.4 k-Recognizable Sets and Number Theory

We end this discussion of k-recognizable sets by giving some number-theoretic applications. In what follows, the notation $\mathbb{F}_p$ denotes the finite field of order p; $\mathbb{F}_p[T]$ denotes the polynomial ring over $\mathbb{F}_p$; and $\mathbb{F}_p(T)$ denotes its field of fractions. The following result shows that p-recognizable sets occur naturally when studying algebraicity over fields of positive characteristic.

Theorem 5 (Christol [12]). *Let X be a set of non-negative integers and let p be a prime. Then X is p-recognizable if and only if the formal power series*

$$\sum_{n \in X} T^n$$

is algebraic over $\mathbb{F}_p(T)$.

Example 8. Let X be the Thue–Morse set and let $\mathbf{t} = t_0 t_1 \cdots$ be its characteristic sequence. We have observed in Example 2 that X is 2-recognizable. Observe

also that $\mathbf{t}$ satisfies the identities $t_{2n} = t_n$ and $t_{2n+1} = t_n + 1$ if we perform the arithmetic over $\mathbb{F}_2$. Consider the formal power series

$$F(T) = \sum_{n \in X} T^n$$

over $\mathbb{F}_2[T]$. We have

$$\begin{aligned} F(T) &= \sum_{n \geq 0} t_n T^n \\ &= \sum_{n \geq 0} t_{2n} T^{2n} + \sum_{n \geq 0} t_{2n+1} T^{2n+1} \\ &= \sum_{n \geq 0} t_n T^{2n} + T \sum_{n \geq 0} (t_n + 1) T^{2n} \\ &= F(T^2) + TF(T^2) + \frac{T}{1 - T^2}. \end{aligned}$$

Since we are working over $\mathbb{F}_2$, we have $F(T^2) = F^2(T)$, whence we obtain

$$(1 + T)^3 F^2(T) + (1 + T)^2 F(T) + T = 0,$$

so that F is algebraic over $\mathbb{F}_2(T)$.

Another remarkable occurrence of p-recognizable sets arises in the theory of linear recurrence sequences over fields of positive characteristic. Let R be a ring and let $U = (U_n)_{n \geq 0}$ be a sequence over R. The sequence U is a *linear recurrence sequence* if there exist $k \geq 1$ and $a_0, \ldots, a_{k-1} \in R$ such that for all $n \geq 0$

$$U_{n+k} = a_{k-1} U_{n+k-1} + \cdots + a_0 U_n.$$

Given a linear recurrence sequence U, its *zero set* is the set $Z(U)$ consisting of all indices n such that $U_n = 0$.

Theorem 6 (Skolem–Mahler–Lech). *Let K be a field of characteristic 0 and let U be a linear recurrence sequence over K. Then $Z(U)$ is a finite union of arithmetic progressions.*

Example 9. Let $A = (A_n)_{n \geq 0}$ be defined by

$$A_n = A_{n-4} + A_{n-2};\ A_0 = A_1 = A_2 = 0, A_3 = 1.$$

Then $Z(A) = \{0, 1, 3, 5, 7, \ldots\}$.

This behaviour fails to hold for linear recurrence sequences over fields of positive characteristic.

Example 10 (Lech). Consider the sequence $B = (B_n)_{n \geq 0}$ over $\mathbb{F}_p(T)$ defined by

$$B_n = (T + 1)^n - T^n - 1.$$

Then B satisfies the linear recurrence

$$B_n - (2+2T)B_{n-1} + (1+3T+3T^2)B_{n-2} - (T+T^2)B_{n-3} = 0$$

over $\mathbb{F}_p(T)$ for $n > 3$. However,

$$B_{p^j} = (T+1)^{p^j} - T^{p^j} - 1 = 0.$$

Since $B_n \neq 0$ when n is not a power of p, we have

$$Z(B) = \{1, p, p^2, p^3, \ldots\},$$

which is clearly not an ultimately periodic set.

Remarkably, it is possible to describe the zero set of a linear recurrence over a field of characteristic p in terms of p-recognizable sets.

Theorem 7 (Derksen). *Let K be a field of characteristic $p > 0$ and let U be a linear recurrence sequence over K. Then $Z(U)$ is a p-recognizable set.*

Adamczewski and Bell [1] have recently given a generalization of this result.

3 Linear Numeration Systems

3.1 Basic Definitions and Results

In the classical integer base numeration systems, we represent the integers as a sum of elements of the sequence $(k^n)_{n\geq 0}$ of powers of a fixed base k. We now generalize this idea by considering an arbitrary increasing sequence of integers $U = (U_n)_{n\geq 0}$ as a basis for representing the integers. As we shall see, the only sequences U of interest to us are those defined by a linear recurrence relation.

Let $U = (U_n)_{n\geq 0}$ be an increasing sequence of integers with $U_0 = 1$ and

$$C_U := \sup_{n\geq 0} \left\lceil \frac{U_{n+1}}{U_n} \right\rceil < \infty.$$

A *greedy representation* of a non-negative integer n is a word $w = w_{\ell-1}\cdots w_0$ over $\{0, 1, \ldots, C_U - 1\}$ such that

$$\sum_{i=0}^{\ell-1} w_i U_i = n,$$

and for all j

$$\sum_{i=0}^{j-1} w_i U_i < U_j.$$

The greedy representation of n with $w_{\ell-1} \neq 0$ is denoted by $[n]_U$. A set X of non-negative integers is *U-recognizable* if $[X]_U := \{[x]_U : x \in X\}$ is accepted by a finite automaton.

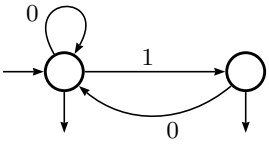


Fig. 1. The automaton for the Fibonacci numeration system

Example 11 (Fibonacci numeration system). Let $U = (U_n)_{n\geq 0}$ be the sequence of *Fibonacci numbers* defined by $U_{n+2} = U_{n+1} + U_n$ and $U_0 = 1$, $U_1 = 2$. Then

$$13 = 1 \cdot 13 + 0 \cdot 8 + 0 \cdot 5 + 0 \cdot 3 + 0 \cdot 2 + 0 \cdot 1,$$

so the representation of 13 in the Fibonacci system is $[13]_U = 100000$. Note however that 13 also has the non-greedy representation 11000. Indeed the language $0^*[\mathbb{N}]_U$ is the language of all words over $\{0, 1\}$ that do not contain the factor 11. The automaton accepting this language is given in Figure 1.

A numeration system $U = (U_n)_{n\geq 0}$ is said to be *linear* if it satisfies a linear recurrence over $\mathbb{Z}$. The importance of linear numeration systems is shown by the following result.

Theorem 8 (Shallit [37]). *If $\mathbb{N}$ is U-recognizable, then U is linear.*

Proof (Loraud [27]). Suppose that $[\mathbb{N}]_U$ is a regular language. Then $0^*[\mathbb{N}]_U$ is also a regular language. Let r_n be the number of words of length n in $[\mathbb{N}]_U$ and let s_n be the number of words of length n in $0^*[\mathbb{N}]_U$. Then $(s_n)_{n\geq 0}$ is a linear recurrence sequence (this is a well-known result for regular languages). However, we have

$$s_n = \sum_{i=0}^{n} r_i = U_n,$$

so U is linear, as required. □

The converse of this theorem is not true in general.

Example 12 (Shallit [37]). Let U be the sequence given by $U_n = (n+1)^2$ for $n \geq 0$. Then $U_0 = 1$, $U_1 = 4$, $U_2 = 9$, and U satisfies the linear recurrence

$$U_{n+3} = 3U_{n+2} - 3U_{n+1} + U_n.$$

Suppose that $[\mathbb{N}]_U$ were a regular language. Then the language

$$[\mathbb{N}]_U \cap 10^*10^* = \{10^a10^b \in \{0, 1\}^* : b^2 < 2a + 4\}$$

would also be regular; however, one easily shows using the pumping lemma that this is not the case.

3.2 Bertrand Systems

In general, the properties of an arbitrary linear numeration system can often be difficult to analyze. We therefore now restrict our attention to a particular class of linear numeration systems that are somewhat easier to study. Let U be a linear numeration system. If

$$\lim_{n\to\infty} \frac{U_{n+1}}{U_n} = \beta$$

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.

The properties of linear numeration systems with a dominant root β are often linked with the properties of the so-called β-expansions of real numbers. Let $\beta > 1$ be a real number. The *β-expansion* of a real number $x \in [0, 1]$ is the sequence $d_\beta(x) = (x_i)_{i\geq 1} \in \mathbb{N}^\omega$ that satisfies

$$x = \sum_{i=1}^{\infty} x_i \beta^{-i}$$

and is the lexicographically largest sequence having this property. If

$$d_\beta(1) = t_1 \cdots t_m 0^\omega,$$

with $t_m \neq 0$, then we say that $d_\beta(1)$ is *finite* and we set

$$d^*_\beta(1) = (t_1 \cdots t_{m-1}(t_m - 1))^\omega.$$

Otherwise, we set $d^*_\beta(1) = d_\beta(1)$. If $d^*_\beta(1)$ is ultimately periodic, then β is a *Parry number*.

The next theorem gives a necessary condition for $\mathbb{N}$ to be U-recognizable when U is a linear numeration system with a dominant root β.

Theorem 9 (Hollander [20]). *Let U be a linear numeration system with dominant root β. If $\mathbb{N}$ is U-recognizable, then β is a Parry number.*

In general it is a difficult problem to characterize the linear numeration systems U for which $\mathbb{N}$ is U-recognizable. Hollander proved a much stronger result than the one quoted above: he gave sufficient conditions for $\mathbb{N}$ to be U-recognizable in the case where U has a dominant root $\beta > 1$. It remains an open problem to give a general characterization of all linear numeration systems U for which $\mathbb{N}$ is U-recognizable. Hollander gave a conjectural characterization at the end of his paper, but this conjecture is still open.

Next we study a particular class of linear numeration systems U for which $\mathbb{N}$ is always U-recognizable. A numeration system $U = (U_n)_{n\geq 0}$ is a *Bertrand numeration system* if it has the following property:

$$\text{a word } w \text{ is in } [\mathbb{N}]_U \text{ if and only if } w0 \text{ is in } [\mathbb{N}]_U.$$

Example 13. For any integer $k > 2$, the integer base-k numeration system is a Bertrand numeration system. The Fibonacci system of Example 11 is also a Bertrand numeration system. On the other hand, if we change the initial conditions of the Fibonacci recurrence $U_{n+2} = U_{n+1} + U_n$ to $U_0 = 1$, $U_1 = 3$, we obtain a system that is no longer a Bertrand numeration system, since the greedy representation of the number 2 is the word 2, but the greedy representation of the number 6 is the word 102, not the word 20.

Let $\mathrm{Fact}(D_\beta)$ denote the set of factors occurring in the β-expansions of the real numbers in $[0, 1)$.

Theorem 10 (Bertrand [7]). *Let $U = (U_n)_{n\geq 0}$ be a numeration system. There exists a real number $\beta > 1$ such that $0^*[\mathbb{N}]_U = \mathrm{Fact}(D_\beta)$ if and only if U is a Bertrand numeration system. In this case, if $d^*_\beta(1) = (t_i)_{i\geq 1}$, then*

$$U_n = t_1 U_{n-1} + \cdots + t_n U_0 + 1. \tag{1}$$

If β is a Parry number, then (1) defines a linear recurrence sequence and β is a root of its characteristic polynomial.

Theorem 11 (Parry [31]). *A sequence $s = (s_i)_{i\geq 1}$ over $\mathbb{N}$ is the β-expansion of a real number in $[0, 1)$ if and only if $(s_{n+i})_{i\geq 1}$ is lexicographically less than $d^*_\beta(1)$ for all $n \geq 1$.*

As a consequence of Theorems 10 and 11, every Parry number β has an associated *canonical numeration system*. The language of the canonical numeration system associated with β is accepted by the deterministic finite automaton $\mathcal{A}_\beta$ accepting the language $\mathrm{Fact}(D_\beta)$. This automaton is defined as follows. Let

$$d^*_\beta(1) = t_1 \cdots t_i (t_{i+1} \cdots t_{i+p})^\omega,$$

where $i \geq 0$ and $p \geq 1$ are the minimal preperiod and period respectively. The set of states of $\mathcal{A}_\beta$ is $Q_\beta = \{q_{\beta,0}, \ldots, q_{\beta,i+p-1}\}$. All states are final. For every $j \in \{1, \ldots, i+p\}$, we have t_j transitions $q_{\beta,j-1} \to q_{\beta,0}$ labeled by $0, \ldots, t_j - 1$ and, for $j < i + p$, we have one transition $q_{\beta,j-1} \to q_{\beta,j}$ labeled by t_j. There is also a transition $q_{\beta,i+p-1} \to q_{\beta,i}$ labeled by t_{i+p}. See, for instance, [16,18,23].

Example 14. Let β be the dominant root of the polynomial $X^3 - 2X^2 - 1$. We have $d_\beta(1) = 2010^\omega$ and $d^*_\beta(1) = (200)^\omega$. The automaton $\mathcal{A}_\beta$ is depicted in Figure 2.

3.3 Pisot Systems

We have seen that for every Parry number β there is an associated Bertrand numeration system U such that $\mathbb{N}$ is U-recognizable. Now we consider linear numeration systems U whose dominant root is a Pisot number. We shall see that for such systems the set $\mathbb{N}$ is always U-recognizable. A *Pisot number* is a real algebraic integer greater than one such that all of its algebraic conjugates have absolute value less than one.

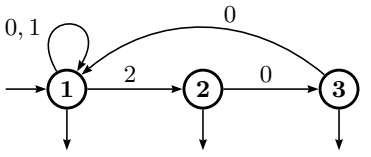


Fig. 2. The automaton $\mathcal{A}_\beta$ for $d^*_\beta(1) = (200)^\omega$

Theorem 12 (Bertrand [6]; Schmidt [36]). *Every Pisot number is a Parry number.*

In view of this result and the discussion above, every Pisot number β has an associated Bertrand numeration system U in which $\mathbb{N}$ is U-recognizable. However, there may be many other linear recurrence sequences U that also have dominant root β yet are not Bertrand systems. Indeed, there will be one choice of initial values for the recurrence defining U that will result in a Bertrand system, but other choices of initial values will result in systems that are not Bertrand systems. Let us call any linear numeration system whose characteristic polynomial is the minimal polynomial of a Pisot number a *Pisot system.*

Example 15. The Fibonacci numeration system of Example 11 is a Pisot system, since the characteristic polynomial $x^2 - x - 1$ of the defining linear recurrence is the minimal polynomial of the Pisot number $(1 + \sqrt{5})/2$.

The principal result concerning Pisot systems is the following.

Theorem 13 (Frougny and Solomyak [18]; Bruyère and Hansel [8]). *Let U be a Pisot system. Then $\mathbb{N}$ is U-recognizable.*

3.4 Further Discussion

As was the case for the integer base numeration systems, it is also possible to give a morphic characterization as well as a logical characterization of the U-recognizable sets for the numeration systems U considered in this section. For more information, see Fabre [16] and Bruyère and Hansel [8].

In Section 2.2 we mentioned that the periodicity problem is decidable for base-k numeration systems. In the setting of linear numberation systems, the problem is as follows. Let U be a linear numeration system such that $\mathbb{N}$ is U-recognizable. Given a automaton accepting $[X]_U$ for some set X, determine if X is ultimately periodic. It is currently unknown if this problem is decidable. For Pisot systems the problem is decidable; for example, the proofs of Muchnik [28] or Allouche et al. [2] can be easily adapted to these systems. Charlier et al. [4] showed that the periodicity problem is decidable for a large class of linear numeration systems, but the general problem remains open.

4 Abstract Numeration Systems

4.1 Basic Definitions and Results

Abstract numerations systems were first introduced by Lecomte and Rigo [25]. They generalize linear numeration systems and include them as a special case. In the case of linear numeration systems, we typically begin with a given linear recurrence sequence and look for conditions under which this sequence results in a regular numeration language. In the case of abstract numeration systems, we change our point of view slightly: instead, we consider any fixed regular language as being a possible numeration language. Of course, the resulting numeration system is unlikely to be positional in general; instead, we order our numeration language genealogically and define the representation of n to be the $(n+1)$-th word of the language.

Let u and v be two finite words of the same length (resp. two infinite words) over an alphabet $A \subset \mathbb{N}$. We say that u is *lexicographically less* than v if we can write $u = pau'$ and $v = pbv'$, where a and b are letters such that a is less than b. If u and v are two finite words (not necessarily of the same length), then u is *genealogically less* than v if either $|u| < |v|$ or $|u| = |v|$ and u is lexicographically less than v.

An *abstract numeration system* is a triple $S = (L, \Sigma, <)$ where L is an infinite regular language over a totally ordered finite alphabet $(\Sigma, <)$. The language L is called the *numeration language*. The map $[\cdot]_S \colon \mathbb{N} \to L$ is a bijection mapping $n \in \mathbb{N}$ to the $(n+1)$-th word of L ordered genealogically. The inverse map is denoted by $\langle \cdot \rangle_S \colon L \to \mathbb{N}$. A set $X \subseteq \mathbb{N}$ is *S-recognizable* if the language $[X]_S = \{[n]_S \colon n \in X\}$ is regular.

Example 16. The base-k numeration systems are all examples of abstract numeration systems. For each k, the defining numeration language is the language

$$\{0, 1, \ldots, k-1\}^* \setminus (0\{0, 1, \ldots, k-1\}^*).$$

Similarly, the Fibonacci numeration system of Example 11 is also an abstract numeration system, as are the Pisot systems discussed in the previous section.

Example 17. Recall from Example 3 that the set $\{n^2 : n \in \mathbb{N}\}$ of squares is not k-recognizable for any k. However, the set of squares is S-recognizable for the abstract numeration system

$$S = (a^*b^* \cup a^*c^*, \{a, b, c\}, a < b < c),$$

since the language of representations of the squares is the regular language a^*.

If fact, we have the following general result concerning polynomial sequences.

Theorem 14 (Rigo [34]; Strogalov [38]). *For any polynomial $P \in \mathbb{Q}[x]$ such that $P(\mathbb{N}) \subseteq \mathbb{N}$, there exists S such that P is S-recognizable.*

The following result, which shows that Cobham's characterization of the sets recgonizable in all integer bases carries through in the more general setting of abstract numeration systems, provides further confirmation that the notion of abstract numeration system is indeed a natural generalization of the integer base numeration systems.

Theorem 15 (Lecomte and Rigo [25]). *Let S be an abstract numeration system. Then every ultimately periodic set is S-recognizable.*

Krieger et al. [22] reproved this result with an explicit bound on the size of the automaton accepting an ultimately periodic set. The same construction was independently found by Angrand and Sakarovitch [3].

Theorem 16 (Krieger et al. [22]). *Let m and r be integers with $m \geq 2$ and $0 \leq r \leq m-1$. If S is an abstract numeration system whose language L is accepted by an n-state deterministic finite automaton, then the minimal deterministic finite automaton accepting the language $[m\mathbb{N} + r]_S$ has at most nm^n states.*

The construction used to establish this result can also be used to prove the following theorem, originally due to Choffrut and Goldwurm [11] and later reproved by Rigo [33] (see [3]). (For the definition of $\mathbb{N}$-rational series, see Berstel and Reutenauer [5].)

Theorem 17 (Choffrut and Goldwurm [11]). *Let $S = (L, \Sigma, <)$ be an abstract numeration system. The formal series*

$$\sum_{w \in L} \langle w \rangle_S w$$

is $\mathbb{N}$-rational.

4.2 Alternative Characterizations

In Section 2.3 we gave an equivalent characterization of k-recognizable sets in terms of uniform morphisms. We have a similar characterization for S-recognizable sets, but now the morphisms are no longer uniform. We say that a sequence is *morphic* if it is of the form $g(h^\omega(a))$, where g is a coding and h is a non-erasing morphism prolongable on the letter a.

Theorem 18 (Rigo [32]; Rigo and Maes [35]). *Let $X \subseteq \mathbb{N}$. Then there exists an abstract numeration system S such that X is S-recognizable if and only if the characteristic sequence of X is morphic.*

Example 18. Recall from Example 17 that the set of squares is S-recognizable for the abstract numeration system

$$S = (a^*b^* \cup a^*c^*, \{a, b, c\}, a < b < c).$$

The characteristic sequence of the set of squares can be generated using the non-uniform morphism

$$h : a \mapsto abcc, b \mapsto bcc, c \mapsto c$$

iterated on a and a coding $g : a, b \mapsto 1, c \mapsto 0$. We have

$$h^\omega(a) = abccbccccbcccccbccccccc \cdots$$

and

$$g(h^\omega(a)) = 1100100001000000100000000 \cdots .$$

4.3 Further Discussion

We end by revisiting the periodicity problem in the general setting of abstract numeration systems. The problem is as follows. Let S be an abstract numeration system. Given an automaton accepting $[X]_S$ for some set X, determine if X is ultimately periodic. As mentioned in Section 3.4, the decidability of this problem is still an open question even in the special case of linear numeration systems, so evidently the problem remains open in the general setting. The periodicity problem becomes particularly interesting in the general setting of abstract numeration systems because of its equivalence to a longstanding open problem in the theory of D0L systems. Theorem 18 gives an equivalence between S-recognizable sets and morphic sequences. A set X is ultimately periodic if and only if its characteristic sequence is ultimately periodic, that is, if and only if its characteristic sequence can be written in the form $uvvvv \cdots$, where u and v are words. However, the decidability of the periodicity problem for morphic sequences is a longstanding open problem in combinatorics on words. If we consider only *purely morphic sequences*, that is, sequences of the form $h^\omega(a)$, where h is a morphism prolongable on the letter a, then Harju and Linna [19] and Pansiot [30] showed that the periodicity problem is decidable, but the question remains unsolved for morphic sequences in general. Hence, the periodicity problem for abstract numeration systems appears to be a difficult one.

There is much more to be said about abstract numeration systems. The theory of such systems has been extensively developed, in particular by Rigo and his co-authors. For more information on this topic, one may consult the survey by Lecomte and Rigo [24].

References

1. Adamczewski, B., Bell, J.: On vanishing coefficients of algebraic power series over fields of positive characteristic (preprint)
2. Allouche, J.P., Rampersad, N., Shallit, J.: Periodicity, repetitions, and orbits of an automatic sequence. Theoret. Comput. Sci. 410, 2795–2803 (2009)
3. Angrand, P.-Y., Sakarovitch, J.: The enumerating series of an abstract numeration system (preprint)

4. Bell, J., Charlier, E., Fraenkel, A.S., Rigo, M.: A decision problem for ultimately periodic sets in non-standard numeration systems. Internat. J. Algebra Comput. 19(6), 809–839 (2009)
5. Berstel, J., Reutenauer, C.: Rational series and their languages. EATCS Monographs on Theoretical Computer Science, vol. 12. Springer, Berlin (1988)
6. Bertrand, A.: Développements en base de Pisot et répartition modulo 1. C. R. Acad. Sci. Paris Sér. A-B 285, A419–A421 (1977)
7. Bertrand-Mathis, A.: Comment écrire les nombres entiers dans une base qui n'est pas entière. Acta Math. Hungar. 54(3-4), 237–241 (1989)
8. Bruyère, V., Hansel, G.: Bertrand numeration systems and recognizability. Theoret. Comput. Sci. 181(1), 17–43 (1997); Latin American Theoretical Informatics (Valparaíso, 1995)
9. Bruyère, V., Hansel, G., Michaux, C., Villemaire, R.: Logic and p-recognizable sets of integers. Bull. Belg. Math. Soc. Simon Stevin 1(2), 191–238 (1994); Journées Montoises (Mons, 1992)
10. Charlier, E., Rampersad, N.: The growth function of S-recognizable sets (submitted)
11. Choffrut, C., Goldwurm, M.: Rational transductions and complexity of counting problems. Math. Sci. Theory 28, 437–450 (1995)
12. Christol, G., Kamae, T., Mendès France, M., Rauzy, G.: Suites algébriques, automates et substitutions. Bull. Soc. Math. France 108(4), 401–419 (1980)
13. Cobham, A.: On the base-dependence of sets of numbers recognizable by finite automata. Math. Systems Theory 3, 186–192 (1969)
14. Cobham, A.: Uniform tag sequences. Math. Systems Theory 6, 164–192 (1972)
15. Eilenberg, S.: Automata, Languages, and Machines. Pure and Applied Mathematics A, 58 (1974)
16. Fabre, S.: Une généralisation du théorème de Cobham. Acta Arith. 67(3), 197–208 (1994)
17. Fagnot, I.: Sur les facteurs des mots automatiques. Theoret. Comput. Sci. 172, 67–89 (1997)
18. Frougny, C., Solomyak, B.: On the representation of integers in linear numeration systems. In: Ergodic Theory of Z_d Actions (Warwick, 1993–1994). London Math. Soc. Lecture Note Ser., vol. 228, pp. 345–368. Cambridge Univ. Press, Cambridge (1996)
19. Harju, T., Linna, M.: On the periodicity of morphisms on free monoids. RAIRO Inform. Théor. Appl. 20(1), 47–54 (1986)
20. Hollander, M.: Greedy numeration systems and regularity. Theory Comput. Syst. 31(2), 111–133 (1998)
21. Honkala, J.: A decision method for the recognizability of sets defined by number systems. RAIRO Inform. Theor. Appl. 20(4), 395–403 (1986)
22. Krieger, D., Miller, A., Rampersad, N., Ravikumar, B., Shallit, J.: Decimations of languages and state complexity. Theoret. Comput. Sci. 410(24-25), 2401–2409 (2009)
23. Lecomte, P., Rigo, M.: Real numbers having ultimately periodic representations in abstract numeration systems. Inform. and Comput. 192 (2004)
24. Lecomte, P., Rigo, M.: Abstract numeration systems. In: Berthé, V., Rigo, M. (eds.) Combinatorics, Automata, and Number Theory. Encyclopedia of Mathematics and its Applications, vol. 135, Cambridge Univ. Press, Cambridge (2010)
25. Lecomte, P.B.A., Rigo, M.: Numeration systems on a regular language. Theory Comput. Syst. 34(1), 27–44 (2001)

26. Leroux, J.: A polynomial time Presburger criterion and synthesis for number decision diagrams. In: 20th IEEE Symposium on Logic in Computer Science, pp. 147–156. IEEE Computer Society, Chicago (2005)
27. Loraud, N.: β-shift, systèmes de numération et automates. J. Théor. Nombres Bordeaux 7(2), 473–498 (1995)
28. Muchnick, A.A.: The definable criterion for definability in Presburger arithmetic and its applications. Theoret. Comput. Sci. 290(3), 1433–1444 (2003)
29. Nathanson, M.: Additive number theory: the classical bases. Springer, Berlin (1996)
30. Pansiot, J.-J.: Decidability of periodicity for infinite words. RAIRO Inform. Théor. Appl. 20(1), 43–46 (1986)
31. Parry, W.: On the β-expansions of real numbers. Acta Math. Acad. Sci. Hung. 11, 401–416 (1960)
32. Rigo, M.: Generalization of automatic sequences for numeration systems on a regular language. Theoret. Comput. Sci. 244(1-2), 271–281 (2000)
33. Rigo, M.: Numeration systems on a regular language: Arithmetic operations, recognizability and formal power series. Theoret. Comput. Sci. 269(1-2), 469–498 (2001)
34. Rigo, M.: Construction of regular languages and recognizability of polynomials. Discrete Math. 254(1-3), 485–496 (2002)
35. Rigo, M., Maes, A.: More on generalized automatic sequences. J. Autom., Lang. and Comb. 7(3), 351–376 (2002)
36. Schmidt, K.: On periodic expansions of Pisot numbers and Salem numbers. Bull. London Math. Soc. 12, 269–278 (1980)
37. Shallit, J.: Numeration systems, linear recurrences, and regular sets. Inform. and Comput. 113(2), 331–347 (1994)
38. Strogalov, A.S.: Regular languages with polynomial growth in the number of words. Diskret. Mat. 2(3), 146–152 (1990)

Rule Formats for Distributivity*

Luca Aceto[1], Matteo Cimini[1], Anna Ingolfsdottir[1],
Mohammad Reza Mousavi[2], and Michel A. Reniers[3]

[1] ICE-TCS, School of Computer Science, Reykjavik University,
Menntavegur 1, IS 101 Reykjavik, Iceland
[2] Department of Computer Science, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
[3] Department of Mechanical Engineering, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

Abstract. This paper proposes rule formats for Structural Operational Semantics guaranteeing that certain binary operators are left distributive with respect to a set of binary operators. Examples of left-distributivity laws from the literature are shown to be instances of the provided formats.

1 Introduction

Over the last three decades, Structural Operational Semantics (SOS), see, e.g., [7,20,22], has proven to be a powerful way to specify the semantics of programming and specification languages. In this approach to semantics, languages can be given a clear behaviour in terms of states and transitions, where the collection of transitions is specified by means of a set of syntax-driven inference rules. This behavioural description of the semantics of a language essentially tells one how the expressions in the language under definition behave when run on an idealized abstract machine.

Designers of languages often have expected algebraic properties of language constructs in mind when defining a language. For example, one expects that a sequential composition operator be associative and, in the field of process algebra [11,16,17], operators such as nondeterministic and parallel composition are often meant to be commutative and associative with respect to bisimilarity. Once the semantics of a language has been given in terms of state transitions, a natural question to ask is whether the intended algebraic properties do hold modulo the notion of behavioural equivalence or preorder of interest. The typical approach to answer this question is to perform an *a posteriori verification*: based on the semantics in terms of state transitions, one proves the validity of the desired algebraic

* The work of Aceto, Cimini and Ingolfsdottir has been partially supported by the projects 'New Developments in Operational Semantics' (nr. 080039021) and 'Metatheory of Algebraic Process Theories' (nr. 100014021) of the Icelandic Research Fund. The work on the paper was partly carried out while Luca Aceto held an Abel Extraordinary Chair at Universidad Complutense de Madrid, Spain, supported by the NILS Mobility Project.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 80–91, 2011.

laws, which describe semantic properties of the various operators in the language. An alternative approach is to ensure the validity of algebraic properties *by design*, using the so called *SOS rule formats* [2]. In this approach, one gives *syntactic templates* for the inference rules used in defining the operational semantics for certain operators that guarantee the validity of the desired laws by design. Not surprisingly, the definition of rule formats is based on finding a reasonably good trade-off between generality and ease of application. On the one hand, one strives to define a rule format that can capture as many examples from the literature as possible, including ones that may arise in the future. On the other, the rule format should be as easy to apply as possible and, preferably, the syntactic constraints of the format should be algorithmically checkable.

The literature on SOS provides rule formats for basic algebraic properties of operators such as commutativity [19], associativity [15], idempotence [3] and the existence of unit and zero elements [5,8]. The main advantage of this approach is that one is able to verify the desired property by syntactic checks that can be mechanized. Moreover, it is interesting to use rule formats for establishing semantic properties since the results so obtained apply to a broad class of languages. Apart from providing one with an insight as to the semantic nature of algebraic properties and its link to the syntax of SOS rules, rule formats like those presented in the above-mentioned references may serve as a guideline for language designers who want to ensure, a priori, that the constructs under design enjoy certain basic algebraic properties.

In the present paper, we develop two rule formats guaranteeing that certain binary operators are left distributive with respect to others modulo bisimilarity. A binary operator $\circledast$ is *left distributive* with respect to a binary operator $\oplus$, modulo some notion of behavioural equivalence, whenever the equation $(x \oplus y) \circledast z = (x \circledast z) \oplus (y \circledast z)$ holds.

A classic example of left-distributivity law within the realm of process algebra is $(x + y) \between z = (x \between z) + (y \between z)$, where '+' and '$\between$' stand for nondeterministic choice and left merge, respectively, from [11,17]. (The reader may find many other examples in the main body of this paper.) Distributivity laws like the aforementioned one play a crucial role in (ground-)complete axiomatizations of behavioural equivalences over fragments of process algebras (see, e.g., the above-mentioned references and [4]), and their lack of validity with respect to choice-like operators is often the key to the nonexistence of finite (in)equational axiomatizations of behavioural semantics—see, for instance, [6,18].

In the rule formats, for the sake of simplicity, the $\oplus$ operator 'behaves like' some form of nondeterministic choice operator. Both rule formats are based on syntactic conditions that are decidable over finite language specifications.

We provide a wealth of examples showing that the validity of several left-distributivity laws from the literature on process algebras can be proved using the proposed rule formats.

Roadmap of the paper. The paper is organized as follows. Section 2 reviews some standard definitions from the theory of SOS that will be used in the remainder of this study. Section 3 presents our first rule format guaranteeing that a

binary operator $\circledast$ is left-distributive with respect to a binary operator $\oplus$ modulo bisimilarity. The rule format is defined in Section 3.2 and some examples of its application are given in Section 3.3. We extend our rule format in Section 4 by allowing for a wider set of terms appearing in the target of deduction rules. Examples that can be handled using the second rule format are offered in the same section. We refer the reader to [1] for proofs and further results.

2 Preliminaries

In this section we recall some standard definitions from the theory of SOS. We refer the readers to, e.g., [7] and [20] for more information.

2.1 Transition System Specifications and Bisimilarity

Definition 1 (Signatures, terms and substitutions). *We let V denote an infinite set of variables and use $x, x', x_i, y, y', y_i, \ldots$ to range over elements of V. A* signature *Σ is a set of function symbols, each with a fixed arity. We call these symbols* operators *and usually represent them by $f, g, \ldots$. An operator with arity zero is called a* constant*. We define the set $\mathbb{T}(\Sigma)$ of* terms *over Σ as the smallest set satisfying the following constraints.*

- *A variable $x \in V$ is a term.*
- *If $f \in \Sigma$ has arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.*

We use s, t, u, possibly subscripted and/or superscripted, to range over terms. We write $t_1 \equiv t_2$ if t_1 and t_2 are syntactically equal. The function $\mathrm{vars} : \mathbb{T}(\Sigma) \to 2^V$ *gives the set of variables appearing in a term. The set $\mathbb{C}(\Sigma) \subseteq \mathbb{T}(\Sigma)$ is the set of* closed terms*, i.e., terms that contain no variables. We use $p, q, p', p_i, \ldots$ to range over closed terms. A* substitution *σ is a function of type $V \to \mathbb{T}(\Sigma)$. We extend the domain of substitutions to terms homomorphically and write $\sigma(t)$ for the result of applying the substitution σ to the term t. If the range of a substitution is included in $\mathbb{C}(\Sigma)$, we say that it is a* closed substitution*. For a sequence $x_1, \ldots, x_n$ of distinct variables and a sequence $t_1, \ldots, t_n$ of terms, we write $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ for a substitution that maps each x_i to t_i, $1 \leq i \leq n$.*

Definition 2 (Transition system specification). *A* transition system specification *(TSS) is a triple $(\Sigma, \mathcal{L}, D)$ where*

- *Σ is a signature.*
- *$\mathcal{L}$ is a set of* labels *(or* actions*) ranged over by a, b, l. If $l \in \mathcal{L}$ and $t, t' \in \mathbb{T}(\Sigma)$, we say that $t \xrightarrow{l} t'$ is a* positive transition formula *and $t \overset{l}{\nrightarrow}$ is a* negative transition formula*. Such formulae are called* t-testing*. A transition formula (or just formula), typically denoted by ϕ or ψ, is either a negative transition formula or a positive one.*
- *D is a set of* deduction rules*, i.e., tuples of the form (Φ, ϕ) where Φ is a set of formulae and ϕ is a positive formula. We call the formulae contained in Φ the* premises *of the rule and ϕ the* conclusion*.*

We write $\mathrm{vars}(\Phi)$ *to denote the set of variables appearing in a set of formulae* Φ*. We say that a formula or a deduction rule is* closed *if all of its terms are closed. Substitutions are also extended to formulae and sets of formulae in the natural way. A set of positive closed formulae is called a* transition relation*.*

We often refer to a positive transition formula $t \xrightarrow{l} t'$ as a *transition* with t being its *source*, l its *label*, and t' its *target*. A deduction rule (Φ, ϕ) is typically written as $\frac{\Phi}{\phi}$. For the sake of consistency with SOS specifications of specific operators in the literature, in examples we use $\frac{\phi_1 \ldots \phi_n}{\phi}$ in lieu of $\frac{\{\phi_1,\ldots,\phi_n\}}{\phi}$.

An *axiom* is a deduction rule with an empty set of premises. We write $\frac{}{\phi}$ for an axiom with ϕ as its conclusion, and often abbreviate this notation to ϕ when this causes no confusion.

Definition 3. *Given a rule* d *of the form* $\frac{\Phi}{f(t_1,\ldots,t_n) \xrightarrow{a} t}$*, we say that* d *is* f-defining*, and write* $\mathrm{op}(d) = f$*,* d *is* a-emitting*, and* $\mathrm{toc}(d) = t$*, the* target *of the* conclusion *of* d*. We also denote by* $D(f, a)$ *the set of* a*-emitting and* f*-defining rules in a set of deduction rules* D*.*

Example 1 (Choice operators). The choice operator from [17] is defined by the following rules, where a ranges over the set of actions:

$$(chl_a)\ \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad (chr_a)\ \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}\ .$$

For each action a, the rules (chl_a) and (chr_a) are a-emitting and $+$-defining. For rule (chl_a), we have that $\mathrm{toc}(chl_a) = x'$.

The meaning of a TSS is defined by the notion of least three-valued stable model [23]. We write $\mathcal{T} \vdash p \xrightarrow{a} p'$ if the transition $p \xrightarrow{a} p'$ is in the least three-valued stable model of $\mathcal{T}$. Since the precise definition of this notion does not play a role in the remainder of this paper, we omit it for the sake of brevity and refer our readers to [1] for details.

Definition 4 (Bisimulation and bisimilarity). *Let* $\mathcal{T}$ *be a transition system specification with signature* Σ *and label set* $\mathcal{L}$*. A relation* $\mathcal{R} \subseteq \mathbb{C}(\Sigma) \times \mathbb{C}(\Sigma)$ *is a* bisimulation relation *if and only if* $\mathcal{R}$ *is symmetric and, for all* $p_0, p_1, p_0' \in \mathbb{C}(\Sigma)$ *and* $l \in \mathcal{L}$*,*

$$(p_0 \,\mathcal{R}\, p_1 \wedge \mathcal{T} \vdash p_0 \xrightarrow{l} p_0') \Rightarrow \exists p_1' \in \mathbb{C}(\Sigma).\ (\mathcal{T} \vdash p_1 \xrightarrow{l} p_1' \wedge p_0' \,\mathcal{R}\, p_1').$$

Two terms $p_0, p_1 \in \mathbb{C}(\Sigma)$ *are called* bisimilar*, denoted by* $p_0 \leftrightarroweq p_1$*, when there exists a bisimulation relation* $\mathcal{R}$ *such that* $p_0 \,\mathcal{R}\, p_1$*.*

Bisimilarity is extended to open terms by requiring that $s, t \in \mathbb{T}(\Sigma)$ are bisimilar when $\sigma(s) \leftrightarroweq \sigma(t)$ for each closed substitution $\sigma : V \to \mathbb{C}(\Sigma)$.

3 The Left-Distributivity Rule Formats

In this section, we present a rule format guaranteeing that a binary operator $\circledast$ is left-distributive with respect to a binary operator $\oplus$ modulo bisimilarity. The rule format suffices to handle many examples from the literature.

Definition 5 (Left-distributivity law). *We say that a binary operator $\circledast$ is* left-distributive *with respect to a binary operator $\oplus$ (modulo bisimilarity) if the following equality holds:*

$$(x \oplus y) \circledast z \leftrightarrows (x \circledast z) \oplus (y \circledast z). \tag{1}$$

For all closed terms p, q, r, proving the algebraic law (1) involves two proof obligations:

- **Firability:** ensuring that $(p \oplus q) \circledast r \xrightarrow{a}$ if, and only if, $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a}$, for each action a;
- **Matching conclusions:** ensuring that, for each closed term p_1, if $(p \oplus q) \circledast r \xrightarrow{a} p_1$, then there exists some closed term p_2 such that $(p \circledast r) \oplus (q \circledast r) \xrightarrow{a} p_2$ and $p_1 \leftrightarrows p_2$, and vice versa.

3.1 The Firability Condition

We begin by introducing the conditions on sets of rules for two binary operators $\circledast$ and $\oplus$ that we shall use to guarantee the firability condition for them. First of all, we present syntactic constraints on the rules for those operators that we shall use throughout the remainder of the paper.

Definition 6. *We say that a deduction rule is of* the form (R1) *when it has the structure*

$$\frac{\Phi_y}{x \circledast y \xrightarrow{a} t} \quad \text{or} \quad \frac{\{x \xrightarrow{a} x'\} \cup \Phi_y}{x \circledast y \xrightarrow{a} t},$$

where

- *the variables x, x', y are pairwise distinct, and*
- *Φ_y is a (possibly empty) set of (positive or negative) y-testing formulae such that $x, x' \notin \mathrm{vars}(\Phi_y)$.*

A deduction rule is of the form (R2) *when it has the structure*

$$\frac{\{x \xrightarrow{a} x'\}}{x \oplus y \xrightarrow{a} t} \quad \text{or} \quad \frac{\{y \xrightarrow{a} y'\}}{x \oplus y \xrightarrow{a} t} \quad \text{or} \quad \frac{\{x \xrightarrow{a} x', y \xrightarrow{a} y'\}}{x \oplus y \xrightarrow{a} t},$$

where the variables x, x', y, y' are pairwise distinct. A rule of the form $(R1)$ or $(R2)$ is non-left-inheriting *if $x \notin \mathrm{vars}(t)$, that is, if x does not appear in the target of the conclusion of the rule. An operation f specified by rules of the form $(R1)$ or $(R2)$ is* non-left-inheriting *if so are all of the f-defining rules.*

Definition 7 (Firability constraint). *Given a TSS T, let $\circledast$ and $\oplus$ be binary operators in the signature of T. For each action a, we write* $\mathrm{Fire}(\circledast, \oplus, a)$ *whenever the following conditions are met:*

- *if $D(\circledast, a) \neq \emptyset$ then $D(\oplus, a) \neq \emptyset$,*
- *each $d \in D(\circledast, a)$ is of the form $(R1)$, and*
- *each $d \in D(\oplus, a)$ is of the form $(R2)$.*

Example 2. Recall the choice operator $+$, presented in Example 1. As our readers can easily check, $\mathrm{Fire}(+, +, a)$ holds for each action a.

The firability constraint in Definition 7 is sufficient to guarantee the aforementioned firability condition.

Theorem 1 (Firability Theorem). *Given a TSS T, let $\circledast$ and $\oplus$ be binary operators from the signature of T. Suppose that* $\mathrm{Fire}(\circledast, \oplus, a)$ *holds for some action a. Then,*

$$(p \oplus q) \circledast r \xrightarrow{a} \text{ if, and only if, } (p \circledast r) \oplus (q \circledast r) \xrightarrow{a},$$

for all closed terms p, q, r.

The import of Theorem 1 is that, when proving the validity of (1), we can guarantee the firability condition for action a just by showing that $\mathrm{Fire}(\circledast, \oplus, a)$ holds. Theorem 1 underlies the soundness of the rule formats we present in what follows.

The reader will have already noticed that the rule form $(R1)$ does not place any restriction on tests for the variable y. This is possible because the second argument of the terms $(p \oplus q) \circledast r$, $p \circledast r$ and $q \circledast r$ is always the same, i.e. the term r. This means that, for each $\circledast$-defining rule, the same tests performed on the second argument on one side of (1) are performed on the other. Roughly speaking, one side of (1) may fire as much as the other does, insofar the second argument is concerned.

3.2 The Matching-Conclusion Condition

Theorem 1 tells us that any rule format, whose constraints imply condition $\mathrm{Fire}(\circledast, \oplus, a)$ for each action a, guarantees the validity of (1) provided that the matching-conclusion condition is met. Intuitively, in order to guarantee syntactically that the matching-conclusion condition is satisfied, the targets of the conclusions of $\circledast$-defining and $\oplus$-defining rules should 'match' when those operators are used in the specific contexts of the left- and the right-hand sides of (1). In what follows, we shall examine two different ways of ensuring the above-mentioned 'match' of the targets of the conclusions of $\circledast$-defining and $\oplus$-defining rules. The first relies on assuming that the targets of the conclusions of $\oplus$-defining rules are target variables of premises of rules of the form $(R2)$. The resulting rule format, which we present in this section, is based on easily checkable syntactic constraints and covers a large number of left-distributivity laws from the literature.

The First Rule Format. The rule format that we present deals with examples of left distributivity with respect to operators whose semantics is given by rules of the form $(R2)$ that, like those for the choice operator we mentioned in Example 1, have target variables of premises as targets of their conclusions. The following definition presents the syntactic constraints of the rule format.

Definition 8 (First rule format). *Let T be a TSS, and let $\circledast$ and $\oplus$ be binary operators in the signature of T. We say that the rules for $\circledast$ and $\oplus$ are in the* first rule format for left distributivity *if the following conditions are met:*

1. $\mathrm{Fire}(\circledast, \oplus, a)$ *holds for each action a,*
2. *$\circledast$ is non-left-inheriting,*
3. *each $\oplus$-defining rule has a target variable of one of its premises as target of its conclusion and*
4. *for each action a, either there is no a-emitting and $\oplus$-defining rule that tests both x and y, or if some a-emitting and $\circledast$-defining rule tests its left argument x then so do all a-emitting and $\circledast$-defining rules.*

Theorem 2 (Left distributivity over choice-like operators). *Let T be a TSS, and let $\circledast$ and $\oplus$ be binary operators in the signature of T. Assume that the rules for $\circledast$ and $\oplus$ are in the* first rule format for left distributivity. *Then*

$$(x \oplus y) \circledast z \leftrightarrows (x \circledast z) \oplus (y \circledast z) \ .$$

Remark 1. Condition 4 in Definition 8 is necessary for the soundness of the rule format for left distributivity proved in the above theorem. To see this, consider the operations $\oplus$ and $\circledast$ with rules

$$\frac{\{x \xrightarrow{a} x', y \xrightarrow{a} y'\}}{x \oplus y \xrightarrow{a} x'} \qquad \frac{\{x \xrightarrow{a} x', y \xrightarrow{a} y'\}}{x \circledast y \xrightarrow{a} x' \circledast y} \qquad \frac{\{y \xrightarrow{a} y'\}}{x \circledast y \xrightarrow{a} y'} \ .$$

The above rules satisfy all the conditions in Definition 8 apart from condition 4. Now, let a be a constant with rule $a \xrightarrow{a} \mathbf{0}$, where $\mathbf{0}$ is a constant with no rules. As our readers can easily check,

$$(a \circledast a) \oplus (\mathbf{0} \circledast a) \not\leftrightarrows (a \oplus \mathbf{0}) \circledast a.$$

Indeed, the term $(a \circledast a) \oplus (\mathbf{0} \circledast a)$ can perform a sequence of two a-labelled transitions, whereas $(a \oplus \mathbf{0}) \circledast a$ cannot because $a \oplus \mathbf{0}$ affords no transitions.

3.3 Examples of Application of the Rule Format

Theorem 2 provides us with a simple, yet rather powerful, syntactic condition in order to infer left-distributivity laws for operators like $+$. Many of the common left-distributivity laws are automatically derived from Theorem 2, as witnessed by the examples we now proceed to discuss.

Example 3 (Left merge and interleaving parallel composition). The operational semantics of the classic left-merge and interleaving parallel composition operators [11,13,17] is given by the rules below:

$$\frac{x \xrightarrow{a} x'}{x \mathbin{\lfloor\!\lfloor} y \xrightarrow{a} x' \parallel y} \qquad \frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} \ .$$

Theorem 2 yields the validity of the following law.

$$(x + y) \mathbin{\lfloor\!\lfloor} z \leftrightarrows (x \mathbin{\lfloor\!\lfloor} z) + (y \mathbin{\lfloor\!\lfloor} z)$$

Example 4 (Synchronous parallel composition). Consider the synchronous parallel composition from CSP [16][1] specified by the rules below, where a ranges over the set of actions:

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_s y \xrightarrow{a} x' \parallel_s y'} \ .$$

Theorem 2 yields the validity of the following law.

$$(x + y) \parallel_s z \leftrightarrows (x \parallel_s z) + (y \parallel_s z)$$

Example 5 (Join and '/' operators). Consider the join operator $\bowtie$ from [12] and the 'hourglass' operator $/$ from [4] specified by the rules below, where a, b range over the set of actions:

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \bowtie y \xrightarrow{a} x' \mp y'} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x/y \xrightarrow{a} x'/y'} \ ,$$

where $\mp$ denotes the delayed choice operator from [12]. (The operational specification of the delayed choice operator is immaterial for the analysis of this example.) Theorem 2 yields the validity of the following laws.

$$(x + y) \bowtie z \leftrightarrows (x \bowtie z) + (y \bowtie z) \qquad (x + y) \,/\, z \leftrightarrows (x \,/\, z) + (y \,/\, z)$$

Example 6 (Disrupt). Consider the following disrupt operator $\blacktriangleright$ [9,14] with rules

$$\frac{x \xrightarrow{a} x'}{x \blacktriangleright y \xrightarrow{a} x' \blacktriangleright y} \qquad \frac{y \xrightarrow{a} y'}{x \blacktriangleright y \xrightarrow{a} y'} \ .$$

Theorem 2 yields the validity of the following law.

$$(x + y) \blacktriangleright z \leftrightarrows (x \blacktriangleright z) + (y \blacktriangleright z)$$

[1] In [16], Hoare uses the symbol $\parallel$ to denote the synchronous parallel composition operator. Here we use that symbol for parallel composition.

Example 7 (Unless operator). The unless operator $\triangleleft$ from [10] and the operator Δ from [4, page 23] are specified by the rules

$$\frac{x \xrightarrow{a} x' \quad y \overset{b}{\nrightarrow} \text{ for } a < b}{x \triangleleft y \xrightarrow{a} x'} \qquad \frac{x \xrightarrow{a} x' \quad y \overset{b}{\nrightarrow} \text{ for } a < b}{x \mathbin{\Delta} y \xrightarrow{a} \theta(x')} \ ,$$

where $<$ is an irreflexive partial order over the set of actions and θ denotes the priority operator from [10]. (The operational specification of the priority operator is immaterial for the analysis of this example.) Theorem 2 yields the validity of the following laws.

$$(x + y) \triangleleft z \leftrightarrows (x \triangleleft z) + (y \triangleleft z) \qquad (x + y) \mathbin{\Delta} z \leftrightarrows (x \mathbin{\Delta} z) + (y \mathbin{\Delta} z)$$

4 Analyzing Targets of Conclusions of Deduction Rules

In this section, we extend the first rule format by generalizing the matching-conclusion conditions. We do so by examining different possible targets of the conclusions of the $\circledast$- and $\oplus$-defining rules. By analyzing different possible syntactic shapes for terms, we check which pairs of shapes can be related (possibly under some further requirements) while preserving the left-distributivity law.

Table 1 summarizes our results. Even though the offered list is not exhaustive, which, at first sight, seems a challenging task to achieve, we believe Table 1 offers enough cases to cover almost all practical cases, as demonstrated by the examples presented in the remainder of this section and in the full version of this paper.

In Table 1, x and y are considered as the variables for the first and second argument, respectively, for both $\circledast$- and $\oplus$-defining rules. When the variable x' is mentioned, implicitly the considered rule has a premise $x \xrightarrow{a} x'$ (for a-emitting rules). Similarly, when the variable y' is mentioned, implicitly the rule considered has a premise $y \xrightarrow{a} y'$. The term t stands for a generic open term from the

Table 1. Analysis of the targets of conclusions

	$\mathrm{toc}(d_1)$	$\mathrm{toc}(d_2)$	result	further requirements
1	$x' \circledast y$	x	$p \circledast r$	
2	$x' \circledast y$	y	$q \circledast r$	
3	x	$x' \oplus y'$	$p \oplus q$	$D(\circledast, a) = \{d_1\}$
4	x'	$x' \oplus y'$	$p' \oplus q'$	$D(\circledast, a) = \{d_1\}$
5	$x \circledast t$	$x' \oplus y'$	$(p \oplus q) \circledast \sigma(t)$	$D(\circledast, a) = \{d_1\}$, $x, x' \notin \mathrm{vars}(t)$
6	$x' \circledast t$	$x' \oplus y'$	$(p' \oplus q') \circledast \sigma(t)$	$D(\circledast, a) = \{d_1\}$, $x, x' \notin \mathrm{vars}(t)$
7	t	$x' \oplus y'$	$\sigma(t)$	$\oplus$ idempotent, $D(\circledast, a) = \{d_1\}$, $x, x' \notin \mathrm{vars}(t)$
8	t	x'	$\sigma'(t)$	Condition 4 of Definition 8, $x \notin \mathrm{vars}(t)$
9	t	y'	$\sigma'(t)$	Condition 4 of Definition 8, $x \notin \mathrm{vars}(t)$

with $\sigma = [y \mapsto r,\ y_i \mapsto r_i\ (i \in I)]$ and $\sigma' = [y \mapsto r,\ x' \mapsto p',\ y_i \mapsto r_i\ (i \in I)]$

signature, and p, q and r are hypothetical closed terms applied to the distributivity equation in this way: $(p \oplus q) \circledast r \leftrightarrows (p \circledast r) \oplus (q \circledast r)$. The symbols p', q', and r_i are considered as targets of possible transitions from p, q and r.

Table 1 is to be read as follows. First of all, $d_1 \in D(\circledast, a)$ and $d_2 \in D(\oplus, a)$, for some action a. In each row, the first column (column $\mathrm{toc}(d_1)$) specifies the form of the target of the conclusion of the $\circledast$-defining rule d_1 (e.g., x in case of row 3), and the second column (column $\mathrm{toc}(d_2)$) specifies the form of the target of the conclusion of the $\oplus$-defining rule d_2 (e.g., $x' \oplus y'$ in case of row 3). If the conditions in the column *further requirements* are satisfied (e.g., in row 3, d_1 is the only $\circledast$-defining and a-emitting rule), then the result of the transition of terms $(p \oplus q) \circledast r$ and $(p \circledast r) \oplus (q \circledast r)$ is specified by the term given in column *result* (e.g., $p \oplus q$ in row 3). In rows 5–6, the stated result is up to one application of the left-distributivity equation (1). The requirement $\oplus$ *idempotent* means that the operator $\oplus$ can be proved idempotent, e.g., by means of the rule format offered in [3].

The reader may want to notice that the first rule format of Section 3.2 is partly based on the analysis which leads to rows 8 and 9.

Theorem 3 (Soundness of Table 1). *Let T be a TSS. Let $\circledast$ and $\oplus$ be binary operations in the signature of T satisfying*

1. $\mathrm{Fire}(\circledast, \oplus, a)$, *and*
2. *if $D(\circledast, a) \neq \emptyset$ then for each $d_1 \in D(\circledast, a)$ and for each $d_2 \in D(\oplus, a)$, the rules d_1 and d_2 match a row in Table 1.*

It holds that:

$$(x \oplus y) \circledast z \leftrightarrows (x \circledast z) \oplus (y \circledast z).$$

In what follows, we apply the rule format provided in this section in order to check some examples of left-distributivity laws whose validity cannot be inferred using Theorem 2.

Example 8 (Unit-delay operator and the choice operator from ATP). Consider any TSS T containing the unit-delay operator $\lfloor\ \rfloor$ and the choice operator $+^*$ from ATP [21][2] and for which the transition relation $\xrightarrow{\chi}$ is deterministic. (The distinguished symbol χ denotes the passage of one unit of time.) The semantics of those operators is defined by the following rules, where $a \neq \chi$:

$$(ud_a)\ \frac{x \xrightarrow{a} x'}{\lfloor x \rfloor(y) \xrightarrow{a} x'} \qquad (ud_\chi)\ \frac{}{\lfloor x \rfloor(y) \xrightarrow{\chi} y} \qquad (extTime)\ \frac{x \xrightarrow{\chi} x' \quad y \xrightarrow{\chi} y'}{x +^* y \xrightarrow{\chi} x' +^* y'}$$

$$(extChl_a)\ \frac{x \xrightarrow{a} x'}{x +^* y \xrightarrow{a} x'} \qquad (extChr_a)\ \frac{y \xrightarrow{a} y'}{x +^* y \xrightarrow{a} y'}\ .$$

[2] In [21], the symbol of this operator is $\oplus$, whose use we prefer to avoid in this paper for the sake of clarity.

Table 1 can be used to match the targets of the conclusions as follows: the combination of ud_a and $extChl_a$ follows from row 8, the combination of ud_a and $extChr_a$ follows from row 9, and finally the combination of ud_χ and $extTime$ follows from row 7.

Example 9 (Timed left merge and the choice operator from ATP). Consider the TSS for ATP with the timed extension of the left-merge operator from Example 3 specified by the following rules, where $a \neq \chi$:

$$(merge_a)\frac{x \xrightarrow{a} x'}{x \between y \xrightarrow{a} x' \parallel y} \qquad (merge_\chi)\frac{x \xrightarrow{\chi} x' \quad y \xrightarrow{\chi} y'}{x \between y \xrightarrow{\chi} x' \between y'} .$$

Table 1 can be used to match the targets of the conclusions as follows: the combination $merge_a$, $extChl_a$ follows from row 8, the combination $merge_a$, $extChr_a$ follows from row 9 and the combination $merge_\chi$, $extTime$ follows from row 6.

In the extended version of this paper [1], we apply our rule formats to several more examples and also show how they can be applied to obtain distributivity for unary operators. The full version of the paper also offers a much more general format for left distributivity based on a notion of distributivity compliance between rules of which Table 1 is an approximation.

References

1. Aceto, L., Cimini, M., Ingolfsdottir, A., Mousavi, M.R., Reniers, M.A.: Rule formats for distributivity. Technical Report CSR-10-16, TU/Eindhoven (2010)
2. Aceto, L., Ingolfsdottir, A., Mousavi, M.R., Reniers, M.A.: Algebraic properties for free! Bulletin of the European Association for Theoretical Computer Science 99, 81–104 (2009); Columns: Concurrency
3. Aceto, L., Birgisson, A., Ingolfsdottir, A., Mousavi, M.R., Reniers, M.A.: Rule formats for determinism and idempotence. In: Arbab, F., Sirjani, M. (eds.) FSEN 2009. LNCS, vol. 5961, pp. 146–161. Springer, Heidelberg (2010)
4. Aceto, L., Bloom, B., Vaandrager, F.W.: Turning SOS rules into equations. Inf. Comput. 111(1), 1–52 (1994)
5. Aceto, L., Cimini, M., Ingolfsdottir, A., Mousavi, M.R., Reniers, M.A.: On rule formats for zero and unit elements. In: Proceedings of the 26th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVI), Ottawa, Canada. Electronic Notes in Theoretical Computer Science, vol. 265, pp. 145–160. Elsevier B.V., The Netherlands (2010)
6. Aceto, L., Fokkink, W., Ingolfsdottir, A., Luttik, B.: Finite equational bases in process algebra: Results and open questions. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 338–367. Springer, Heidelberg (2005)
7. Aceto, L., Fokkink, W., Verhoef, C.: Structural operational semantics. In: Handbook of Process Algebra, pp. 197–292. Elsevier, Amsterdam (2001)
8. Aceto, L., Ingolfsdottir, A., Mousavi, M.R., Reniers, M.A.: Rule formats for unit elements. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) SOFSEM 2010. LNCS, vol. 5901, pp. 141–152. Springer, Heidelberg (2010)

9. Baeten, J., Bergstra, J.: Mode transfer in process algebra. Technical Report Report CSR 00–01, Eindhoven University of Technology (2000)
10. Baeten, J., Bergstra, J., Klop, J.W.: Syntax and defining equations for an interrupt mechanism in process algebra. Fundamenta Informaticae IX(2), 127–168 (1986)
11. Baeten, J., Basten, T., Reniers, M.A.: Process Algebra: Equational Theories of Communicating Processes. Cambridge Tracts in Theoretical Computer Science, vol. 50. Cambridge University Press, Cambridge (2009)
12. Baeten, J., Mauw, S.: Delayed choice: An operator for joining Message Sequence Charts. In: Hogrefe, D., Leue, S. (eds.) Formal Description Techniques VII, Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques, Berne, Switzerland. IFIP Conference Proceedings, vol. 6, pp. 340–354. Chapman & Hall, Boca Raton (1995)
13. Bergstra, J., Klop, J.W.: Fixed point semantics in process algebras. Report IW 206, Mathematisch Centrum, Amsterdam (1982)
14. Brinksma, E.: A tutorial on LOTOS. In: Proceedings of the IFIP WG6.1 Fifth International Conference on Protocol Specification, Testing and Verification V, pp. 171–194. North-Holland Publishing Co., Amsterdam (1985)
15. Cranen, S., Mousavi, M.R., Reniers, M.A.: A rule format for associativity. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 447–461. Springer, Heidelberg (2008)
16. Hoare, C.: Communicating Sequential Processes. Prentice-Hall International, Englewood Cliffs (1985)
17. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc., Upper Saddle River (1989)
18. Moller, F.: The importance of the left merge operator in process algebras. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 752–764. Springer, Heidelberg (1990)
19. Mousavi, M.R., Reniers, M.A., Groote, J.F.: A syntactic commutativity format for SOS. Information Processing Letters 93, 217–223 (2005)
20. Mousavi, M.R., Reniers, M.A., Groote, J.F.: SOS formats and meta-theory: 20 years after. Theor. Comput. Sci. 373(3), 238–272 (2007)
21. Nicollin, X., Sifakis, J.: The algebra of timed processes, ATP: Theory and application. Information and Computation 114(1), 131–178 (1994)
22. Plotkin, G.D.: A structural approach to operational semantics. J. Log. Algebr. Program. 60-61, 17–139 (2004)
23. Przymusinski, T.: The well-founded semantics coincides with the three-valued stable semantics. Fundamenta Informaticae 13(4), 445–463 (1990)

Mutation Systems*

Dana Angluin, James Aspnes, and Raonne Barbosa Vargas

Department of Computer Science, Yale University, USA

Abstract. We propose *Mutation Systems* as a model of the evolution of a string subject to the effects of mutations and a fitness function. One fundamental question about such a system is whether knowing the rules for mutations and fitness, we can predict whether it is possible for one string to evolve into another. To explore this issue we define a specific kind of mutation system with point mutations and a fitness function based on conserved strongly k-testable string patterns. We show that for $k \geq 2$, such systems can simulate computation by both finite state machines and asynchronous cellular automata. The cellular automaton simulation shows that in this framework, universal computation is possible and the question of whether one string can evolve into another is undecidable. We also analyze the efficiency of the finite state machine simulation assuming random point mutations.

1 Introduction

Biological evolution proceeds by variation and selection. Efforts to determine the evolutionary relationships of different organisms often involve comparing the DNA sequences of their genomes to find similar subsequences that have been conserved during evolution, on the assumption that the conserved subsequences affect the fitness of the organisms. In this work we propose mutation systems as a simple model of variation and selection acting on strings of symbols, with the goal of exploring the properties of such systems, specifically what we can predict and learn about their behavior. Variation is modeled as a mutation function that maps a string to the set of possible mutations of that string. Selection is modeled as a fitness function that determines whether each string is fit or not. The main relation we consider in this paper is whether one fit string can evolve to another fit string through a sequence of fit strings, each of which is a possible mutation of its predecessor.

2 Preliminaries

An alphabet Σ is a finite nonempty set of symbols. Σ^* denotes the set of all finite strings of symbols from Σ. The empty string is denoted λ. A language is any subset of Σ^*. Σ^k denotes those elements of Σ^* of length k. The symbols in a string s of length n are indexed from 1 to n and $s[i]$ denotes the i^{th} symbol of s.

* Research supported by the National Science Foundation under Grant CCF-0916389.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 92–104, 2011.

We consider non-deterministic finite state machines with no accepting states, defined as follows. A finite state machine (FSM) is a quadruple $M = (\Sigma, Q, q_0, \delta)$, where Σ is the alphabet of input symbols, Q is the set of states, q_0 is the initial state, and δ is the transition function, which maps $Q \times \Sigma$ to subsets of Q. If every $\delta(q, a)$ contains exactly one state, then M is deterministic. In this case we may write $\delta(q, a) = q'$ instead of $\delta(q, a) = \{q'\}$.

3 Mutation Systems

We propose a model of the evolution of a string subject to the effects of mutations and a fitness function. A single step consists of a mutation of the current string followed by an application of the fitness function. If the fitness function determines that the mutated string is fit, the mutated string replaces the current string; otherwise the mutated string is discarded and the current string is kept.

Definition 1. *A **mutation system** $S = (\Sigma, \mu, f)$ is composed of an alphabet Σ, a mutator μ that maps Σ^* to subsets of Σ^* and a fitness function $f : \Sigma^* \to \{0, 1\}$. The mutator μ specifies the set of strings to which a given string can mutate in one step. The fitness function f determines whether a given string s is fit ($f(s) = 1$) or not ($f(s) = 0$).*

Given a mutation system S and two fit strings s_1 and s_2, we are interested in the question of whether s_1 can evolve to s_2 through a sequence of steps permitted by S.

Definition 2. *Let a mutation system $S = (\Sigma, \mu, f)$ and two strings $s_1, s_2 \in \Sigma^*$ be given. We say that s_1 **can mutate to** s_2 **in one step**, denoted $s_1 \to_\mu s_2$, if $s_2 \in \mu(s_1)$. We say that s_1 **can evolve to** s_2 **in one step**, denoted $s_1 \to_S s_2$, if $f(s_1) = f(s_2) = 1$ and s_1 can mutate to s_2 in one step.*

As is usual, we denote the reflexive transitive closure of these relations by a superscripted * on the arrow. We say that s_1 **can mutate to** s_2 if $s_1 \to_\mu^* s_2$, that is, there is a finite sequence of zero or more mutation steps that carries s_1 to s_2. Similarly, we say that s_1 **can evolve to** s_2 if $s_1 \to_S^* s_2$, that is, there is a finite sequence of zero or more evolution steps that carries s_1 to s_2. Note that in the latter case, s_1, s_2 and any intermediate strings in some evolution must be fit.

3.1 Point Mutations

A point mutation of a string is obtained by deleting or inserting a single occurrence of a symbol or by replacing a single occurrence of a symbol by any symbol.

Definition 3. *Let s be any string. The mutators μ_d, μ_i, μ_r, and μ_p are defined as follows.*

1. *$\mu_d(s)$ is the set of strings that can be obtained by deleting exactly one occurrence of a symbol from s.*

2. $\mu_i(s)$ *is the set of strings that can be obtained from* s *by inserting exactly one occurrence of a symbol from* Σ *into* s.
3. $\mu_r(s)$ *is the set of strings that can be obtained from* s *by replacing exactly one occurrence of a symbol in* s *by any symbol from* Σ.
4. $\mu_p(s) = \mu_d(s) \cup \mu_i(s) \cup \mu_r(s)$.

The mutator μ_p permits any single point mutation of a string. Reversibility is a relevant property of mutators and mutation systems.

Definition 4. *A mutator* μ *is* ***stepwise reversible*** *if for all strings* s_1 *and* s_2,

$$s_2 \in \mu(s_1) \Leftrightarrow s_1 \in \mu(s_2).$$

That is, if s_1 *can mutate to* s_2 *in one step, then* s_2 *can mutate back to* s_1 *in one step. A mutation system* $S = (\Sigma, \mu, f)$ *is* ***reversible*** *if for all strings* s_1 *and* s_2,

$$(s_1 \rightarrow_S^* s_2) \Leftrightarrow (s_2 \rightarrow_S^* s_1).$$

That is, if s_1 *can evolve to* s_2, *then* s_2 *can evolve to* s_1.

The point mutator μ_p is stepwise reversible: an insertion can be reversed by a deletion, a deletion by an insertion, and a replacement by the opposite replacement. The following lemma is immediate.

Lemma 1. *If* μ *is stepwise reversible then* $S = (\Sigma, \mu, f)$ *is reversible.*

3.2 Conservation of Strictly k-Testable Patterns

We consider fitness functions defined by very local properties of a string, namely properties characterized by strictly k-testable languages [3,7,10]. Head [5] and Yokomori and Kobayashi [13] describe applications of k-testable languages to modeling biological phenomena.

Definition 5. *Let* Σ *be an alphabet. A strictly* k*-testable pattern* $P = (PRE, MID, SUF)$ *is composed of three sets of strings with* $PRE \subseteq \Sigma^{k-1}$, $MID \subseteq \Sigma^k$, *and* $SUF \subseteq \Sigma^{k-1}$. *The language of* P, *denoted* L_P, *is the set of all strings* s *of length at least* k *such that the prefix of* s *of length* $k-1$ *is in* PRE, *every substring of* s *of length* k *is in* MID, *and the suffix of* s *of length* $k-1$ *is in* SUF.

A fitness function f is defined to be strictly k-testable if there exists a strictly k-testable pattern P such that for every string s, $f(s) = 1$ iff $s \in L_P$. A **k-simple mutation system** is is a mutation system with mutation operator μ_p and a strictly k-testable fitness function. In what follows we focus on 2-simple mutation systems.

The technique of symbol duplication is useful in preventing unwanted point mutations in a 2-simple mutation system. If the alphabet is Σ, then the **duplicated alphabet** $D(\Sigma)$ consists of two copies of each symbol $a \in \Sigma$, one with index 1, denoted a^1, and one with index 2, denoted a^2. We define the

duplication map d from Σ^* to $D(\Sigma)^*$ such that $d(s)$ is obtained from s by replacing every occurrence of a symbol a in s by the string a^1a^2. We define a **projection map** h_1 from $D(\Sigma)^*$ to Σ^* such that $h_1(s)$ replaces every index 1 symbol a^1 by a and every index 2 symbol a^2 by the empty string. For example, $d(abb) = a^1a^2b^1b^2b^1b^2$ and $h_1(a^1b^1b^2a^2a^1) = aba$. Clearly $h_1(d(s)) = s$.

Example: Symbol Duplication. Let $\Sigma = \{a, b\}$. We define a 2-simple mutation system $S_2 = (\Sigma_2, \mu_p, f_2)$ that protects strings against point mutations. The alphabet Σ_2 is $D(\Sigma) = \{a^1, a^2, b^1, b^2\}$ and the strictly 2-testable fitness function f_2 is defined by the prefix strings $\{a^1, b^1\}$, the suffix strings $\{a^2, b^2\}$, and the middle strings

$$\{a^1a^2, a^2a^1, a^2b^1, b^1b^2, b^2a^1, b^2b^2\}.$$

The set of strings that are fit with respect to f_2 are exactly those of the form $d(s)$ for some nonempty $s \in \Sigma^*$, for example, $a_1a_2b_1b_2b_1b_2$. If a fit string undergoes any non-identity point mutation, the resulting string is not fit with respect to f_2.

4 Simulating FSM Computation

To represent FSM computation using a reversible mutation system, we choose a reversible representation: FSM computation histories, analogous to Bennett's construction to make Turing machines reversible [1]. Let $M = (\Sigma, Q, q_0, \delta)$ be a finite state machine. Choose an element $x \notin Q$ and define the **state-annotated alphabet** Σ_Q as the set of all symbols a_q such that $a \in \Sigma$ and $q \in Q \cup \{x\}$. The symbol a_q represents the state q of M after reading the symbol a, with x indicating that the symbol is unread. The main symbol component of a_q is a and the state component is q.

Given a string $s \in \Sigma^*$ of length n, a **computation history of** M **on** s is a string $s' \in (\Sigma_Q)^*$ of length n such that the string of main symbol components of s' is s, and the sequence of state components consists of $q_1, q_2, \ldots, q_i \in Q$ followed by $(n - i)$ x's for some $0 \leq i \leq n$, where for each $1 \leq j < i$, $q_{j+1} \in \delta(q_j, s[j])$. In this case, s' represents the computation in which M has read the first i symbols of s and for each j gives the state reached after reading the j^{th} input symbol. The **initial computation history of** M **on** s, denoted $I_x(s)$, is obtained from s by replacing each a by a_x, signifying that all the input symbols of s are unread.

Example: M_1. Define a deterministic finite state machine $M_1 = (\{a, b\}, \{0, 1\}, 0, \delta_1)$ with transition function δ_1 given by $\delta_1(0, a) = 1$, $\delta_1(0, b) = 0$, $\delta_1(1, a) = 0$, and $\delta_1(1, b) = 1$. The state of M_1 indicates whether it has read an odd (1) or even (0) number of a's. The computation histories of M_1 on the input string $abaa$ are the following: $a_xb_xa_xa_x$, $a_1b_xa_xa_x$, $a_1b_1a_xa_x$, $a_1b_1a_0a_x$, $a_1b_1a_0a_1$.

4.1 From FSMs to Mutation Systems

Given a FSM $M = (\Sigma, Q, q_0, \delta)$, we describe how to construct a 2-simple mutation system $S = (\Sigma', \mu_p, f)$ such that for any non-empty input string s for M,

$$
\begin{array}{cccccccc}
a_x^1 & a_x^2 & b_x^1 & b_x^2 & a_x^1 & a_x^2 & a_x^1 & a_x^2 \\
a_1^1 & a_x^2 & b_x^1 & b_x^2 & a_x^1 & a_x^2 & a_x^1 & a_x^2 \\
a_1^1 & a_1^2 & b_x^1 & b_x^2 & a_x^1 & a_x^2 & a_x^1 & a_x^2 \\
a_1^1 & a_1^2 & b_1^1 & b_x^2 & a_x^1 & a_x^2 & a_x^1 & a_x^2 \\
a_1^1 & a_1^2 & b_1^1 & b_1^2 & a_x^1 & a_x^2 & a_x^1 & a_x^2 \\
a_1^1 & a_1^2 & b_1^1 & b_1^2 & a_0^1 & a_x^2 & a_x^1 & a_x^2 \\
a_1^1 & a_1^2 & b_1^1 & b_1^2 & a_0^1 & a_0^2 & a_x^1 & a_x^2 \\
a_1^1 & a_1^2 & b_1^1 & b_1^2 & a_0^1 & a_0^2 & a_1^1 & a_x^2 \\
a_1^1 & a_1^2 & b_1^1 & b_1^2 & a_0^1 & a_0^2 & a_1^1 & a_1^2
\end{array}
$$

Fig. 1. Sequence of mutations for computation of M_1 on input *abaa*

the computation histories of M on input s are represented by the strings that $d(I_x(s))$ may evolve to in S. The alphabet Σ' is $D(\Sigma_Q)$. In the symbol a_q^i, the main symbol component is a, the state component is q and the index is i. For the example FSM M_1,

$$\Sigma' = \{a_x^1, a_x^2, a_0^1, a_0^2, a_1^1, a_1^2, b_x^1, b_x^2, b_0^1, b_0^2, b_1^1, b_1^2\}.$$

Corresponding to the initial computation history $I_x(s)$ of M on input s is the initial string $d(I_x(s))$ with every symbol replaced by its duplicates indexed 1 and 2. For the FSM M_1, we have

$$d(I_x(abaa)) = a_x^1 a_x^2 b_x^1 b_x^2 a_x^1 a_x^2 a_x^1 a_x^2.$$

The strictly 2-testable pattern P = (PRE, MID, SUF) that determines the fitness function f is defined as follows. The set PRE contains all symbols of the form a_x^1 and a_q^1 such that $a \in \Sigma$ and $q \in \delta(q_0, a)$. The set SUF contains all symbols of the form a_x^2 and a_q^2 such that $a \in \Sigma$ and $q \in Q$. The set MID contains several types of strings of length 2, as follows.

1. Initial duplicate: $a_x^1 a_x^2$ for all $a \in \Sigma$.
2. Initial boundary: $a_x^2 b_x^1$ for all $a, b \in \Sigma$.
3. Duplicate update needed: $a_q^1 a_x^2$ for all $a \in \Sigma$ and $q \in Q$.
4. Updated duplicate: $a_q^1 a_q^2$ for all $a \in \Sigma$ and $q \in Q$.
5. Updated boundary: $a_q^2 b_x^1$ for all $a, b \in \Sigma$ and $q \in Q$.
6. State transition: $a_q^2 b_{q'}^1$ for all $a, b \in \Sigma$, $q \in Q$, and $q' \in \delta(q, b)$.

For example, the sequence of steps of M_1 on input *abaa* can be achieved by the mutation steps shown in Figure 1. Each FSM step requires two mutations.

4.2 Correctness of the FSM Simulation

To see that S correctly represents computation by M, we establish certain properties of evolvability in S. Note that for every $a, b \in \Sigma$, $a_x^1 \in$ PRE, $a_x^2 \in$ SUF, $a_x^1 a_x^2 \in$ MID and $a_x^2 b_x^1 \in$ MID, and therefore for every nonempty $s \in \Sigma^*$ we have $f(d(I_x(s))) = 1$, that is, $d(I_x(s))$ is fit in S. The following lemmas prove Theorem 1; their proofs are omitted for lack of space.

Lemma 2. *Let $s' \in (\Sigma')^*$ be any nonempty string such that $f(s') = 1$. Then s' has the following properties.*

1. *The indices of s' alternate between 1 and 2, beginning with 1 and ending with 2.*
2. *If two consecutive symbols of s' are indexed 1 and 2, they must be $a_x^1 a_x^2$ or $a_q^1 a_x^2$ or $a_q^1 a_q^2$ for some $a \in \Sigma$ and $q \in Q$.*
3. *If two consecutive symbols of s' are indexed 2 and 1, they must be $a_x^2 b_x^1$ or $a_q^2 b_x^1$ or $a_q^2 b_{q'}^1$ for some $a, b \in \Sigma$ and $q, q' \in Q$ such that $q' \in \delta(q, b)$.*
4. *The state components of s' consist of a sequence of elements of Q followed by a sequence of x's.*
5. *The string $h_1(s')$ is a computation history of M on the input s composed of the sequence of main symbol components of $h_1(s')$.*

Lemma 3. *Let s be a nonempty input string for M. Let s' be any string evolvable from $d(I_x(s))$ in S. Then $h_1(s')$ is a computation history of M on input s.*

Lemma 4. *Let s be a nonempty input string for M. If t is any computation history of M on input s, then $d(t)$ is evolvable in S from $d(I_x(s))$.*

Theorem 1. *Let a finite state machine $M = (\Sigma, Q, q_0, \delta)$ be given, and let $S = (\Sigma', \mu_p, f)$ be the 2-simple mutation system constructed from M according to the method described above. Let $s \in \Sigma^*$ be a nonempty input string for M. For every string s' evolvable in S from $d(I_x(s))$, $h_1(s')$ is a computation history of M on input s. For every computation history t of M on input s, $d(t)$ is evolvable from $d(I_x(s))$.*

In case M is nondeterministic, the strings evolvable from $d(I_x(s))$ in S give all possible computation histories of M on input s because S is a reversible mutation system and may evolve backward to the initial string from any string it reaches. In case M is deterministic, the strings evolvable from $d(I_x(s))$ form a line graph of $2n$ vertices, with $d(I_x(s))$ at one end and the history in which all symbols have state components in Q at the other end.

We consider **random point mutations**, in which each type of mutation (deletion, insertion, replacement) is selected with some probability, and for each type, a string position to apply it is selected equiprobably, and a symbol is selected equiprobably from the alphabet for an insertion or a replacement. For a deterministic machine M, the result is a Markov chain of $2n$ vertices that moves forward when a mutation causes another symbol to have state component $q \in Q$ and backward when a mutation causes another symbol to have state component x.

In the construction described above, the probability of a forward mutation and a backward mutation is the same, namely $p_r/(2n|\Sigma'|)$ where p_r is the probability of choosing replacement. By standard results on random walks, this implies that the expected number of attempted mutations for the simulation to reach the final string is $O(|\Sigma'|n^3/p_r)$. However, by biasing the random walk in the forward

direction, this can be reduced to $O(|\Sigma'|n^2/p_r)$, as suggested by Bennett [2]. For example, if we make an additional copy of every symbol a_q^i such that $q \in Q$, and treat them as equivalent in the simulation, then the probability of a forward mutation is twice that of a backward mutation.

5 Simulating Cellular Automata

Cellular automata are a well known model of computation introduced by Von Neumann [12], motivated by physical and biological problems. In a recent survey paper, Kari [6] notes that cellular automata have several fundamental properties of the physical world: they are massively parallel, homogeneous, and reversible, have only local interactions, and facilitate formulation of conservation laws based on local update rules. These properties match well with the features of our mutation system model, and a detailed comparison sheds light on the power and expressiveness of our new model. We consider one-dimensional asynchronous reversible cellular automata with insertions and deletions because they support universal computation [9].

A **cellular automaton** $C = (\Sigma, \delta)$ is composed of an alphabet of symbols Σ and a set δ transition rules of the form $axb \leftrightarrow ayb$ for substitutions or $ab \leftrightarrow axb$ for insertions and deletions, where $a, b, x, y \in \Sigma$. The idea is that the value of a given cell of the automaton may change only when both its neighbors have specific values.

For $s_1, s_2 \in \Sigma^*$, s_1 **can reach** s_2 **in one step of** C, denoted $s_1 \rightarrow_C s_2$, if applying one transition rule to s_1 yields s_2. And s_1 **can reach** s_2 **in** C if $s_1 \rightarrow_C^* s_2$. Given an input string $s \in \Sigma^*$, a **snapshot of** C **on input** s is any string s' such that s can reach s' in C. For example if we have the rules $\{abc \leftrightarrow adc, dce \leftrightarrow dfe, fe \leftrightarrow fge\}$, and an input $abce$, the snapshots of the computation on this input are $\{abce, adce, adfe, adfge\}$.

5.1 From Cellular Automata to Mutation Systems

Given a cellular automaton $C = (\Sigma, \delta)$, we describe how to construct a 2-simple mutation system $S = (\Sigma', \mu_p, f)$ such that for every nonempty input string $s \in \Sigma^*$, the snapshots of C on input s are represented by the strings evolvable from $d(s)$ in S.

The simulation of a cellular automaton is more complex than the simulation of a FSM; one step of the cellular automaton may require fourteen point mutations. To ensure the correct coordination of these mutations, we duplicate the symbols and also allow them to store information about one or two symbols to the left or right. The idea is that before performing a transition of the cellular automaton, the system "locks" the left and right neighbors of the symbol to be changed. The additional symbol $(-)$ marks the left and right edges of the transition. To permit insertions and deletions in the string, there is an extra index $(*)$ besides 1 and 2. As an example, the following string

$$a^1 \cdot a^2 \cdot {}_-b^1 \cdot {}_b b^2 \cdot {}_{bb}c^1 \cdot c^2_{dd} \cdot d^1_d \cdot d^2_- \cdot e^1 \cdot e^2$$

represents the string *abcde* where c has locked its left and right neighbors preparing for a transition. The explicit concatenation operator $(\cdot)$ separates individual symbols above. After a transition has been performed, symbols may unlock their neighbors and return to having empty neighbor information.

Let $J = \{1, 2, *\}$ be the set of indices and $N = \{\lambda\} \cup \{-\} \cup \Sigma \cup \Sigma^2$ be the set of possible neighbor strings. Define the alphabet Σ' for the mutation system as follows.

$$\Sigma' = \{{}_u a^i_v : a \in \Sigma, i \in J, u \in N, v \in N\}.$$

In the symbol ${}_u a^i_v$, a is the main symbol component, i is the index, u (resp. v) is the left (resp. right) neighbor information. Let Σ_1 denote the set of symbols of the form a^i with empty neighbor information and index $i \in \{1, 2\}$.

The symbol duplication map d maps Σ^* to $(\Sigma_1)^*$ by replacing each occurrence of a symbol a by the string $a^1 \cdot a^2$. We define a projection h_1 from $(\Sigma')^*$ to Σ^* that maps each symbol with index 1 to its main symbol component, and maps all others to the empty string. Thus $h_1(d(s)) = s$ for all $s \in \Sigma^*$. Also, for example, $h_1({}_- a^1 \cdot {}_a a^2 \cdot {}_{aa} d^1 \cdot b^2_{cc} \cdot c^1_c \cdot c^2_-) = adc$.

5.2 Defining the Fitness Function

We describe the strictly 2-testable pattern $P = (\text{PRE}, \text{MID}, \text{SUF})$ that determines the fitness function f of the mutation system. PRE consists of all symbols a^1 and ${}_- a^1$ such that $a \in \Sigma$. SUF consists of all symbols a^2 and a^2_- such that $a \in \Sigma$. The set MID contains strings of length two to deal with the situations: (1) empty neighbor information, (2) substitution rules, and (3) insertion/deletion rules.

Empty neighbor information. To permit duplicated symbols we have $a^1 \cdot a^2$ for all $a \in \Sigma$. To permit a boundary between symbols we have $a^2 \cdot b^1$ for all $a, b \in \Sigma$. Together with PRE and SUF, these cases ensure that $f(d(s)) = 1$ for every nonempty string $s \in \Sigma^*$.

Substitution rules. For each substitution rule $axb \leftrightarrow ayb$ we add strings to MID that permit $d(axb)$ and $d(ayb)$ to mutate to each other as follows.

To add left neighbor information $-$ to a^1 we have $c^2 \cdot {}_- a^1$ and $c^2_- \cdot {}_- a^1$ for all $c \in \Sigma$, as well as ${}_- a^1 \cdot a^2$. To add right neighbor information $-$ to b^2 we have $b^2_- \cdot d^1$ and $b^2_- \cdot {}_- d^1$ for all $d \in \Sigma$, as well as $b^1 \cdot b^2_-$.

To add left neighbor information a to the symbol a^2 we have ${}_- a^1 \cdot {}_a a^2$, as well as ${}_a a^2 \cdot x^1$ and ${}_a a^2 \cdot y^1$. To add right neighbor information b to the symbol b^1 we have $b^1_b \cdot b^2_-$, as well as $x^2 \cdot b^1_b$ and $y^2 \cdot b^1_b$.

To add left neighbor information aa to the symbol x^1 or y^1 we have ${}_a a^2 \cdot {}_{aa} x^1$ and ${}_{aa} x^1 \cdot x^2$, as well as ${}_a a^2 \cdot {}_{aa} y^1$ and ${}_{aa} y^1 \cdot y^2$. To add right neighbor information bb to the symbol x^2 or y^2 we have $x^2_{bb} \cdot b^1_b$ and $x^1 \cdot x^2_{bb}$, as well as $y^2_{bb} \cdot b^1_b$ and $y^1 \cdot y^2_{bb}$. The strings that permit both left neighbor information of aa on x^1 and right neighbor information bb on x^2 (and similarly for y^1 and y^2) are ${}_{aa} x^1 \cdot x^2_{bb}$ and ${}_{aa} y^1 \cdot y^2_{bb}$.

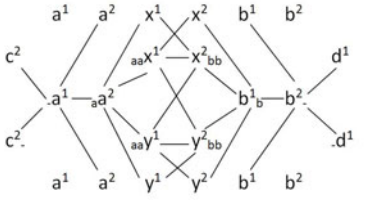


Fig. 2. MID strings allowing substitutions for the rule $axb \leftrightarrow ayb$

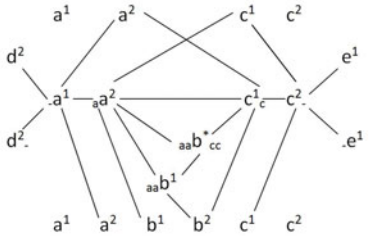


Fig. 3. MID strings allowing insertions and deletions for the rule $ac \leftrightarrow abc$

The above strings permit consecutive symbols indexed 1 and 2 only if they have the same main symbol. However, we need to permit x to be replaced by y and vice versa. The strings that permit this are ${}_{aa}x^1 \cdot y^2_{bb}$ and ${}_{aa}y^1 \cdot x^2_{bb}$. Figure 2 shows the strings added to MID for the substitution rule $axb \leftrightarrow ayb$. Each line connects two symbols forming a string in MID.

Insertion/deletion rules. For each insertion/deletion transition rule $ac \leftrightarrow abc$ we add the following strings to MID. To add left neighbor information $-$ to a^1 we have $d^2 \cdot \_a^1$ and $d^2_- \cdot \_a^1$ for all $d \in \Sigma$, as well as $\_a^1 \cdot a^2$. To add right neighbor information $-$ to c^2 we have $c^2_- \cdot e^1$ and $c^2_- \cdot \_e^1$ for all $e \in \Sigma$, as well as $c^1 \cdot c^2_-$.

To add left neighbor information a to a^2 we have $\_a^1 \cdot {}_a a^2$ as well as ${}_a a^2 \cdot b^1$ and ${}_a a^2 \cdot c^1$. To add right neighbor information c to c^1 we have $c^1_c \cdot c^2_-$ as well as $b^2 \cdot c^1_c$ and $a^2 \cdot c^1_1$. The string that permits both left neighbor information of a on a^2 and right neighbor information of c on c^1 when a^2 and c^1 are adjacent is ${}_a a^2 \cdot c^1_c$.

To add left neighbor information aa to b^1 when a^2 is adjacent to b^1, we have ${}_a a^2 \cdot {}_{aa} b^1$ and ${}_{aa} b^1 \cdot b^2$.

To allow b to be deleted or inserted, we add strings using the $*$ index that permit b^2 to become ${}_{aa} b^*_{cc}$ and vice versa, namely ${}_{aa} b^1 \cdot {}_{aa} b^*_{cc}$ and ${}_{aa} b^*_{cc} \cdot c^1_c$. Finally we add a string that permits the insertion/deletion of b^1 and ${}_{aa} b^*_{cc}$, namely ${}_a a^2 \cdot {}_{aa} b^*_{cc}$. Figure 3 shows the strings in MID for the insertion/deletion rule $ac \leftrightarrow abc$. Again each line connecting two symbols indicates a string in MID.

This completes the construction of MID and the mutation system S. To see that f permits the transitions of C to be simulated, we prove the following.

Lemma 5. *If $s \in \Sigma^*$ is nonempty and $s \rightarrow_C t$ then $d(s) \rightarrow^*_S d(t)$.*

Proof. If t is obtained from s by using a substitution rule to substitute ayb for axb in s, then the sequence of point mutations in Figure 4 applied to the relevant portion of $d(s)$ shows that $d(t)$ is evolvable from $d(s)$. Symbols (if any) to the left and right of this portion of $d(s)$ are unchanged.

If t is obtained from s by using an insertion/deletion rule to replace abc in s by ac, then the sequence of point mutations in Figure 5 applied to the relevant portion of $d(s)$ shows that $d(t)$ is evolvable from $d(s)$. Symbols (if any) to the

$$
\begin{array}{l}
a^1 \cdot a^2 \cdot x^1 \cdot x^2 \cdot b^1 \cdot b^2 \\
{}_{\_}a^1 \cdot a^2 \cdot x^1 \cdot x^2 \cdot b^1 \cdot b^2 \\
{}_{\_}a^1 \cdot a^2 \cdot x^1 \cdot x^2 \cdot b^1 \cdot b^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot x^1 \cdot x^2 \cdot b^1 \cdot b^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot x^1 \cdot x^2 \cdot b^1_b \cdot b^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot {}_{aa}x^1 \cdot x^2 \cdot b^1_b \cdot b^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot {}_{aa}x^1 \cdot x^2_{bb} \cdot b^1_b \cdot b^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot {}_{aa}y^1 \cdot x^2_{bb} \cdot b^1_b \cdot b^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot {}_{aa}y^1 \cdot y^2_{bb} \cdot b^1_b \cdot b^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot {}_{aa}y^1 \cdot y^2 \cdot b^1_b \cdot b^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot y^1 \cdot y^2 \cdot b^1_b \cdot b^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot y^1 \cdot y^2 \cdot b^1 \cdot b^2_{\_} \\
{}_{\_}a^1 \cdot a^2 \cdot y^1 \cdot y^2 \cdot b^1 \cdot b^2_{\_} \\
{}_{\_}a^1 \cdot a^2 \cdot y^1 \cdot y^2 \cdot b^1 \cdot b^2 \\
a^1 \cdot a^2 \cdot y^1 \cdot y^2 \cdot b^1 \cdot b^2
\end{array}
$$

Fig. 4. Sequence of mutations to achieve $d(axb) \leftrightarrow_S^* d(ayb)$

$$
\begin{array}{l}
a^1 \cdot a^2 \cdot b^1 \cdot b^2 \cdot c^1 \cdot c^2 \\
{}_{\_}a^1 \cdot a^2 \cdot b^1 \cdot b^2 \cdot c^1 \cdot c^2 \\
{}_{\_}a^1 \cdot a^2 \cdot b^1 \cdot b^2 \cdot c^1 \cdot c^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot b^1 \cdot b^2 \cdot c^1 \cdot c^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot b^1 \cdot b^2 \cdot c^1_c \cdot c^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot {}_{aa}b^1 \cdot b^2 \cdot c^1_c \cdot c^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot {}_{aa}b^1 \cdot {}_{aa}b^*_{cc} \cdot c^1_c \cdot c^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot {}_{aa}b^*_{cc} \cdot c^1_c \cdot c^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot c^1_c \cdot c^2_{\_} \\
{}_{\_}a^1 \cdot {}_a a^2 \cdot c^1 \cdot c^2_{\_} \\
{}_{\_}a^1 \cdot a^2 \cdot c^1 \cdot c^2_{\_} \\
{}_{\_}a^1 \cdot a^2 \cdot c^1 \cdot c^2 \\
a^1 \cdot a^2 \cdot c^1 \cdot c^2
\end{array}
$$

Fig. 5. Sequence of mutations to achieve $d(abc) \leftrightarrow_S^* d(ac)$

left and right of this portion of $d(s)$ are unchanged. Because point mutations are reversible, the reverse of this sequence indicates how ac can be replaced by abc. □

5.3 Correctness of the Cellular Automaton Simulation

Theorem 2. *Let $C = (\Sigma, \delta)$ be a cellular automaton and let $S = (\Sigma', \mu_p, f)$ be the 2-simple mutation system constructed from C as described above. Let $s \in \Sigma^*$ be a nonempty string. For any string t reachable from s in C, the string $d(t)$ is evolvable from $d(s)$. Conversely, for any string s' evolvable in S from $d(s)$, $h_1(s')$ is reachable from s in C.*

Proof. The first part follows by induction on the number of transitions to reach t from s in C, using Lemma 5.

For the converse, it suffices to show that if $d(s) \rightarrow_S^* s'$ and $s \rightarrow_C^* h_1(s')$ and $s' \rightarrow_S t$ then $h_1(s') \rightarrow_C^* h_1(t)$.

Suppose a^1 is the first symbol and b^2 is the last symbol of $d(s)$. To maintain fitness, these symbols cannot be deleted, and no symbol can be inserted before the first or after the last. The only changes they can undergo that result in fit strings is that a^1 can be replaced by ${}_{\_}a^1$ and vice versa, and b^2 can be replaced by $b^2_{\_}$ and vice versa. Thus we need only consider changes to interior symbols.

Let $s' \in (\Sigma')^*$ be any nonempty fit string. The indices of any three consecutive symbols in s' must be one of the seven possibilities: $(1, 2, 1)$, $(2, 1, 2)$, $(1, 2, *)$, $(2, 1, *)$, $(1, *, 1)$, $(2, *, 1)$, and $(*, 1, 2)$.

If the deletion of a symbol from the interior of s' yields another fit string t, then the symbol deleted must be the middle symbol in one of the index sequences: $(1, 2, *)$, $(2, 1, *)$ or $(2, *, 1)$. In the first and third cases the symbols of index 1

are unchanged and $h_1(t) = h_1(s')$. In the case of $(2, 1, *)$, the three symbols in s' must be

$$ {}_{a}a^{2} \cdot {}_{aa}b^{1} \cdot {}_{aa}b^{*}_{cc}, $$

which implies that $abc \leftrightarrow ac$ is a rule in C. Moreover, the symbol before this triple must be ${}_{-}a^{1}$ and the symbol after it must be c^{1}_{c}, which means that $h(t)$ is obtained from $h_1(s')$ by replacing abc by ac, and $h_1(s') \rightarrow_C h_1(t)$.

Analogously, if an insertion of a symbol in the interior of s' yields another fit string t, then only an insertion into $(2, *)$ (yielding $(2, 1, *)$) results in $h_1(t) \neq h_1(s')$. This implies that the inserted symbol and its two neighbors to the left and right in t are as follows:

$$ {}_{-}a^{1} \cdot {}_{a}a^{2} \cdot {}_{aa}b^{1} \cdot {}_{aa}b^{*}_{cc} \cdot c^{1}_{c}. $$

Thus, $abc \leftrightarrow ac$ is a rule of C and $h_1(t)$ is obtained from $h_1(s')$ by replacing ac by abc and $h_1(s') \rightarrow_C h_1(t)$.

If a replacement of one interior symbol of s' by another yields a fit string t, then either the replacement changes the index of the symbol or not. The only possible kinds of replacements that change the index of the symbol are of the form $(1, 2, 1) \leftrightarrow (1, *, 1)$. This leaves the symbols of index 1 unchanged, and $h_1(t) = h_1(s')$.

Thus only replacements that we must consider are replacements of symbols of index 1 by symbols of index 1 with a different main symbol, so that $h_1(t) \neq h_1(s')$. The indices of the replaced symbol and its two neighbors must be either $(2, 1, *)$ or $(2, 1, 2)$. In the first case, the three symbols of s' are of the form

$$ {}_{a}a^{2} \cdot {}_{aa}b^{1} \cdot {}_{aa}b^{*}_{cc}, $$

and there is no other symbol that can replace ${}_{aa}b^{1}$ and yield a fit string t. In the case of $(2, 1, 2)$ the possibilities for the symbol of index 1 are a^1, ${}_{-}a^{1}$, a^{1}_{a}, and ${}_{bb}a^{1}$. When the symbol to its right is one of a^2, a^{2}_{-}, or ${}_{a}a^{2}$, replacing the symbol of index 1 in s' by a symbol of index 1 and main symbol other than a does not yield a fit string. Thus, the only possibilities in s' for the symbol of index 1 and its right neighbor are the following: (1) $a^{1} \cdot a^{2}_{bb}$, (2) ${}_{bb}a^{1} \cdot a^{2}_{cc}$, (3) ${}_{bb}a^{1} \cdot c^{2}_{dd}$.

In case (1) the only replacement for a^1 that changes the main symbol component is of the form ${}_{dd}c^{1}$ and yields

$$ {}_{-}d^{1} \cdot {}_{d}d^{2} \cdot {}_{dd}c^{1} \cdot a^{2}_{bb} \cdot b^{1}_{b} $$

in t. Then $dcb \leftrightarrow dab$ is a rule in C and $h_1(t)$ is obtained from $h_1(s')$ by replacing dab by dcb, so that $h_1(s') \rightarrow_C h_1(t)$.

In cases (2) and (3) the symbols to the left of ${}_{bb}a^{1}$ must be ${}_{-}b^{1} \cdot {}_{b}b^{2}$. The only possible replacement for ${}_{bb}a^{1}$ that changes the main symbol component is of the form ${}_{bb}e^{1}$.

In case (2), the result in t is

$$ {}_{-}b^{1} \cdot {}_{b}b^{2} \cdot {}_{bb}e^{1} \cdot a^{2}_{cc} \cdot c^{1}_{c}. $$

Thus $bec \leftrightarrow bac$ is a rule in C and $h_1(t)$ is obtained from $h_1(s')$ by replacing bac by bec, so that $h_1(s') \rightarrow_C h_1(t)$.

In case (3), the result in t is

$$_{-}b^1 \cdot {}_{b}b^2 \cdot {}_{bb}e^1 \cdot c^2_{dd} \cdot d^1_{d}.$$

Thus both $bed \leftrightarrow bcd$ and $bad \leftrightarrow bcd$ are rules in C, and $h_1(t)$ is obtained from $h_1(s')$ by replacing bad by bed. Though this is not necessarily a single step of C, it is accomplished by two steps: $bad \rightarrow_C bcd \rightarrow_C bed$, so that $h_1(s') \rightarrow^*_C h_1(t)$, which concludes the proof of Theorem 2. □

6 Discussion

We have introduced mutation systems to model the evolution of a string subject to the effects of mutations and a fitness function. Some possible generalizations of our definition may be fruitful to explore: a population of evolving strings, a probabilistic or time-varying fitness function, or a fitness function that depends on comparing strings in the current population.

Comparing our mutation systems to Valiant's concept of evolvability [11] we note that his model is designed to explore the question of what functions can be efficiently approximated through a polynomial-time evolution process, while our model does not have a final ideal target, but instead has a variety of evolution pathways and outcomes defined by the mutation operator and the fitness function.

We have shown that mutation systems with point mutations and strictly 2-testable fitness functions can represent general computation, and therefore it is in general undecidable to predict whether one string can evolve into another in such systems. By contrast, for any k the class of strictly k-testable languages, and even the class of concatenations of strictly k-testable languages, are learnable in the limit from positive data [4,8]. A promising future direction is to explore the learnability of fitness functions given positive data derived from the evolution of one or more strings in a mutation system.

Acknowledgements. Raonne Barbosa Vargas is now employed by Microsoft Corporation. The authors thank David Eisenstat and Sarah Eisenstat for help with aspects of this paper.

References

1. Bennett, C.: Logical reversibility of computation. IBM J. Res. Develop., 525–532 (November 1973)
2. Bennett, C.H.: The thermodynamics of computation – a review. International Journal of Theoretical Physics 21, 905–940 (1982)
3. Brzozowski, J., Simon, I.: Characterizations of locally testable events. Discrete Mathematics 4, 243–271 (1973)
4. García, P., Vidal, E.: Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. IEEE Trans. Pattern Anal. Mach. Intell. 12, 920–925 (1990)

5. Head, T.: Splicing representations of strictly locally testable languages. Discrete Appl. Math. 87, 139–147 (1998)
6. Kari, J.: Theory of cellular automata: A survey. Theoretical Computer Science 334(1-3), 3–33 (2005)
7. Kim, S.M., McNaughton, R., McCloskey, R.: A polynomial time algorithm for the local testability problem of deterministic finite automata. Algorithms and Data Structures 382, 420–436 (1989)
8. Kobayashi, S., Yokomori, T.: Learning concatenations of locally testable languages from positive data. In: Arikawa, S., Jantke, K. (eds.) AII 1994 and ALT 1994. LNCS, vol. 872, pp. 407–422. Springer, Heidelberg (1994)
9. Lindgren, K., Nordahl, M.: Universal computation in simple one-dimensional cellular automata. Complex Systems 4, 299–318 (1990)
10. McNaughton, R.: Algebraic decision procedures for local testability. Theory of Computing Systems 8(1), 60–76 (1974)
11. Valiant, L.G.: Evolvability. J. ACM 56, 3:1–3:21 (2009)
12. Von Neumann, J.: Theory of self-reproducing automata. In: Burks, A.W. (ed.), University of Illinois Press, Urbana (1966)
13. Yokomori, T., Kobayashi, S.: Learning local languages and their application to DNA sequence analysis. IEEE Trans. Pattern Anal. Mach. Intell. 20, 1067–1079 (1998)

Classification of String Languages via Tiling Recognizable Picture Languages*

Marcella Anselmo[1], Dora Giammarresi[2], and Maria Madonia[3]

[1] Dipartimento di Informatica, Università di Salerno I-84084 Fisciano (SA) Italy
anselmo@dia.unisa.it
[2] Dipartimento di Matematica. Università di Roma "Tor Vergata",
via della Ricerca Scientifica, 00133 Roma, Italy
giammarr@mat.uniroma2.it
[3] Dip. Matematica e Informatica, Università di Catania, Viale Andrea Doria 6/a,
95125 Catania, Italy
madonia@dmi.unict.it

Abstract. We introduce the definition of string language S recognized via picture language P and prove that there is a one-to-one correspondence between a linear bounded automaton (LBA) for S and a tiling system for P. As consequence tiling systems become an alternative description for LBA that possibly exploits some geometric properties of lines and shapes inside the rectangular pictures. We are able to show a classification of sub-classes of context-sensitive languages via REC subclasses. Moreover we state some relations among languages in Chomsky's hierarchy (from regular up to context-sensitive) and the corresponding size of the picture languages that recognize them.

1 Introduction

Picture languages are sets of rectangular pictures, i.e. two-dimensional arrays of symbols chosen in a finite alphabet. Tiling recognizable picture languages were first introduced in [6] as a two-dimensional counterpart of regular string languages. Their definition is given by extending to two dimensions a characterization of recognizable string languages in terms of local languages and projections (cf. [5]). A local picture language is defined by means of a finite set of 2×2 pictures (called *tiles*), and the pair of a local picture language and a projection is called *tiling system*. The family of tiling recognizable languages is referred to as REC. Family REC is a robust class which can be defined using also other approaches (logic, automata, algebraic) and satisfies different types of closure properties; moreover REC coincides with regular string languages in the special case of one-row pictures ([7]). Nevertheless, we remark that, unlike regular string languages, tiling recognizable picture languages are not closed under complementation and therefore the family REC is intrinsically non-deterministic.

* Partially supported by MIUR Project *"Aspetti matematici e applicazioni emergenti degli automi e dei linguaggi formali"*, by ESF Project *"AutoMathA"* (2005-2010), by 60% Projects of University of Catania, Roma "Tor Vergata", Salerno (2009).

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 105–116, 2011.

Since its introduction, family REC has been intensively studied by investigating its properties, by examining relations with other models and by considering sub-families of particular interest (see e.g. [7,2,3,4,10,11,12]). In particular, in this paper, we will refer to the families of *unambiguous*, *row-unambiguous* and *deterministic tiling recognizable picture languages* (referred to as UREC, Row-UREC and DREC, respectively) that are defined by extending to two dimensions the concepts of unambiguous and deterministic automata computations.Those classes are all distinct and define a hierarchy inside REC ([1,2]).

In [10], M. Latteux and D. Simplot use picture languages to represent string languages. The *frontier of a picture* p, referred to as $fr(p)$, is defined as the string in the top row of p, while the *frontier of a picture language* L, referred to as $fr(L)$, is the string language containing the frontiers of all pictures in L. In [10] it is proved that the family of frontiers of tiling recognizable picture languages is exactly the family of context-sensitive (string) languages.

The Latteux and Simplot Theorem is a quite interesting theoretical result. Nevertheless, the way it is stated and its proof techniques do not provide a tool to push further the result. In this paper we will put in one-to-one correspondence LBA and tiling systems: a local picture p will represent a computation of an LBA on its frontier in a compact way. As a consequence a tiling system can be taken as an alternative description for an LBA and we can work to put in correspondence REC sub-families with sub-families of context-sensitive languages.

We associate a given string s with very simple pictures p such that the first row of p (i.e. the frontier) is equal to s while all the remaining positions contain the blank symbol. The interpretation for this kind of pictures will be of a string put on top of a whiteboard initialized with all blank symbols $\flat$s and this whiteboard is the place where to write the "calculation" to accept string s. We formally give the definition of *string language recognized via picture language* (where pictures are of the simple kind just described) and prove that a string language is context-sensitive if and only if it is recognized *via* a tiling recognizable picture language. Remark that if p belongs to a tiling recognizable picture language then p is the projection of a local picture p' and such p' is actually the "computation" to accept p. Then if a string s is recognized via a picture p then the corresponding local picture p' will be the "calculation" to accept s.

The proof of this theorem gives an algorithm to transform an LBA into a tiling system and vice-versa and therefore allows to consider tiling systems as an effective device to recognize strings. We show that, very interestingly, when we look at the local language corresponding to a tiling system we can "discover" some geometric properties used to recognize the frontier of the picture. On the other side, one could exploit reasoning on geometric properties of a local language for the LBA design. We illustrate this concept with different examples.

Then, the next results are obtained by taking tiling systems as effective descriptions of LBA. In particular we are able to show that recognizability via picture languages in REC sub-classes Row-UREC and UREC will correspond to deterministic and unambiguous context-sensitive languages, respectively. We use a characterization of family Row-UREC in terms of snake deterministic tiling

system given in [11] and prove that such kind of deterministic properties of the tiling system can be inherited by the LBA.

We remark that problems on LBA can be translated in the context of tiling recognizable picture languages. We mention here the most famous one: whether deterministic LBA are equivalent to non-deterministic ones (or, in complexity terminology, whether DSPACE(n) is equal to NSPACE(n)). This problem can be therefore restated as: "Is each string language S, recognized via picture languages in REC, recognized also via picture languages in Row-UREC?" We recall that, as picture languages classes, Row-UREC is strictly included in REC.

Another issue we consider here, relates the size of the pictures (actually the number of rows as function of the number of columns) with the type of the string language in the Chomsky's hierarchy. Remark that the size of the picture is not a measure of space complexity but of time complexity: in fact since pictures are local, they can be constructed row-by-row (starting from the top) maintaining always only the last row just calculated. We prove that if S is recognized via a picture language P then a string $s \in S$ of length n is recognized via a picture $p \in P$ of $O(1)$ or $O(n)$ or $2^{O(n)}$ rows depending whether S is regular, context-free or context-sensitive, respectively.

2 Preliminaries

We introduce some definitions about two-dimensional languages. The notation used and more details can be mainly found in [7].

A *picture* over a finite alphabet Σ is a two-dimensional rectangular array of elements of Σ. Given a picture p, let $p_{i,j}$ denote the symbol in p with coordinates (i, j), $|p|_r$ the number of rows and $|p|_c$ the number of columns; the pair $(|p|_r, |p|_c)$ is the *size* of p. The set of all pictures over Σ is denoted by Σ^{**}. A *two-dimensional language* (or *picture language*) over Σ is a subset of Σ^{**}.

In order to identify the symbols on the boundary of a picture p, we consider the *bordered picture* $\widehat{p}$ of size $(|p|_r + 2, |p|_c + 2)$ obtained by surrounding p with a special *boundary symbol* $\# \notin \Sigma$. A *tile* is a picture of dimension $(2, 2)$ and $B_{2,2}(p)$ is the set of all sub-blocks of size $(2, 2)$ of a picture p. Given a finite alphabet Γ, a two-dimensional language $L \subseteq \Gamma^{**}$ is *local* if there exists a set Θ of tiles over $\Gamma \cup \{\#\}$ (the set of *allowed blocks*) such that $L = \{p \in \Gamma^{**} | B_{2,2}(\widehat{p}) \subseteq \Theta\}$ and we will write $L = L(\Theta)$.

2.1 Tiling Recognizable Picture Languages

Tiling recognizable languages are defined as projection of local languages. More precisely, let Γ and Σ be two finite alphabets and $\pi : \Gamma \to \Sigma$ be a *projection* (π can be extended, in the usual way, to pictures and languages). A picture $p' \in \Gamma^{**}$ is a *pre-image* of $p \in \Sigma^{**}$ if $\pi(p') = p$. A *tiling system* 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. 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 over the alphabet Σ, is denoted by *REC(Σ)* or simply *REC*.

In all the paper, we will avoid to specify the alphabet of a family of languages, when it does not cause confusion.

Example 1. Consider the language L of square pictures over a one-letter alphabet, say $\Sigma = \{a\}$, that is pictures with same number of rows and columns. L is not a local language, but it belongs to REC. Indeed it can be obtained as the projection of the local language of squares over the alphabet $\{a, X\}$ in which all the symbols in the main diagonal and below it are X, whereas the remaining positions carry symbol a. Below it is given a picture $p \in L$ together with its pre-image p'. The reader can infer the set of tiles by taking all possible 2×2 sub-pictures of the bordered picture.

$$p = \begin{array}{|c|c|c|c|}\hline a & a & a & a \\ \hline a & a & a & a \\ \hline a & a & a & a \\ \hline a & a & a & a \\ \hline \end{array} \qquad p' = \begin{array}{|c|c|c|c|}\hline X & a & a & a \\ \hline X & X & a & a \\ \hline X & X & X & a \\ \hline X & X & X & X \\ \hline \end{array}$$

Observe that, given a tiling system for a language L, to check whether a given picture p belongs to L we have to find a pre-image in $L(\Theta)$. To do this, we start a computation process that takes $\widehat{p}$ and rewrites symbols in p in the local alphabet, in a way that is compatible with the projection π and with the set of allowed tiles. The process is accomplished following some scanning strategy (e.g. look for a tile matching the top-left corner of the picture, and then continue filling the positions on the first column, and so on, column by column). It terminates when all symbols in p are rewritten in the local alphabet. Note that, in general, such process is non-deterministic: even if a scanning strategy is fixed, at each step of the computation there can be several choices.

Taking into account this process of computation, in [1], the definition of determinism in REC was given. More precisely four types of determinism, one for each corner-to-corner direction of reading of a picture, were introduced. Observe that this is also the case for string languages: there can be given two notions of determinism according to the reading direction, classical determinism (from left-to-right) and co-determinism (from right-to-left). *DREC* denotes the class of all *deterministic* recognizable languages, that is languages that can be recognized by a deterministic tiling system along one of the four corner-to-corner directions. In this paper we principally deal with family $DREC_t$, that is the class of all languages that can be recognized by a tiling system that is deterministic along a direction from one top corner to the opposite corner.

Still referring to the computation process, a tiling system $(\Sigma, \Gamma, \Theta, \pi)$ for $L \subseteq \Sigma^{**}$ is said *unambiguous* if for any picture $p \in L$ there exists a unique local pre-image $p' \in L(\Theta)$. L is *unambiguous* if it is recognized by an unambiguous tiling system and *UREC* denotes the family of all unambiguous two-dimensional languages (see [6]). Note that this notion is not directed.

In [1], the *row-unambiguity* was introduced as an intermediate notion between determinism and unambiguity. There are two types of row-unambiguity, one along the direction from top-to-bottom (t2b for short) and another one along the opposite direction (b2t). A tiling system $(\Sigma, \Gamma, \Theta, \pi)$ is *t2b-unambiguous*

if for any pair of two rows $r_1 \in \Gamma^{**} \cup \{\#\}^{**}$ and $s \in \Sigma^{**}$, there exists at most one local row $r_2 \in \Gamma^{**}$, such that $\pi(r_2) = s$ and $B_{2,2}(p) \subseteq \Theta$ where $p = \begin{array}{|c|c|c|}\hline \# & r_1 & \# \\ \hline \# & r_2 & \# \\ \hline \end{array}$. A language is said *row-unambiguous* if it is recognized by a tiling system that is t2b- or b2t-unambiguous. *Row-UREC* denotes the class of row-unambiguous languages. In this paper we will be interested in *Row-UREC$_t$*, the class of all picture languages that are recognized by t2b-unambiguous tiling systems. Remark that DREC$\subset$ Row-UREC $\subset$ UREC$\subset$ REC, (see [1]), and the strict inclusions hold even if the alphabet is unary [2].

A tiling system $(\Sigma, \Gamma, \Theta, \pi)$ is *snake-deterministic* [11] if Γ and Θ can be partitioned as $\Gamma = \Gamma_1 \cup \Gamma_2$ and $\Theta = \Theta_1 \cup \Theta_2$ where $(\Sigma, \Gamma, \Theta_1, \pi)$ ($(\Sigma, \Gamma, \Theta_2, \pi)$, resp.) is a tiling system that is deterministic along the direction from top-left corner (from top-right corner, resp.) to the opposite corner, and Θ_1 (Θ_2, resp.) contains only tiles like $\begin{array}{|c|c|}\hline a_2 & b_2 \\ \hline a_1 & b_1 \\ \hline \end{array}$ ($\begin{array}{|c|c|}\hline a_1 & b_1 \\ \hline a_2 & b_2 \\ \hline \end{array}$ where $(a_1, b_1) \neq (\#, \#)$, resp.) with $a_i, b_i \in \Gamma_i \cup \{\#\}$ for $i = 1, 2$. It is also proved that the class of all picture languages that are recognized by t2b-unambiguous tiling systems coincides with the class of languages recognized by snake-deterministic tiling systems.

2.2 Family REC and Context-Sensitive String Languages

An interesting result due to M. Latteux and D. Simplot points out a relation between the family REC and the family of context-sensitive languages ([10]).

Recall that a context-sensitive (string) language is a language generated by a grammar with no length-decreasing rules or recognized by a linear-bounded automaton (LBA). Moreover one can always assume that such LBA use only the input cells: for the sequel we assume that an LBA is a non-deterministic Turing machine (TM) that uses only the portion of the tape containing the input string. We denote by *CSL* the family of all context-sensitive languages and by *DCSL* the family of all deterministic context-sensitive languages, i.e. those recognized by deterministic LBA. We recall that it is an open problem whether CSL=DCSL.

Let $p \in \Sigma^{**}$ be a picture of size (m, n). The *frontier of* p is the string corresponding to the first row of p that is: $fr(p) = p_{1,1}p_{1,2}\ldots p_{1,n}$. Moreover if L is a picture language, the *frontier of* L is defined as $fr(L) = \{fr(p) \,|\, p \in L\}$. It holds the following theorem.

Theorem 1. [Latteux, Simplot, 97] *Let F be a string language. Then F is context-sensitive if and only if there exists a picture language L in REC such that $F = fr(L)$.*

The proof of this theorem is based on the following observations. The computation of an LBA can be described as a sequence of instantaneous descriptions and if we write such descriptions one under the others we get a rectangular picture. The set of so defined pictures (i.e. corresponding to all accepting computations of a given LBA) is a language in REC. Conversely, given a tiling system for a picture language L, we can define a context-sensitive grammar G that generates exactly the frontiers of pictures in L by associating to each tile a rule for G.

3 Recognition of String Languages via Picture Languages

The Latteux and Simplot Theorem (Theorem 1) together with its original proof does not provide any tool to push further the result. Our goal is to put in correspondence sub-families of context-sensitive languages with REC sub-families and to take advantage of the geometric properties of tiling systems in the recognition process of the strings. With this aim we will put in one-to-one correspondence LBA and tiling systems: each local picture p will represent a computation of an LBA on a string $s = fr(p)$ in a compact way. As a consequence a tiling system can be taken as an alternative description for an LBA and we can introduce the definition of string language recognized via a tiling recognizable picture language. Moreover we will consider much simpler picture languages (all positions not in the frontiers contain blank symbol). The interpretation for a string recognized via pictures will be of a string put on top of a whiteboard initialized with all blank symbols $\flat$s.

We introduce the notation Σ^{**}_{Γ}, for some alphabets Σ and Γ, to indicate the set of pictures with first row symbols on Σ and all other symbols in Γ. When $\Gamma = \{\flat\}$ we will simply write $\Sigma^{**}_{\flat}$. We will always assume that $\flat \notin \Sigma$.

Definition 1. *A string language $S \subseteq \Sigma^*$ is* recognized via picture language P, *if $S = fr(P)$ and $P \subseteq \Sigma^{**}_{\flat}$.*

In the sequel we will be interested in string languages recognized via picture languages in REC or in some REC sub-families. In this case, referring to the mentioned interpretation for a string s recognized via a picture p, the local picture p', pre-image of p, will be the computation of s written in the whiteboard.

We first consider a simple example that will help to understand the ideas of the theorem we prove at the end of this section.

Example 2. Consider the string language $S = \{a^n b^n \mid n > 1\}$ over $\Sigma = \{a, b\}$ and let $P \subseteq \Sigma^{**}_{\flat}$ be the language containing pictures of $n + 1$ rows, whose first row is $a^n b^n$ while the n rows below contain the "blank" symbol $\flat$. It is like the $a^n b^n$ string put on top of two "empty" squares of side n. Language P is tiling recognizable as a projection of the local language L over $\Gamma = \{a, b, X, Y\}$ of pictures such that leftmost square has all positions in main diagonal and below filled with X, and positions above the main diagonal contain a, while the rightmost square has all positions in main diagonal and below filled with Y and positions above the diagonal contain b. Essentially the local language "verifies" that under the two strings a^n and b^n there are two equal squares. Below it is given the picture $p \in P$ with frontier $a^4 b^4$ together with its pre-image p'.

$p =$

a	a	a	a	b	b	b	b

$p' =$

a	a	a	a	b	b	b	b
X	a	a	a	Y	b	b	b
X	X	a	a	Y	Y	b	b
X	X	X	a	Y	Y	Y	b
X	X	X	X	Y	Y	Y	Y

Now consider the (usual) TM for S proposed in [9] and reported below.

$State \backslash Symbol$	a	b	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	(q_1, a, R)	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	(q_2, a, L)	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

It can be easily verified that there is a correspondence between the run of this TM on a^4b^4 and picture p'. In particular reading the rows of p' from top to bottom we find the shots of the tape contents between two head reversals. Informally we can say that the local picture p' is a compact representation of the LBA computation (the word "compact" is referred to the fact that each row represents several steps of the computation). With the next theorem we will show that this correspondence can be always made. As a consequence, tiling systems can be taken as alternative descriptions of TM. Moreover it is interesting to notice that the description via local language with its geometric shapes could describe the algorithm more immediately than the table of the TM transitions.

We are now ready to state our theorem. Remark that the statement resembles the one of Theorem 1 (instead of the whole family REC it refers to languages in $\Sigma^{*_\flat}$); nevertheless the proof uses very different techniques that allow us to gain the results given in Section 4. An LBA is called *sweeping* if it changes head direction and accepts only on the leftmost or rightmost symbol of the input.

Theorem 2. *A string language S is context-sensitive if and only if S is recognized via picture languages in REC.*

Proof. (*Sketch*) Let $S \subseteq \Sigma^*$ be a string language recognized by an LBA $\mathcal{A}$. Without loss of generality, we may suppose that $\mathcal{A}$ is a sweeping automaton and that the alphabet of symbols written by $\mathcal{A}$ is disjoint from Σ. Then a picture language P recognizing S can be obtained as the language recognized by a tiling system where the local symbols are the transitions of $\mathcal{A}$. A tile $\begin{array}{|c|c|}\hline \alpha & \gamma \\ \hline \beta & \eta \\ \hline \end{array}$, with α, β, γ, $\delta \in \Gamma$, is in Θ iff α and γ (β and η, resp.) are two subsequent transitions, while the symbol written by transition α (γ, resp.) is the symbol read by β (η, resp.). Corner and border tiles are defined according to initial and final states, and to transitions where the direction of the head is reversed (recall that $\mathcal{A}$ is a sweeping automaton). The projection maps a transition to a when the read symbol is $a \in \Sigma$, and to $\flat$ otherwise.

Conversely, let $S \subseteq \Sigma^*$ recognized via picture language P given by a tiling system $\mathcal{T}$. Then it is possible to associate with $\mathcal{T}$, a sweeping LBA $\mathcal{A}$ that recognizes $fr(P)$. The idea is that $\mathcal{A}$, during a computation on $w \in \Sigma^*$, attempts to recover, row by row, a pre-image p' of $p \in P$ such that $w = fr(p)$. The set of states is partitioned in the states used for moving rightwards and the ones used for moving leftwards. While the sweeping LBA is doing the i-th scanning of w (along some direction), if the head is in the j-th position, then it reads $p'_{(i,j)}$

and replaces it with $p'_{(i+1,j)}$ according to the set of tiles Θ. If the computation is successful, then, the concatenation row by row of the content of the tape, before the head reverses its direction, is exactly picture p'. Note that we can assume that S is also the frontier of the corresponding local language P'. □

3.1 Further Examples and Remarks

Previous theorem entitles recognizable picture languages to be an effective model of computation for context-sensitive languages. In particular the computation for a given string w of length n is done in a rectangle with n columns and the length of the computation is related to the number of rows of the pictures.

We show some other examples of tiling recognizable picture languages used to recognize some popular string languages. The aim is to emphasize as the local languages possibly exploit geometrical properties of lines and figures in a two-dimensional space for counting and comparing symbols in the input strings.

Example 2 (cont.) Consider again language $S = \{a^n b^n \mid n > 1\}$ in Example 2 recognized via P. Notice that P can be also accepted as projection of a different local language L_1 of pictures where the first row contains $a^n b^n$ while the n rows below contain two juxtaposed squares with a sort of "V" shapes filled with as to the left and bs to the right, while the two outside triangles are filled with Xs to the left and Ys to the right. The following figure represents a picture in L_1 and its "geometric shapes" used to count and compare symbols from the input string.

a	a	a	a	b	b	b	b
X	a	a	a	b	b	b	**Y**
X	**X**	a	a	b	b	**Y**	**Y**
X	**X**	**X**	a	b	**Y**	**Y**	**Y**
X	**X**	**X**	**X**	**Y**	**Y**	**Y**	**Y**

a	a	a	a	b	b	b	b
↘	↘	↘	↘	↙	↙	↙	↙
	↘	↘	↘	↙	↙	↙	
		↘	↘	↙	↙		
			↘	↙			

Notice that language L_1 corresponds to an LBA that accepts strings by matching the leftmost a with the last b, and then the second a with the second-to-last b, and so on, while L in Example 2 corresponds to an LBA that matches first a with the leftmost b, then the second a with the second b, and so on.

Another possibility is to recognize S via the language of pictures with $n+2$ rows and frontier $a^n b^n$: this can be obtained as projection of local language L_2 that contains pictures with a diagonal stripe of Xs of width n that starts below the as in the first row and then goes down moving one step right at each row till it overlays all bs.

a	a	a	a	b	b	b	b
X	**X**	**X**	**X**	b	b	b	b
a	**X**	**X**	**X**	**X**	b	b	b
a	a	**X**	**X**	**X**	**X**	b	b
a	a	a	**X**	**X**	**X**	**X**	b
a	a	a	a	**X**	**X**	**X**	**X**

a	a	a	a	b	b	b	b
↘	↘	↘	↘				
	↘	↘	↘	↘			
		↘	↘	↘	↘		
			↘	↘	↘	↘	
				↘	↘	↘	↘

Language L_2 corresponds to another strategy: the LBA reads the string from left-to-right and marks all the as, then reads again the string from left-to-right and moves all marks one position right and then re-starts from left-to-right moving all marks one position right and so on until eventually it succeeds in moving all marks to cover all and only the bs.

Remark that techniques of above example can be applied to recognize languages of strings $a^n b^n c^n$, or even palindromes ww^R and other similar variations.

Example 3. Let $S = \{ww \mid w \in \{a,b\}^*\}$. S is recognized via language of pictures with frontier ww and $n+1$ rows where n is the lenght of w. The corresponding local language is the set of pictures with first row containing ww on top of two juxtaposed squares. The information about any symbol in the first row is carried diagonally till the bottom border where continues to be carried one cell rightwards, and also down vertically till the bottom border. In the bottom positions of rightmost square it is checked whether the two symbols match.

4 Classification of String Languages by Picture Languages

In this section we use Theorem 2 and the technique introduced in its proof to push further the result and establish a correspondence among various kinds of tiling recognizable picture languages and sub-classes of context-sensitive string languages. In particular we give two main classifications. The first one considers REC sub-classes DREC$_t$, *Row*-UREC$_t$, and UREC, while the second one relates the size of the pictures (actually the number of rows as function of the number of columns) with the type of the string language in the Chomsky's hierarchy.

We start by considering the family of row-unambiguous tiling recognizable languages (see Section 2).

Proposition 1. *A string language S is deterministic context-sensitive if and only if S is recognized via picture languages in Row-UREC$_t$.*

Proof. (*Sketch*) Let $S \subseteq \Sigma^*$ be a string language recognized by a deterministic LBA $\mathcal{A}$. Also in the deterministic case, we can suppose without loss of generality, that $\mathcal{A}$ is a sweeping automaton. Consider then the tiling system $\mathcal{T}$, obtained as in Theorem 2, that recognizes picture language $P \in \Sigma^{*_b}$ recognizing S. Since $\mathcal{A}$ is deterministic, then $\mathcal{T}$ is t2b-unambiguous and $P \in$ Row-UREC$_t$.

For the converse, suppose $S \subseteq \Sigma^*$ is a string language recognized via picture language P, where P is a t2b-row unambiguous language. Then, from [11], P is recognized by a snake-deterministic tiling system $\mathcal{T}$. Applying the construction of Theorem 2 to $\mathcal{T}$, we obtain an LBA $\mathcal{A}$ recognizing S that is deterministic. □

We recall that a context-sensitive language is unambiguous if it can be recognized by an unambiguous LBA. Using techniques as in the proof of Proposition 1 it can be proved an analogous result for UREC and unambiguous context-sensitive languages (UCSL). Previous results enable us to associate with REC sub-classes their counterparts as sub-families of context-sensitive languages. We summarize those results in the following theorem.

Theorem 3. *Let L be a string language.*
1. L is CSL iff L is recognized via a picture language in REC.
2. L is UCSL iff L is recognized via a picture language in UREC.
3. L is DCSL iff L is recognized via a picture language in Row-UREC$_t$.

We consider also string languages recognized via picture languages in DREC$_t$: we refer to this family as $\mathcal{L}(DREC_t)$. By similar technique as in Theorem 2, it can be proved that $L \in \mathcal{L}(DREC_t)$ iff L is recognized by a special kind of sweeping deterministic LBA that writes only when moving left-to-right and remains in the same state when moving right-to-left. For this we affirm that $\mathcal{L}(DREC_t)$ is included in the class of deterministic context-sensitive languages but we are not able to give a precise characterization of this class.

Another interesting point of view to classify pictures that recognize strings is to consider their sizes. Recalling the interpretation of the picture as a whiteboard where to write the computation for the string, we remark that, in the construction of a tiling system from an LBA, the size of the picture (i.e. the "area" of the whiteboard) is not a measure of space but of time instead (each position is a step of the corresponding LBA). Let us give the following definition.

Definition 2. *S is recognized via P within height $f(n)$ if for any w of length n, there exists $p \in P$ with $fr(p) = w$ such that $|p|_r \leq f(n)$*

Proposition 2. *A string language S is context sensitive if and only if it can be recognized within height $2^{O(n)}$ via a picture language in REC.*

Proof. Any string language S recognized via a picture language in REC is context sensitive by Theorem 2.

Let now $S \in CSL$ and $\mathcal{A}$ be an LBA recognizing it. Consider the picture language $P \in REC$ recognizing S constructed as in the proof of Theorem 2. We claim that for any $w \in S$ of length n, there exists $p \in P$ such that $w = fr(p)$ and $|p|_r \leq k^n$ for some constant k. Suppose by contradiction that there does not exist a k as above, and let $w \in S$ such that any $p \in P$ with $w = fr(p)$ has more than an exponential number of rows in n. Therefore $|p|_r \geq \gamma^n$, where γ is the cardinality of the local alphabet in the tiling system for P. Then a pre-image of p has two repeated rows. By removing the part of computation of the LBA between the two repeated rows, we obtain another valid computation of the LBA and hence another picture $p_{min} \in P$ with $|p_{min}|_r \leq \gamma^n$ and $fr(p_{min}) = w$. □

Proposition 3. *A context-free string language S can be recognized within height $O(n)$ via a picture language in REC.*

Proof. A language S is context-free iff there exists a constant k such that S is recognized by an LBA that can modify the symbol in a tape cell only in its first k visits of the cell (cf. [8]). Then the corresponding tiling system, constructed as in the proof of Theorem 2, contains only pictures with $|p|_r \leq k \cdot |p|_c$. □

Remark 1. The converse of previous Proposition 3 does not hold. For instance, language $S = \{ww \mid w \in \{a, b\}^*\}$ is recognized within height $O(n)$ via a picture language in REC as in Example 3, but it is not context-free.

Proposition 4. *A string language S is regular iff it can be recognized within height $O(1)$ via a picture language in $DREC_t$.*

Proof. A language S is regular iff if is recognized by a deterministic finite automaton that can be viewed as a deterministic tiling system recognizing S taken as a language of pictures with a single row. □

We collect the results of previous propositions in the following theorem that relates the Chomsky's hierarchy languages with some bounds on corresponding tiling recognizable picture languages.

Theorem 4. *Let S be recognized via a picture language in REC.*
1. If S is context-sensitive then it can be recognized within height $2^{O(n)}$.
2. If S is context-free then it can be recognized within height $O(n)$.
3. If S is regular then it can be recognized within height $O(1)$.

5 Conclusions and Future Works

The paper presented tiling recognizable picture languages as a computational model for context-sensitive string languages. We proved that properties of subfamilies of REC can be related to properties of sub-classes of context-sensitive languages and therefore questions on strings can be translated in the context of pictures. In particular, denoting by $\mathcal{L}$(F) the family of string languages recognized via picture languages in family F, we have proved that

$$\begin{array}{ccccc} \mathcal{L}(DREC_t) \subseteq & \mathcal{L}(Row{-}UREC_t) & \subseteq & \mathcal{L}(UREC) & \subseteq \mathcal{L}(REC) \\ & \| & & \| & \| \\ & DCSL & & UCSL & CSL \end{array}$$

We left open the problem of characterizing class $\mathcal{L}(DREC_t)$; moreover we do not know if some of those inclusions are strict. Remark that we cannot use the fact that inclusions among corresponding picture languages classes are all strict (cf. [1]), and in fact the problem whether DCSL is equal to CSL (in complexity theory, whether DSPACE(n)=NSPACE(n)) is a longstanding open problem.

Those problems in the "picture world" can be states as follows. Define two languages L_1 and L_2 to be *fr-equivalent* iff $fr(L_1) = fr(L_2)$ and then investigate the quotient of REC and its sub-classes with respect to this relation.

We conjecture that $\mathcal{L}(DREC_t)$ is strictly included in the class DCSL and we have different candidate languages. Following the examples in this paper we can verify that $\mathcal{L}(DREC_t)$ includes both languages of palindromes ww^r and of squares ww, but we believe that the language of strings that contain a sufficiently long factor of type ww^r or of type ww are not in $\mathcal{L}(DREC_t)$.

Acknowledgments

We sincerely thank Alberto Bertoni and Giovanni Pighizzini for having pointed out some results on context-free and context-sensitive languages.

References

1. Anselmo, M., Giammarresi, D., Madonia, M.: Deterministic and unambiguous families within recognizable two-dimensional languages. Fundamenta Informaticae 98(2-3), 143–166 (2010)
2. Anselmo, M., Madonia, M.: Deterministic and unambiguous two-dimensional languages over one-letter alphabet. Theoretical Computer Science 410(16), 1477–1485 (2009)
3. Cherubini, A., Pradella, M.: Picture languages: From Wang tiles to 2D grammars. In: Bozapalidis, S., Rahonis, G. (eds.) CAI 2009. LNCS, vol. 5725, pp. 13–46. Springer, Heidelberg (2009)
4. Crespi Reghizzi, S., Pradella, M.: Tile Rewriting Grammars and Picture Languages. Theoretical Computer Science 340(2), 257–272 (2005)
5. Eilenberg, S.: Automata, Languages and Machines, vol. A. Academic Press, New York (1974)
6. Giammarresi, D., Restivo, A.: Recognizable picture languages. International Journal Pattern Recognition and Artificial Intelligence 6(2-3), 241–256 (1992)
7. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Salomaa, A., Rozenberg, G. (eds.) Handbook of Formal Languages, vol. 3, pp. 215–267. Springer, Heidelberg (1997)
8. Hibbard, T.N.: A generalization of context-free determinism. Information and Control 11(1/2), 196–238 (1967)
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation, 2nd edn. Addison-Wesley-Longman (2001)
10. Latteux, M., Simplot, D.: Context-sensitive string languages and recognizable picture languages. Information and Computation 138(2), 160–169 (1997)
11. 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)
12. 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)

A Simple and Efficient Universal Reversible Turing Machine

Holger Bock Axelsen and Robert Glück

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

Abstract. We construct a universal reversible Turing machine (URTM) from first principles. We take a strict approach to the semantics of reversible Turing machines (RTMs), under which they can compute exactly all *injective*, computable functions, but not non-injective ones. The natural notion of universality for RTMs is *RTM-universality*, where programs are considered part of both input and output of a URTM.

The machine described here is the first URTM which does not depend on reversibilizing any existing universal machine. The interpretive overhead of the URTM is limited to a (program dependent) constant factor slowdown, with no other complexity-wise cost *wrt* time and space. The URTM is also able to function as an *inverse interpreter* for RTMs at no asymptotic cost, simply by reversing the string representing the interpreted machine.

1 Introduction

Reversible computation models are time-invertible, forward and backward deterministic. For stateful computation models this means that not only the next, but also the previous state is uniquely defined at all times. There is a wide range of reversible computation models, from cellular automata [9] and pushdown automata [7], over logic circuits [16,13] to quantum computing [5,14]. Reversible computing principles also finds use in program transformations such as inversion [15], reversible programming [3,17] and translation [1], and bidirectional model transformation [12,6].

Here, we are concerned with the foundational question of *universality* for reversible Turing machines (RTMs.) Universality is the (semantic) notion of a specific machine being able to perform any computation possible within a computation model. In programming languages, the equivalent notion is that of a *self-interpreter*.

Previous work by Morita *et al.* exists asserting the universality (in the sense of full Turing completeness) of an RTM [11]. On close examination, their construction implicitly allows for a semantical relaxation with respect to what function is being computed. Specifically, one is allowed to *extract* part of the tape and consider it the output of the computation, whereby information is irreversibly lost.

In recent work [2] we studied the RTMs under the stricter viewpoint that the *entire* configuration must be considered for the output. This decouples the

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 117–128, 2011.

semantic functionality of RTMs from any particular transformation, such as Bennett's method [4]. However, under this interpretation reversibility of a machine implies injectivity of the computed function. The RTMs are then *not* fully Turing complete: They cannot simulate irreversible Turing machines without changing their functional behavior, and in fact cannot even faithfully simulate all *reversible* Turing machines. To amend this, we introduced a more natural notion of universality for reversible Turing machines, *RTM-universality*: An RTM is RTM-universal (is a URTM) if it can simulate any RTM while also remembering the simulated machine's program text.

Here, for the first time, we show how to construct a 3-tape URTM from first principles. The URTM is efficient in the sense that the asymptotic complexity of the interpreted RTM is preserved. The interpretive overhead is limited to a program dependent constant time factor, and there is no change in the space behavior except for adding a short string encoding the simulated internal state. Furthermore, the URTM can function as an inverse interpreter at *no* asymptotic cost. This is the first demonstration of a URTM with such properties.

2 Reversible Turing Machines

We here define the reversible Turing machines (RTMs). We state only results and properties relevant here. For a more complete exposition, see [2,4].

Definition 1 (Turing machine). *A TM T is a tuple $(Q, \Sigma, \delta, b, q_s, q_f)$ where Q is a finite set of states, Σ is a finite set of tape symbols, $b \in \Sigma$ is the blank symbol,*

$$\delta \subseteq \Delta = (Q \times [(\Sigma \times \Sigma) \cup \{\leftarrow, \downarrow, \rightarrow\}] \times Q)$$

is a partial relation defining the transition relation, $q_s \in Q$ is the starting state, and $q_f \in Q$ is the final state. There must be no transitions leading out of q_f nor into q_s. Symbols $\leftarrow$, $\downarrow$, $\rightarrow$ represent the three shift directions (left, stay, right).

Note that we use a *triple format* for the transition relation, with two kinds of rule. A *symbol rule* $(q, (s, s'), q') \in \delta$ says that in state q, if the tape head is reading symbol s, write s' and change into state q'. A *move rule* $(q, d, q') \in \delta$ says that in state q, move the tape head in direction d and change into state q'. This is easily extended to k-tape machines, where we have

$$\Delta = (Q \times [(\Sigma \times \Sigma)^k \cup \{\leftarrow, \downarrow, \rightarrow\}^k] \times Q) \ .$$

The *configuration* of a TM is a tuple $(q, (l, s, r)) \in Q \times (\Sigma^* \times \Sigma \times \Sigma^*)$, where q is the internal state, $l, r \in \Sigma^*$ are the left and right parts of the tape (as strings), and $s \in \Sigma$ is the current symbol being scanned. A TM T in configuration C *leads to* configuration C', written as $T \vdash C \rightsquigarrow C'$, in a single *computation step* in the obvious manner defined by the transition relation.

Definition 2 (Local fwd/bwd determinism). *A TM $T = (Q, \Sigma, \delta, b, q_s, q_f)$ is* local forward deterministic *iff for any distinct pair of triples $(q_1, a_1, q_1') \in \delta$ and*

$(q_2, a_2, q_2') \in \delta$, *if* $q_1 = q_2$ *then* $a_1 = (s_1, s_1')$ *and* $a_2 = (s_2, s_2')$, *and* $s_1 \neq s_2$. *A TM* T *is* local backward deterministic *iff for any distinct pair of triples* $(q_1, a_1, q_1') \in \delta$ *and* $(q_2, a_2, q_2') \in \delta$, *if* $q_1' = q_2'$ *then* $a_1 = (s_1, s_1')$ *and* $a_2 = (s_2, s_2')$, *and* $s_1' \neq s_2'$.

We say a TM is *reversible* iff it is locally forward and backward deterministic.

As examples, the rules $(\mathtt{q}, (\mathtt{a}, \mathtt{b}), \mathtt{p})$ and $(\mathtt{q}, (\mathtt{a}, \mathtt{c}), \mathtt{p})$ respect backward determinism (but not forward determinism); rules $(\mathtt{q}, (\mathtt{a}, \mathtt{b}), \mathtt{p})$ and $(\mathtt{r}, (\mathtt{c}, \mathtt{b}), \mathtt{p})$ are not backward deterministic; and neither are $(\mathtt{q}, (\mathtt{a}, \mathtt{b}), \mathtt{p})$ and $(\mathtt{r}, \rightarrow, \mathtt{p})$.

The semantic function of a TM is defined by its input/output behavior on standard configurations. A TM is in *standard configuration* iff the tape head is to the immediate left of a finite, blank-free string $s \in (\Sigma \backslash \{b\})^*$, and the rest of the tape is blank, i.e., it is in configuration $(q, (\varepsilon, b, s))$ for some state q.[1]

Definition 3 (String transformation semantics). *The* semantics $[\![T]\!]$ *of a TM* $T = (Q, \Sigma, \delta, b, q_s, q_f)$ *is given by the relation*

$$[\![T]\!] = \{(s, s') \in ((\Sigma \backslash \{b\})^* \times (\Sigma \backslash \{b\})^*) \mid T \vdash (q_s, (\varepsilon, b, s)) \rightsquigarrow^* (q_f, (\varepsilon, b, s'))\} \ .$$

A computation with a TM is as follows: From starting state q_s with input s in standard configuration $(q_s, (\varepsilon, b, s))$, run the machine until it halts in standard configuration $(q_f, (\varepsilon, b, s'))$ with output s', or diverges. We say T *computes* function f iff $[\![T]\!] = f$. Under this semantics, the reversibility of individual steps leads directly to injectivity in terms of functional behavior.

Theorem 1 (RTMs are injective [2]). *If* T *is an RTM, then* $[\![T]\!]$ *is an injective function.*

Lemma 1 (RTM inversion, Bennett [4]). *Given an RTM* $T = (Q, \Sigma, \delta, b, q_s, q_f)$, *the RTM* $T^{-1} \stackrel{\text{def}}{=} (Q, \Sigma, inv(\delta), b, q_f, q_s)$ *computes the inverse function of* $[\![T]\!]$, *i.e.* $[\![T^{-1}]\!] = [\![T]\!]^{-1}$, *where* $inv : \Delta \rightarrow \Delta$ *is defined as*

$$\begin{aligned} inv(q, (s, s'), q') &= (q', (s', s), q) & inv(q, \leftarrow, q') &= (q', \rightarrow, q) \\ inv(q, \downarrow, q') &= (q', \downarrow, q) & inv(q, \rightarrow, q') &= (q', \leftarrow, q) \ . \end{aligned}$$

This means that an RTM can be inverted into another RTM very easily.

Theorem 2 (Bennett's method [4]). *Given a 1-tape TM* T, *there exists a 3-tape RTM* $B(T)$, *s.t.* $[\![B(T)]\!](x) = (x, [\![T]\!](x))$.

This seminal result states that any TM can be *reversibilized*, *i.e.* turned into an RTM. It is important to note that Bennett's method does *not* not preserve semantics, $[\![T]\!] \neq [\![B(T)]\!]$, as the output of $B(T)$ includes the input x to T.[2]

Theorem 3 (Expressiveness [4,2]). *The RTMs can compute* exactly all *injective computable functions. That is, for every 1-tape TM* T *such that* $[\![T]\!]$ *is an injective function, there is a 3-tape RTM* T' *such that* $[\![T]\!] = [\![T']\!]$.

[1] The empty string ε denotes the infinite string of blanks b^ω, and is usually omitted.

[2] Output values that are added to ensure reversibility are known as *garbage* in reversible computing.

Thus, even though Theorem 1 restricts RTMs to injective (computable) functions, *all* of these are nevertheless in scope. Finally, we only need 1 tape and 3 symbols for any RTM computation.

Theorem 4 (Robustness [2]). *Let T be a k-tape, m-symbol RTM. Then there exists a 1-tape, 3-symbol RTM T' s.t.*

$$[\![T]\!](x_1, \ldots, x_k) = (y_1, \ldots, y_k) \quad \textit{iff} \quad [\![T']\!](e(\langle x_1, \ldots, x_k \rangle)) = e(\langle y_1, \ldots y_k \rangle)\,,$$

where $\langle \cdot \rangle$ is the convolution of tape contents, and $e(\cdot)$ is a binary encoding.

This is used to simplify our construction of a universal RTM, below.

2.1 Universality for RTMs

A universal machine is a machine that can simulate the (functional) behavior of any other machine. Usually, a universal TM U is defined as a *self-interpreter* for Turing machines:

$$[\![U]\!](\ulcorner T \urcorner, x) = [\![T]\!](x)\ .$$

Here, $\ulcorner T \urcorner \in \Sigma^*$ is a Gödel number (or *program text*) representing some TM T. However, $[\![U]\!]$ is a non-injective function (even if T has to be an RTM), so the RTMs are not universal in the classical sense. In [2], the authors therefore argued to define universality for the RTMs as follows.

Definition 4 (RTM-universality [2]). *An RTM U_R is* RTM-universal *(or, a* URTM*) iff for all RTMs T and all (blank-free) inputs $x \in \Sigma^*$,*

$$[\![U_R]\!](\ulcorner T \urcorner, x) = (\ulcorner T \urcorner, [\![T]\!](x))\ .$$

Theorem 5 (URTM existence [2]). *There exists an RTM U_R, such that U_R is RTM-universal.*

This follows directly from the expressiveness of the RTMs, Theorem 3.

3 A First-Principles URTM

We shall now describe the design and inner workings of a URTM constructed from first principles.

Besides novelty, our main motivation for constructing such a machine is to avoid any use of reversibilization and to limit the interpretive overhead to a constant factor. Nothing forces us to mechanically reversibilize an irreversible (classically universal or RTM-universal) machine when constructing a URTM, and reversibilizations come with considerable drawbacks. Most importantly, they change the asymptotic complexities of the programs. For instance, the URTM of Theorem 5 [2] relies on Bennett's trick [4], which means that the URTM uses as much (temporary) space as time, regardless of the space usage of the interpreted

program. This change in complexity is a highly undesirable side effect of using reversibilization.[3]

Such asymptotic inefficiency is not necessary: A URTM can (and should) conserve the asymptotic complexities of the interpreted machines (up to a program dependent constant.) However, no URTM with such properties has been exhibited until now.

Structure and Scope. We know that the 1-tape, 3-symbol RTMs are sufficient to express every injective computable function. We can therefore restrict our URTM to interpret exactly these machines. (We shall refer to the interpreted machine as T.)

We aim for a "textbook" structure for our URTM, with three tapes:

1. The first *work tape* will directly correspond to T's single tape.
2. The second *program tape* will hold the program text $\ulcorner T \urcorner$ (described below.)
3. The third *state tape* (blank at start and halt) will hold the encoding of the internal state of the interpreted machine T during the simulation (using the same encoding used for states in $\ulcorner T \urcorner$.)

Our focus here is on simplicity and efficiency, rather than minimization, so it shall not concern us that the URTM has a large number of states and symbols. However, for the same reason we shall not show the complete rule table here.

Program Encoding. We encode the interpreted 1-tape, 3-symbol machine $T = (Q, \{b, \mathtt{0}, \mathtt{1}\}, \delta, b, q_s, q_f)$ by a program text $\ulcorner T \urcorner$ as follows: The program is a string listing the rules in δ. The first rule in this program text *must* be the single rule that leaves the starting state q_s, and the final rule *must* be the single rule that enters the halting state q_f. With rules individually translated as below, such a string is sufficient to uniquely specify T.[4]

For the actual string $\ulcorner T \urcorner$ we use the alphabet $\Sigma = \{b, \mathtt{0}, \mathtt{1}, \mathtt{B}, \mathtt{S}, \mathtt{M}, \mathtt{\#}\}$. The two rule types (symbol and move) are translated by

$$\begin{aligned} \mathit{trans}(q, (s, s'), q') &= \mathtt{S\#}\mathrm{enc}_Q(q)\mathtt{\#}\mathrm{enc}_\Sigma(s)\mathrm{enc}_\Sigma(s')\mathtt{\#}\mathrm{rev}(\mathrm{enc}_Q(q'))\mathtt{\#S} \\ \mathit{trans}(q, d, q') &= \mathtt{M\#}\mathrm{enc}_Q(q)\mathtt{\#}\mathrm{enc}_D(d)\mathtt{\#}\mathrm{rev}(\mathrm{enc}_Q(q'))\mathtt{\#M}\ , \end{aligned}$$

where $\mathrm{enc}_Q : Q \to \{\mathtt{0},\mathtt{1}\}^{\lceil \log |Q| \rceil}$ is some injective binary encoding of the states in Q, and where

$$\mathrm{enc}_\Sigma(s) = \begin{cases} \mathtt{B} & \text{if } s = b \\ s & \text{otherwise}\ , \end{cases} \qquad \mathrm{enc}_D(d) = \begin{cases} \mathtt{10} & \text{if } d = \leftarrow \\ \mathtt{BB} & \text{if } d = \downarrow \\ \mathtt{01} & \text{if } d = \rightarrow\ , \end{cases}$$

encode the 3 symbols of T and the possible directions. Finally, $\mathrm{rev}(\cdot)$ simply reverses a string. The use of the special symbol $\mathtt{B}$ rather than the actual blank

[3] In addition to the functional redundancy of conserving the inputs, this complexity-wise inefficiency is a reason for discarding Bennett's suggestion of $B(U)$ as a universal machine.

[4] Note that a specific T may have more than one representation, as δ is unordered.

symbol b ensures that the encoding of a given machine will be a blank-free string that can be given in standard configuration form. As an example, the simple RTM $T = (\{q_0, q_1, q_2, q_3\}, \{b, \mathtt{0}, \mathtt{1}\}, \delta, b, q_0, q_3)$, with transition relation

$$\delta = \{(q_0, \rightarrow, q_1), (q_1, (\mathtt{0}, \mathtt{1}), q_2), (q_1, (\mathtt{1}, \mathtt{0}), q_2), (q_2, \leftarrow, q_3)\} \ ,$$

can be represented by program text

$$\ulcorner T \urcorner = \mathtt{M\#00\#01\#10\#MS\#01\#01\#01\#SS\#01\#10\#01\#SM\#10\#10\#11\#M} \ ,$$

where enc_Q is given by $q_0 \mapsto \mathtt{00}$, $q_1 \mapsto \mathtt{01}$, $q_2 \mapsto \mathtt{10}$ and $q_3 \mapsto \mathtt{11}$.

This encoding has the great advantage that we can perform program inversion by simply reversing the string, $\mathrm{rev}(\ulcorner T \urcorner) = \ulcorner T^{-1} \urcorner$. (String reversal can be performed in linear time by a simple 2-tape RTM.)

URTM Program. The URTM program has the following overall structure.

1. Copy the starting state q_s (first state of first rule) onto the state tape.
2. Sequentially try to apply each rule on the program tape, from left to right.
3. When all rules have been tried, compare the halting state q_f (last state of last rule) with the encoding of the current state q_c (on the state tape). If identical, clear q_c reversibly, rewind the program tape, and halt.
4. If q_c and q_f are not identical, rewind the program tape head and go to 2.

Thus, the URTM program consists of two nested loops: an inner loop where we try to apply each of the rules in the interpreted program to the current simulated configuration, and an outer loop where we test for the halting condition.

This is a straightforward and well-known design for a universal machine. The difficulty lies in that it now has to be done completely reversibly, which poses considerable challenges. For instance, there is control flow confluence at step 2, which is a source of backward non-determinism. In this case the situation is resolved by testing the equality of the simulated state with the starting state: They are only equal at the start of the simulation (*i.e.*, if we came from step 1), and (by definition) must be different thereafter (*i.e.*, if we came from step 4). This is closely related to the use of entry assertions in loops in reversible programming [17].

Fig. 1 shows a state diagram of the program: nodes are states, and edges are the actions of the associated rules. We have used a notation mixing ordinary state diagrams with reversible flowcharts [18]. The *diamond* (test) and *circle* (assertion) with inscribed expressions are reversible control flow operators (CFOs.) The expression of the assertion must be true when entering from the branch marked "true", and false if entering from the branch marked "false". The CFOs are here used as shorthand for RTM programs for comparing strings, as described below.

String Comparison. A central functionality that we rely on throughout the machine is string comparison. We must continuously compare the encoding of the current state (on the state tape) with the encodings of states in the rules

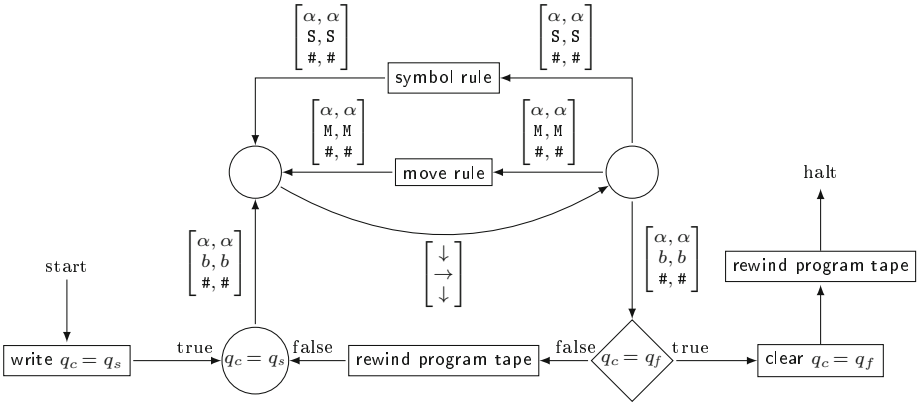


Fig. 1. Overall URTM program structure. The inner loop (topmost, written in ordinary state diagram form) sequentially tries to apply each rule on the program tape, and the outer loop (bottommost, written with shorthand reversible flowchart notation) tests for the halting (and starting) conditions. The symbol rule and move rule phases test for and apply the current rule on the program tape, passing over it in process. The write and clear phases initialize and clear the state tape. Edges with the symbolic variable α encode for three different transitions, with $\alpha \in \{b, 0, 1\}$.

of the interpreted machine (on the program tape). How does this work in the reversible setting?

Assume that we want to check strings $\#s_1 \ldots s_n\#$ and $\#t_1 \ldots t_n\#$ (each on a separate tape) for equality, moving the tape heads from the starting # to the terminating # in the process. As usual, we can scan the cells from left to right until either the termination symbol # is reached, or a mismatch is found. If the rightmost # symbol is reached without a mismatch, then the strings are equal (and this information is stored in the internal state). If there is a mismatch, however, then we cannot simply pass directly over the rest of the string, like one would do in the irreversible case.[5] Instead, we rewind until the starting # and then pass obliviously over both strings until the string terminator is found. Fig. 2 shows a transition diagram for this functionality. Note that this is done without writing anything to any of the tapes. Also note that the comparison still takes only linear time, the same as the irreversible case.

A central insight from reversible flowcharts tells us that the inverse of a *test* (a conditional split in control flow) is an *assertion* (a conditional join of control flow) [18]. Thus, if we invert the transition diagram in Fig. 2 as specified in Lemma 1, we get an RTM program that can *merge* two control branches if we know two strings are equal in one branch, and different in the other. We did this in the overall URTM program in Fig. 1.

[5] The first mismatch is special, in that the prefixes preceding it are equal. If we only do a single pass of the strings, then the resulting machine cannot be reversible: In reverse mode, it would have to predict exactly when the prefixes are equal without having visited them which is clearly impossible.

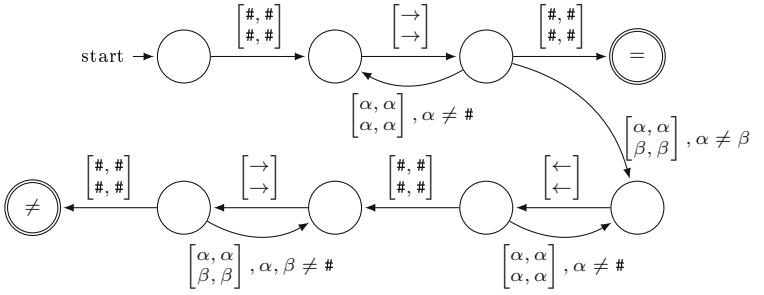


Fig. 2. 2-tape RTM state transition diagram for comparing strings $\#s_1 s_2 \ldots s_n\#$ and $\#t_1 t_2 \ldots t_n\#$ reversibly. (The reader is encouraged to verify the reversibility of the transitions.) The tape heads start at the left # and end at the right #. Edges with symbolic variables α and β encode multiple rules, subject to side conditions (e.g., that $\alpha \neq \#$).

Testing and Applying a Rule. We can distinguish symbol and move rules by their two encompassing `S` or `M` symbols, so we have separate subroutines for each case. We shall here only show the more involved symbol rule.

Application of a given transition rule $(q, (s, s'), q')$ given current state q_c and current symbol s_c is done as follows: We first compare states q and q_c. If they match, we compare symbols s and s_c. If these also match, we apply rule, *i.e.*, perform the substitution specified by the rule, changing the current state to q' and the current symbol to s'. (Appendix A shows the straight-line state transition diagram for this.)

This means that there are three distinct branches for the control flow. An irreversible machine would be able to simply merge these directly at the exit. This is *not* an option in the reversible setting, as it breaks backward determinism. We solve this problem as follows.

We have two disjoint branches where the rule was not applicable: One with a state mismatch $q \neq q_c$, and one where the states matched $q = q_c$ but the tape symbols were different $s \neq s_c$. Thus, we can reversibly merge the control flow of the two possible failures by comparing the current state to the source state of the symbol rule, using the inverted string comparator above.

The key problem is to merge the branch where the rule was applied to the one where it was not. For this we exploit the reversibility of the interpreted machine. Specifically, *only* if rule $(q, (s, s'), q')$ was just applied to cause the step $T \vdash (q, (l, s, r)) \rightsquigarrow (q', (l, s', r))$ can the current state and symbol be q' and s' simultaneously: reversibility guarantees it. With this insight, we can perform essentially the inverse of the testing we did for applicability to merge the branches.

A reversible flowchart for testing and application of a symbol rule is shown in Fig. 3. We have left out a few unimportant details (such as moving the program tape head across string terminators, etc.) Note the use of (reversible) string comparison for both splits and joins in control flow.

Since there is a fixed number of string comparisons, the symbol rule subroutine only takes time linear in the size of the rule, and importantly does not need to inspect any of the other rules in the program. So, even though reversibility is

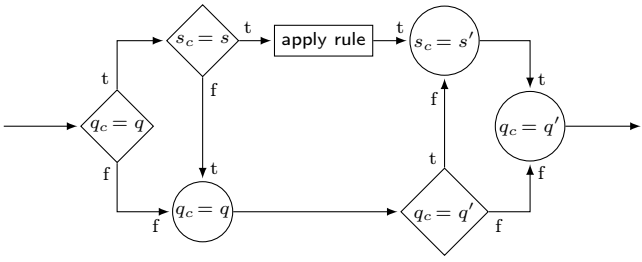


Fig. 3. Reversible flowchart for application of a symbol rule $(q, (s, s'), q')$, given current state q_c and current symbol s_c. The string comparisons are (implicitly) organized such that the program tape head moves from the left `S` in the encoding of the rule to the right `S`.

a *global* property of a program, we can still rely on it for the application of individual rule as a *peephole* property.[6] Note also that the sole change to the tape contents is the substitution of symbols on the work tape, and possibly an update of the state tape.

The case of the move rule is similar.

Asymptotic Complexity. In each pass over the program text at least one rule is applied and the work and state tapes are updated. This means that the interpretation cost in terms of time behavior is a slowdown proportional to the size of the program. A similar program dependent slowdown is also seen for irreversible UTMs. With respect to space, the URTM completely follows the interpreted machine, with only the addition of the string encoding the current simulated state (dominated by the size of the program).

Given a specific program $\ulcorner T \urcorner$, the URTM thus conserves the asymptotic complexities of the machine T.

Inverse Interpretation. As mentioned, our chosen encoding allows for extremely simple program inversion, $\mathrm{rev}(\ulcorner T \urcorner) = \ulcorner T^{-1} \urcorner$. This means that we can use the URTM for reversible *inverse interpretation*: Simply apply string reversal to the program before and after running the URTM. Let R_1 be an RTM that reverses its first argument, $R_1(x, y) = (\mathrm{rev}(x), y)$. We have that

$$[\![R_1 \circ U \circ R_1]\!] (\ulcorner T \urcorner, y) = (\ulcorner T \urcorner, [\![T^{-1}]\!](y)) \ .$$

This completes our presentation of the URTM.

4 Related Work

In the recent work [2] the authors studied reversible Turing machines from a programming language viewpoint, defined RTM-universality, and showed the results summarized in Sect. 2 about the expressiveness and robustness of the

[6] The reliance on the reversibility of the interpreted machine explains why our URTM is not a general UTM: It will get stuck if the interpreted program is not reversible.

1-tape 3-symbol RTMs. These results form the theoretical basis for the URTM developed in this paper.

Morita *et al.* have also studied RTMs [11,10] and other powerful reversible computation models, including cellular automata [9]. In [11] a small universal RTM was proposed, which implements an interpreter for a *cyclic tag system* (a Turing complete formalism.) However, under our strict semantics approach, the proposed machine does not demonstrate universality (or RTM-universality), as the halting configuration encompasses not just the program and output, but also the entire string produced by the tag system along the way, analogous to a Landauer embedding [8]. Of course, the intent with their machine also appears to be rather different from ours. We did not aim for a minimal machine in terms of states and symbols, and allowed ourselves 3 tapes compared to only 1 in [11].

5 Conclusion

The study of reversible computation models complements that of deterministic and non-deterministic models. Despite a long history, the fundamental properties of reversible computation models are still not well-understood. In our approach, where reversibility of a Turing machine implies injectivity of its semantical function, we have that reversible Turing machines (RTMs) are not quite Turing complete, but still expressive enough to be universal for their own class with the natural concept of *RTM-universality* (which allows a universal machine to remember the interpreted program text).

We here showed the first RTM-universal reversible Turing machine (URTM) constructed from first principles. The resulting machine is a very clean and simple design. We did not have to rely on reversibilization techniques, and the URTM never writes any temporary values to tape, except for the constant-sized string encoding the current simulated state. The URTM interprets 1-tape 3-symbol RTMs with a program dependent constant factor slowdown (bounded by the size of the interpreted program). Importantly, there are no other change in the time and space behavior as compared to the interpreted machine. Thus, for individual machines the URTM is effectively complexity preserving, which has not been seen before with reversible Turing machines. In addition, the URTM is also able to function as an *inverse interpreter* for running RTM programs *backwards* by simply reversing the program text, again with no impact on asymptotic complexity. These positive qualities were made possible by explicitly exploiting the promise of reversibility of the interpreted machine in the URTM design.

We conclude that the RTMs can simulate themselves efficiently.

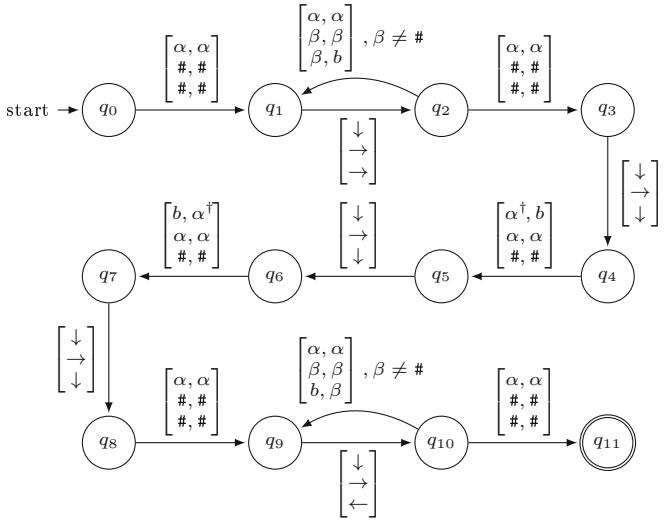
References

1. Axelsen, H.B.: Clean translation of an imperative reversible programming language. In: Knoop, J. (ed.) CC 2011. LNCS, vol. 6601, pp. 144–163. Springer, Heidelberg (2011)
2. Axelsen, H.B.: Glück, R.: What do reversible programs compute? In: Hofmann, M. (ed.) FOSSACS 2011. LNCS, vol. 6604, pp. 42–56. Springer, Heidelberg (2011)

3. Axelsen, H.B., Glück, R., Yokoyama, T.: Reversible machine code and its abstract processor architecture. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) CSR 2007. LNCS, vol. 4649, pp. 56–69. Springer, Heidelberg (2007)
4. Bennett, C.H.: Logical reversibility of computation. IBM Journal of Research and Development 17, 525–532 (1973)
5. Feynman, R.: Quantum mechanical computers. Optics News 11, 11–20 (1985)
6. Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C., Schmitt, A.: Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. ACM Trans. Prog. Lang. Syst. 29(3), Article 17 (2007)
7. Kutrib, M., Malcher, A.: Reversible pushdown automata. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 368–379. Springer, Heidelberg (2010)
8. Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development 5(3), 183–191 (1961)
9. Morita, K.: Reversible computing and cellular automata — A survey. Theoretical Computer Science 395(1), 101–131 (2008)
10. Morita, K., Shirasaki, A., Gono, Y.: A 1-tape 2-symbol reversible Turing machine. Trans. IEICE E 72(3), 223–228 (1989)
11. Morita, K., Yamaguchi, Y.: A universal reversible turing machine. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 90–98. Springer, Heidelberg (2007)
12. Mu, S.C., Hu, Z., Takeichi, M.: An algebraic approach to bi-directional updating. In: Chin, W.N. (ed.) APLAS 2004. LNCS, vol. 3302, pp. 2–20. Springer, Heidelberg (2004)
13. Thomsen, M.K., Axelsen, H.B.: Parallelization of reversible ripple-carry adders. Parallel Processing Letters 19(2), 205–222 (2009)
14. Thomsen, M.K., Glück, R., Axelsen, H.B.: Reversible arithmetic logic unit for quantum arithmetic. Journal of Physics A: Mathematical and Theoretical 42(38), 382002 (2010)
15. van de Snepscheut, J.L.A.: What computing is all about. Springer, Heidelberg (1993)
16. Van Rentergem, Y., De Vos, A.: Optimal design of a reversible full adder. International Journal of Unconventional Computing 1(4), 339–355 (2005)
17. Yokoyama, T., Axelsen, H.B., Glück, R.: Principles of a reversible programming language. In: Proceedings of Computing Frontiers, pp. 43–54. ACM Press, New York (2008)
18. Yokoyama, T., Axelsen, H.B., Glück, R.: Reversible flowchart languages and the structured reversible program theorem. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 258–270. Springer, Heidelberg (2008)

Appendix A

Rule Application



Above is given an RTM state transition diagram for applying rule $\big(q, (s, s'), q'\big)$ (represented on the program tape) given that the current interpreted state q_c (on the state tape) is q, and the current interpreted symbol (on the work tape) is s. States q_0 through q_3 (corresponding to clear $q_c = q$) deletes the current interpreted state q from the state tape; states q_4 through q_7 (corresponding to write $q_c = q'$) substitutes the current interpreted symbol s on the first tape with s'; states q_8 though q_{11} writes the new interpreted state q' on the state tape. $\alpha^\dagger$ in the q_4 to q_5 and q_6 to q_7 transitions is b if $\alpha = \mathtt{B}$ and α otherwise.

The Parameterized Complexity of Chosen Problems for Finite Automata on Trees

Agata Barecka[1] and Witold Charatonik[2]

[1] Institute of Mathematics, University of Wroclaw,
pl. Grunwaldzki 2/4, 50-384 Wroclaw, Poland
[2] Institute of Computer Science, University of Wroclaw,
ul. Joliot-Curie 15, 50-383 Wroclaw, Poland

Abstract. There are many decision problems in automata theory (including membership, emptiness, emptiness of intersection, inclusion and universality problems) that for some classes of tree automata are NP-hard. The study of their parameterized complexity allows us to find new bounds of their non-polynomial time algorithmic behaviors. We present results of such a study for classical tree automata (TA), rigid tree automata (RTA), tree automata with global equality and disequality (TAGED) and t-DAG automata.

1 Introduction

Parameterized complexity. In the classical complexity theory the complexity of a problem is measured by the amount of a resource (time or space) required to solve the problem, which is presented as a function of the size of the input of the problem. There are many problems that are hard in this theory, but appear not to be so hard under a more refined analysis of complexity that takes into account the structure of the input data. A notable example of such a problem is LTL (Linear Temporal Logic) model checking (i.e., the problem if a given Kripke structure satisfies a given formula of the logic LTL) in the area of automated verification [11]. The problem is hard in classical complexity theory, where we consider input as a whole, but it becomes tractable if we look into the structure of the input and assume that the "hard" part (here, the checked formula) is relatively small in comparison to the "easy" part (the checked Kripke structure).

Parameterized complexity [7] [9] gives a framework for analysis of such problems. In this theory, an instance x of a problem comes together with a parameter k (typically, the size of some part of the input x). If a problem is decidable in time $f(k)p(|x|)$ for some function f and some polynomial p, then the problem is considered to be tractable (with respect to the parameter k) and is called *fixed-parameter tractable*. There are two fundamental hierarchies of problems that are not known to be fixed-parameter tractable: the W-hierarchy and the A-hierarchy. It is believed (but there is no proof of that, similarly as there is no proof of P$\neq$NP) that they do not collapse, so if one proves that a problem is hard for some levels of these hierarchies, then the existence of efficient algorithms for this problem is unlikely.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 129–141, 2011.

Tree automata. The theory of finite tree automata [6] is a straightforward extension of the theory of finite word automata. The main task of this theory is to provide a finite representation for infinite sets of terms, with efficient operations for manipulating these sets, and with decidable basic decision problems. In this paper we investigate parameterized complexity of NP-hard problems for tree automata. In addition to classical tree automata we also consider recent extensions like rigid tree automata (RTA), automata with global constraints (TAGED) and automata on DAG representations of trees (t-DAG). It was known that these extensions gain additional expressive power for the price of harder (in many cases NP-hard) decision problems.

Our contribution. Our results are summarized in Table 2 at the end of the paper. In particular, we show that for classical tree automata the inclusion problem and the universality problem parameterized by the number of states are para-co-NP-hard. For RTA and TAGED automata the membership problem parameterized by the number of states is W[2]-hard; parameterized by the size of the input term and the size of the signature it is in W[1]. We also show that for t-DAG automata the membership problem parameterized by the size of a t-dag and the size of a signature is W[1]-complete and parameterized by the number of states of the input automaton it is para-NP-complete while the k-emptiness problem parameterized by k and the size of a signature is W[1]-complete. A consequence of all these hardness (which includes completeness) results is that these problems are not fixed-parameter tractable (unless the W-hierarchy is not strict). This is rather a bad news for the theory of tree automata, and it was quite surprising for us — we expected at least some of these problems to be fixed-parameter tractable.

Related work. We are not aware of any work on parameterized complexity of decision problems for tree automata. The closest results [12] concern automata on words. It is shown there that the $k-\cap$-EMPTINESS problem (see Section 3.3 for the definition of the problem) parameterized by either $|\Sigma| + k$, $|A| + |Q|$ or $|\Sigma|+|Q|$ is fixed-parameter tractable while parameterized by $|A|+k$ is W[1]-hard and parameterized by $|Q| + k$ is W[2]-hard. From the proofs it is not difficult to infer that $\cap$-EMPTINESS parameterized by the number of intersected automata is W[1]-hard.

2 The Parameterized Complexity Theory

Below we recall the most important concepts from the parameterized complexity theory that are used in this paper ([7],[9]).

Definition 1. *A **parameterized problem** over an alphabet Σ is a pair (A, κ) consisting of a set $A \subset \Sigma^*$ and a function $\kappa : \Sigma^* \to \mathbb{N}$. The function κ is called a **parameterization** of the problem.*

Definition 2. *Let (A, κ_A) and (B, κ_B) be parameterized problems over Σ and Γ respectively. An **FPT-reduction** of (A, κ_A) to (B, κ_B) is a mapping $F : \Sigma^* \to \Gamma^*$*

such that for every input $x \in \Sigma^*$ *we have* $x \in A \Leftrightarrow F(x) \in B$*, there exist a function* h *and a polynomial* p *such that for every* $x \in \Sigma^*$ *the result* $F(x)$ *is computable in time* $h(\kappa_A(x))p(|x|)$ *and there exists a function* g *such that for every* $x \in \Sigma^*$ *we have* $\kappa_B(F(x)) \leqslant g(\kappa_A(x))$.

We say that a parameterized problem (A, κ) is *fixed parameter tractable* or that it is in the class FPT (respectively, it is in the class *para-NP*) if there exists a deterministic (respectively, nondeterministic) algorithm that for all $x \in \Sigma^*$ decides whether $x \in A$ in time $f(\kappa(x))p(|x|)$, where f is a computable function and p is a polynomial. A parameterized problem (A, κ) is in the class *para-co-NP* class if its complement $(\Sigma^* \backslash A, \kappa)$ is in the para-NP class. W[P] is the class of parameterized problems (A, κ) such that there exists a nondeterministic algorithm that for all $x \in \Sigma^*$ decides whether $x \in A$ in time $f(\kappa(x))p(|x|)$ and uses $g(\kappa(x)) \log |x|$ nondeterministic steps where f, g are computable functions and p is a polynomial.

The W-hierarchy is a fundamental hierarchy of problems that are not known to be fixed-parameter tractable. For the purpose of the current paper it is not important how exactly the levels of this hierarchy are defined; it is more important that the inclusions

$$\text{FPT} \subseteq \text{W[1]} \subseteq \text{W[2]} \subseteq \ldots \text{W[P]} \subseteq \text{para-NP}.$$

are known to be true (and despite a lot of research already done, they are not known to be strict or to collapse). The following problems are known to be complete for the respective levels of these hierarchies (see [9] for proofs). To simplify the notation, we write values of parameter functions next to inputs of problems instead of treating parameters as separate functions.

The problem of parameterized model checking for Σ_1-formulas and the parameterized clique problem are W[1]-complete. A clique for a graph $G = (V, E)$ is a set $X \subseteq V$ such that for every $v, v' \in X$ there is an edge $\{v, v'\} \in E$.

Problem (p-MC(Σ_1)). Instance: A structure $\mathscr{A}$ and a first-order formula ϕ with quantifier prefix $\exists^*$. *Parameter*: $|\phi|$. *Question*: $\mathscr{A} \models \phi$?

Problem (p-CLIQUE). Instance: A graph G and $k \in \mathbb{N}$. *Parameter*: k. *Question*: Is there a clique of k elements in G ?

The problem p-DOMINATING-SET is W[2]-complete. A dominating set for a graph $G = (V, E)$ is a set $X \subset V$ such that for every $v \in V$ either $v \in X$ or there exists $v' \in X$ such that $\{v, v'\} \in E$.

Problem (p-DOMINATING-SET). Instance: A graph G and $k \in \mathbb{N}$. *Parameter*: k. *Question*: Is there a dominating set of k elements in G ?

The parameterized colorability problem is para-NP-complete. Its dual is para-co-NP-complete. We say that a graph $G = (V, E)$ is k colorable if there exists a function $\alpha : V \to \{1, \ldots, k\}$ such that for every $v, v' \in V$ if $\alpha(v) = \alpha(v')$ then $\{v, v'\} \notin E$.

Problem (p-COLORABILITY). Instance: A graph G and $k \in \mathbb{N}$. *Parameter*: k. *Question*: Is G k-colorable?

Problem (p-NON-COLORABILITY). Instance: A graph G and $k \in \mathbb{N}$. *Parameter*: k. *Question*: Is G a non-k-colorable graph?

3 Finite Automata

3.1 Preliminaries

Before discussing different types of finite automata on trees we introduce a basic terminology connected with this subject. This is standard terminology when dealing with tree automata [6].

A *signature* Σ is a finite set of function symbols with their arity. A *term* over a signature Σ is either a constant symbol from Σ or has the form $f(t_1, t_2, \ldots, t_n)$, where $f \in \Sigma$ is an n-ary function symbol and $t_1, t_2, \ldots, t_n$ are terms. We identify terms with their tree representations, so we use both notions of a tree and a term interchangeably. A *size* of a term t (in this paper denoted by $|t|$) is a number of subterms of this term (or, equivalently, a number of vertices in its tree presentation). A *language* over a signature Σ is a set (not necessarily finite) of terms over Σ. The positions $Pos(t)$ in a term t are sequences of positive integers (ϵ, the empty sequence, is the root position; generally a number sequence represents a node as the path of argument-order edges followed from the root to get to that node). Hence t can be seen as a function from its set of positions $Pos(t)$ into a set of function symbols Σ. By $t|_p$, where t is a term and $p \in Pos(t)$, we denote a subterm of t at the position p. A *DAG representation of a term* (in short, a *t-dag*) over a signature Σ is a directed, acyclic, ordered graph with vertices labeled with symbols from Σ such that if a vertex is labeled with an n-ary symbol then it has n immediate successors. Moreover, it cannot contain two different vertices representing the same subterm. The *size* of a t-dag is the number of its vertices.

3.2 Classes of Automata

In this section we present the models of automata we are interested in. All considered automata are nondeterministic.

Classical Tree Automata (TA). This class of automata is described with details in [6].

Definition 3. *A **tree automaton (TA)** is a 4-tuple $\langle \Sigma, Q, F, \delta \rangle$, where Σ is a finite signature, Q is a finite set of states, $F \subseteq Q$ is a set of final states and δ is a set of transition rules of the form $f(q_1, q_2 \ldots, q_n) \to q$ with $q, q_1, q_2 \ldots, q_n \in Q$ and $f \in \Sigma$ of arity n.*

An automaton starts a computation at the leaves of a tree and moves upward to the root associating inductively states with subtrees in such a way that its transition rules are fulfilled. The *size* of a tree automaton $\mathscr{A} = \langle \Sigma, Q, F, \delta \rangle$ is equal to $|Q| + |\Sigma| + |\delta|$ and is denoted by $|\mathscr{A}|$.

Definition 4. *A **run of a TA automaton** $\langle \Sigma, Q, F, \delta \rangle$ on a term t is a mapping $r : Pos(t) \to Q$ such that for every position $p \in Pos(t)$, if $t(p) = f$ where f is a n-ary symbol in $\Sigma, r(p) = q$ and $r(pi) = q_i$ for all $i \in \{1, \ldots, n\}$ then the transition $f(q_1, q_2, \ldots, q_n) \to q$ belongs to δ. A run is successful if it maps the root of t to a final state.*

An automaton $\mathscr{A}$ *accepts* a tree t if there exists a successful run of $\mathscr{A}$ on t. The set of all trees accepted by $\mathscr{A}$ is called the *language* of $\mathscr{A}$ and is denoted $\mathscr{L}(\mathscr{A})$.

Tree Automata with Global Equality and Disequality Constraints (TAGED). The TAGED class is described with details in [8].

Definition 5. *A **tree automaton with global constraints (TAGED)** is a 6-tuple $\langle \Sigma, Q, R_=, R_{\neq}, F, \delta \rangle$, where $\langle \Sigma, Q, F, \delta \rangle$ is a tree automaton and $R_=, R_{\neq}$ are binary relations on Q.*

Definition 6. *A **run of a TAGED automaton** $\langle \Sigma, Q, R_=, R_{\neq}, F, \delta \rangle$ on a term t is a mapping r that is a run of TA $\langle \Sigma, Q, F, \delta \rangle$ on t satisfying additional conditions for all $p_1, p_2 \in Pos(t)$:*

$$(r(p_1), r(p_2)) \in R_= \;\Rightarrow\; t|_{p_1} = t|_{p_2} \quad \text{and} \quad (r(p_1), r(p_2)) \in R_{\neq} \;\Rightarrow\; t|_{p_1} \neq t|_{p_2}.$$

Rigid Tree Automata (RTA). The RTA class is presented in [10]. It is a restriction of the TAGET class where some states are identified as rigid.

Definition 7. *A **rigid tree automaton (RTA)** is a 5-tuple $\langle \Sigma, Q, R, F, \delta \rangle$, where $\langle \Sigma, Q, F, \delta \rangle$ is a tree automaton and $R \subset Q$ is a set of rigid states.*

During a computation of an RTA automaton all subtrees associated with one rigid state must be equal.

Definition 8. *A **run of an RTA automaton** $\langle \Sigma, Q, R, F, \delta \rangle$ on a term t is a mapping r that is a run of TA $\langle \Sigma, Q, F, \delta \rangle$ on t satisfying for all $p_1, p_2 \in Pos(t)$ an additional condition: $r(p_1) = r(p_2) \in R \;\Rightarrow\; t|_{p_1} = t|_{p_2}$.*

Automata on DAG Representations of Trees (t-DAG Automata). The t-DAG class is introduced in [4] and used in [5] for solving set constraints. It is a class of automata running on dag representations of terms. In other words, a t-DAG automaton is a tree automaton that with equal subtrees of a tree associates equal states. Using the notation from Definition 8 it means that $t|_{p_1} = t|_{p_2} \Rightarrow r(p_1) = r(p_2)$. In this sense it is an automata class dual to the RTA class.

Definition 9. *A **t-DAG automaton** is a 4-tuple $\langle \Sigma, Q, F, \delta \rangle$, where Σ is a finite signature, Q is a finite set of states, $F \subset Q$ is a set of final states and δ is a set of transitions of a form $f(q_1, q_2 \ldots, q_n) \to q$ with $q, q_1, q_2 \ldots, q_n \in Q$ and $f \in \Sigma$ of arity n.*

Definition 10. *A **run of a t-DAG automaton** $\langle \Sigma, Q, F, \delta \rangle$ on a t-dag G is a mapping r from the set of nodes of G to the set Q such that for every node v of G labeled with n-ary symbol $f \in \Sigma$, if $v_1, v_2, \ldots, v_n$ are successors of v, then $f(r(v_1), r(v_2), \ldots, r(v_n)) \to r(v)$ belongs to δ. A run is successful if it maps the root of G to a final state.*

3.3 Parameterized Decision Problems for Finite Automata on Trees

We introduce some parameterized decision problems for finite automata on trees. By Q we denote the set of states of an automaton and by Σ its signature. By k we denote a unary encoded number. For each problem different parameterizations can be considered.

Problem (EMPTINESS). Instance: An automaton $\mathscr{A}$. *Parameter*: $|Q|, |\Sigma|$ or a sum of them. *Question*: Is the language recognized by $\mathscr{A}$ empty?

Problem (k-EMPTINESS). Instance: An automaton $\mathscr{A}$ and $k \in \mathbb{N}$. *Parameter*: $|Q|, |\Sigma|$, k or a sum of some of them. *Question*: Does there exist a tree/t-dag of size k accepted by $\mathscr{A}$?

In the following problems all automata in A are defined over the same signature Σ. By $|Q|$ we denote the maximal number of states of an automaton from A.

Problem ($\cap$-EMPTINESS). Instance: A set of automata $A = \{\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_n\}$. *Parameter*: $|Q|, |\Sigma|, |A|$ or a sum of some of them. *Question*: Does there exist a tree/t-dag accepted by all automata from A?

Problem (k-$\cap$-EMPTINESS). Instance: A set of automata $A = \{\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_n\}$ and $k \in \mathbb{N}$. *Parameter*: $|Q|, |\Sigma|, |A|, k$ or a sum of some of them. *Question*: Does there exist a tree/t-dag of size k accepted by all automata from A?

Problem (MEMBERSHIP). Instance: An automaton $\mathscr{A}$ and a word/tree/t-dag t. *Parameter*: $|Q|, |\Sigma|, |t|$ or a sum of some of them. *Question*: Is t accepted by $\mathscr{A}$?

Problem (UNIVERSALITY). Instance: An automaton $\mathscr{A}$. *Parameter*: $|Q|, |\Sigma|$ or a sum of them. *Question*: Is the language recognized by $\mathscr{A}$ total?

Problem (INCLUSION). Instance: Two automata $\mathscr{A}_1$ and $\mathscr{A}_2$ defined over the same signature Σ. *Parameter*: $|Q_1|, |Q_2|, |\Sigma|$ or a sum of some of them. *Question*: Is it true that $\mathscr{L}(\mathscr{A}_1) \subseteq \mathscr{L}(\mathscr{A}_2)$?

3.4 Known Results

The table below presents known results in the area of the classical complexity of described decision problems. For proofs see [1], [3],[4],[6],[8],[10].

Table 1. Summary of known results

	TA	RTA	TAGED	t-DAG
EMPTINESS	PTIME	PTIME	EXPTIME-hard decidable	NP-complete
$\cap$-EMPTINESS	EXPTIME-compl	EXPTIME-compl	EXPTIME-hard	
MEMBERSHIP	PTIME	NP-complete	NP-complete	NP-complete
UNIVERSALITY	EXPTIME-compl	undecidable	undecidable	undecidable
INCLUSION	EXPTIME-compl	undecidable	undecidable	undecidable

4 Results

Most of the theorems from this section are proved by FPT-reductions. A definition of FPT-reduction can be found in Section 2 (see Definition 2). Due to space limits some proofs are sketched or omitted. Complete proofs can be found in [2].

4.1 Classical Tree Automata (TA)

***k*-EMPTINESS, ∩-EMPTINESS and *k*-∩-EMPTINESS.** Our results on emptiness problems for classical tree automata are summarized in 2. As they are simple observations we omit the proofs (the proofs can be found in [2]).

UNIVERSALITY

Theorem 1. *For TA automata the UNIVERSALITY problem parameterized by the number of states of an automaton is para-co-NP-hard.*

Sketch of proof. We propose a reduction of p-NON-COLORABILITY, which is a para-co-NP-complete problem. Consider an instance of the p-NON-COLORABILITY problem: a graph $G = (V, E)$ of n vertices $v_1, v_2, \ldots, v_n$ and a number $k \in \mathbb{N}$. We construct a TA automaton $\mathscr{A} = \langle \Sigma, Q, F, \delta \rangle$ such that G is not k-colorable if and only if $\mathscr{A}$ accepts all terms over the signature Σ. Let $\Sigma = \{a_1, a_2, \ldots, a_k, f\}$, where a_i is a constant symbol and f is an n-ary function symbol.

A term over Σ of the form $f(x_1, x_2, \ldots, x_n)$ for $x_i \in \{a_1, a_2, \ldots, a_k\}$ represents a coloring of G with a set of k-colors ($x_i = a_j$ means that the vertex v_i has the color j). Let $Q = \{q_1, q_2, \ldots, q_k, q, q', q_F\}$ and $F = \{q_F\}$. Let δ consist of the following transitions:

(i) $a_i \mapsto q \,|\, q_i \,|\, q_F$ for $i \in \{1, 2, \ldots, k\}$,
(ii) $f(p_1, p_2, \ldots, p_n) \mapsto q \,|\, q' \,|\, q_F$ if there exists an edge $\{v_j, v_l\} \in E$ such that $p_j = p_l = q_i$ for some i and $p_s = q$ for all $s \neq j, s \neq l$,
(iii) $f(p_1, p_2, \ldots, p_n) \mapsto q \,|\, q' \,|\, q_F$ if $p_j = q'$ for some j and $p_l = q$ for all $l \neq j$.

The automaton $\mathscr{A}$ accepts a term t if and only if it does not contain a subterm representing a good coloring. This fact can be proved by the induction on the structure of a term. Thus, $\mathscr{A}$ accepts all terms over the signature Σ if and only if there are no good k-colorings of G i.e. G is not k-colorable. Moreover, $|Q| = k+3$ and the size of $\mathscr{A}$ depends polynomially on the number k and the size of G. □

The result above shows that universality is a very hard problem. Even if we fix the number of states in the input automaton, it remains co-NP-hard. In fact, since 3-colorability is an NP-complete problem, we can observe that already universality of automata with 6 states is co-NP-hard. If one identifies (which is quite common) the number of states of an automaton with its size, this becomes a surprising result, because for automata of fixed size the problem should be solvable in constant time.

INCLUSION. The following result is a simple observation. We omit the proof (it can be found in [2]).

Proposition 1. *For TA automata INCLUSION problem parameterized by the size of $\mathscr{A}_2$ is in FPT.*

Theorem 2. *For TA automata INCLUSION problem parameterized by $|Q_1| + |Q_2|$ is para-co-NP-hard.*

Proof. There is an automaton of one state that accepts all terms over a given signature. The theorem is a consequence of this fact and Theorem 1. □

4.2 Rigid Tree Automata (RTA) and Tree Automata with Equalities and Disequalities (TAGED)

***k*-EMPTINESS**

Theorem 3. *For RTA and TAGED automata the k-EMPTINESS problem parameterized by $k + |\Sigma|$ is in $W[1]$.*

Proof. It is enough to show it for TAGED automata. We reduce the problem to p-MC(Σ_1). Consider a TAGED automaton $\langle \Sigma, Q, R_=, R_{\neq}, F, \delta \rangle$ and a number $k \in \mathbb{N}$. We construct a structure $\mathscr{A}$ and a formula ϕ such that the formula ϕ is fulfilled in $\mathscr{A}$ if and only if the automaton $\langle \Sigma, Q, R_=, R_{\neq}, F, \delta \rangle$ accepts some term of size k. Let $\mathscr{A}$ be defined over the set $A = \Theta \cup Q \cup \Sigma \cup \{\bot\}$, where Θ is the set of terms over Σ of size not greater than k. Let $M = \max\{\, \mathrm{ar}(f) \mid f \in \Sigma \,\}$. Let there be the following relations in $\mathscr{A}$.

$$D := \big\{(q_0, f, q_1, \ldots, q_M) \,|\, f(q_1, \ldots, q_{\mathrm{ar}(f)}) \mapsto q_0 \in \delta \text{ and } q_i = \bot \text{ for } \mathrm{ar}(f) < i \leqslant M\big\},$$
$$T := \big\{(t_0, f, t_1, \ldots, t_M) \mid\ f(t_1, \ldots, t_{\mathrm{ar}(f)}) = t_0 \text{ and } t_i = \bot \text{ for } \mathrm{ar}(f) < i \leqslant M\big\},$$
$$T_= := \big\{(t, t) \mid\ t \in \Theta\big\}, \quad T_{\neq} := \Theta \times \Theta \setminus T_= \quad R_{A=} := R_=,$$
$$R_{A\neq} := R_{\neq} \quad \text{and} \quad F_A := F.$$

Now we define the formula $\phi = \exists x_1, \ldots, x_k, y_1, \ldots, y_k\, \phi'$ where ϕ' is the conjunction

$$\bigwedge_{f \in \Sigma} \bigwedge_{1 \leq i_0, \ldots, i_{\mathrm{ar}f} \leq k} \Big(T(y_{i_0}, f, y_{i_1}, \ldots, y_{i_{\mathrm{ar}f}}, \bot, \ldots, \bot) \Rightarrow D(x_{i_0}, f, x_{i_1}, \ldots, x_{i_{\mathrm{ar}f}}, \bot, \ldots, \bot) \Big)$$
$$\wedge \bigwedge_{1 \leq i_1, i_2 \leq n} \Big(R_{A=}(x_{i_1}, x_{i_2}) \Rightarrow T_=(y_{i_1}, y_{i_2}) \Big) \wedge \Big(R_{A\neq}(x_{i_1}, x_{i_2}) \Rightarrow T_{\neq}(y_{i_1}, y_{i_2}) \Big)$$
$$\wedge F_A(x_1) \wedge \psi(y_1, \ldots, y_k),$$

where $\psi(y_1, \ldots, y_k)$ is fulfilled if terms $y_1, \ldots, y_k$ form a tree and are written from the top to the bottom and from the left to the right.

$$\psi(y_1, \ldots, y_k) := \bigvee_{t \in \Theta_k} \bigwedge_{1 \leq i \leq k} T(y_i, f_t(i), y_{c_t(i)+1}, \ldots, y_{c_t(i)+\mathrm{ar}f}, \bot, \ldots, \bot),$$

where Θ_k is the set of all terms from Θ that have the size k, $c_t(i)$ is the index of the first child of the vertex of the index i in the tree t (we are numbering from the top to the bottom and from the left to the right) and $f_t(i)$ is the function symbol labeling that vertex. The length of ψ depends only on k and $|\Sigma|$. The size of the structure $\mathscr{A}$ depends on k and $|\Sigma|$ and polynomially on the size of the automaton. The length of ϕ depends only on k and $|\Sigma|$. □

MEMBERSHIP

Theorem 4. *For RTA and TAGED automata the MEMBERSHIP problem parameterized by $|Q|$ is W[2]-hard.*

Sketch of proof. It is enough to show it for RTA automata. We propose a reduction of p-DOMINATING-SET. Consider an instance of this problem: a graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ and a number $k \in \mathbb{N}$. We construct an RTA automaton $\mathscr{A} = (\Sigma, Q, R, F, \delta)$ and a term t over the signature Σ such that in G there is a dominating set of k elements if and only if $\mathscr{A}$ accepts the term t.

Let $\Sigma = \{x_v | \, v \in V\} \cup \{f_{v_1,v_2} | \, v_1, v_2 \in V\} \cup \{f, g\}$, where x_v are constant symbols, f_{v_1,v_2} are symbols of binary functions and f and g are symbols of n-ary functions. Let $Q = \{q_1, \ldots, q_k, q, q_0, q_F\}$, $R = \{q_1, \ldots, q_k\}$, $F = \{q_F\}$. States q_i for $i \in \{1, \ldots, k\}$ represent a choice of vertices of a dominating set. Let δ consist of the following transitions.

(i) $x_v \mapsto q_i$ for $v \in V$ and $i \in \{0, 1, \ldots, k\}$,
(ii) $f_{v,w}(q_i, q_j) \mapsto q_0$ for $v, w \in V$ and $i, j \in \{0, 1, \ldots, k\}$,
(iii) $f_{v,w}(q_0, q_i) \mapsto q$ for $v, w \in V$ such that $\{v, w\} \in E$ and $i \in \{1, 2, \ldots, k\}$,
(iv) $f_{v,w}(q_i, q_j) \mapsto q$ for $v, w \in V$, $i \in \{1, 2, \ldots, k\}$ and $j \in \{0, 1, \ldots, k\}$,
(v) $g(p_1, p_2, \ldots, p_n) \mapsto q$, if there exists $i \in \{1, 2, \ldots, n\}$ such that $p_i = q$ and $p_j = q_0$ for all $j \neq i$,
(vi) $f(q, q, \ldots, q) \mapsto q_F$.

Let t be the term:

$$f\Big(\, g\big(f_{v_1,v_1}(x_{v_1}, x_{v_1}), \ldots, f_{v_1,v_n}(x_{v_1}, x_{v_n})\big), \ \ldots, \\ g\big(f_{v_n,v_1}(x_{v_n}, x_{v_1}), \ldots, f_{v_n,v_n}(x_{v_n}, x_{v_n})\big) \, \Big).$$

The size of the automaton $\mathscr{A}$ and the length of the term t depend polynomially on the size of the graph G. Indeed, there are $n(k+1)$ transitions of the form (i), $n^2(k+1)^2$ transitions of the form (ii), n^2k transitions of the form (iii), $n^2k(k+1)$ transitions of the form (iv), n transitions of the form (v) and one transition of the form (vi). Moreover, $|Q| = k+3$. One can check that G has a dominating set of k elements if and only if $\mathscr{A}$ accepts the term t. Indeed, let r be a successful run of $\mathscr{A}$ on t. Let $X = \{v \in V \mid \exists l \in t_v \ r(l) \neq q_0\}$, where t_v is the set of all vertices of t labeled by x_v. The rigidity of states q_i for $i > 0$ implies that $|X| \leqslant k$. The run r is successful, so $r\Big(g\big(f_{v_i,v_1}(x_{v_i}, x_{v_1}), \ldots, f_{v_i,v_n}(x_{v_i}, x_{v_n})\big)\Big) = q$ for every

$i \in \{1, 2, \ldots, n\}$. Thus for exactly one $j \in \{1, ..., n\}$ we have $r(f_{v_i,v_j}(x_{v_i}, x_{v_j})) = q$. Then either $r(x_{v_i}) \neq q_0$ and $v_i \in X$ or $r(x_{v_j}) \neq q_0, v_j \in X$ and $\{v_i, v_j\} \in E$. So v_i is dominated by v_j. Hence, the set X is a dominating set in G. A proof of the second implication is omitted. □

By using similar methods as in Theorem 3 we can solve the membership problem. The main difficulty is to make the length of the formula ϕ depend only on the size of the term t (the input of the membership problem) and not on $|\Sigma|$. This can be done by using only those symbols from Σ that appear in t.

Theorem 5. *For RTA and TAGED automata the MEMBERSHIP problem parameterized by the length of a term belongs to* $W[1]$.

4.3 Automata on DAG Representations of Finite Trees (t-DAG Automata)

EMPTINESS

Proposition 2. *For t-DAG automata the EMPTINESS problem parameterized by* $|Q|$ *belongs to* $W[P]$. *The same problem parameterized by* $|Q| + |\Sigma|$ *belongs to FPT.*

Proof. In [4] it was proved that if a language recognized by a t-DAG automaton $\mathscr{A}$ is nonempty then $\mathscr{A}$ accepts a t-dag of size not greater than $2|Q|^3$ where Q is the set of states of $\mathscr{A}$. To check the EMPTINESS for $\mathscr{A}$ it is enough to guess a t-dag t of a size not greater than $2|Q|^3$ and a mapping r from the set of subterms of t to Q and check if r is an accepting run. The algorithm guesses $2\,|Q|^3 \log(\,|\Sigma|\,)$ bits; moreover, there are $O(|Q|^{2|Q|^3})$ possible runs on a term of size not greater than $2|Q|^3$.

In the case of parameterization with $|Q| + |\Sigma|$, it is enough to check, for every t-dag of size not greater than $2|Q|^3$, if it is accepted by $\mathscr{A}$. One can do this by analyzing all possible runs of the automaton on this term. There are $O(|Q|^{6|\Sigma|})$ such terms and there are $O(|Q|^{2|Q|^3})$ possible runs on each of them. □

k-EMPTINESS

Theorem 6. *For t-DAG automata the k-EMPTINESS problem parameterized by* $k + |\Sigma|$ *is* $W[1]$*-complete.*

Sketch of proof. We propose a reduction of p-CLIQUE which is a W[1]-complete problem. Consider an instance of the p-CLIQUE problem: a graph $G = (V, E)$ and a number $k \in \mathbb{N}$. We construct a t-DAG automaton $\mathscr{A} = (\Sigma, Q, F, \delta)$ and a number k' such that in G there is a clique of k elements if and only if $\mathscr{A}$ accepts a tree of the size k'.

Let $\Sigma = \{x_1, x_2, \ldots, x_k\} \cup \{f, g\}$, where x_i is a constant symbol, f is a symbol of a $k(k-1)$-ary function and g is a symbol of a binary function. Let $Q = \{q_{1,2}, q_{1,3}, \ldots, q_{k,k-1}, q_F\} \cup \{q_{v,i} | \, v \in V,\, 1 \leq i \leq k\}$, $F = \{q_F\}$. Let δ consist of the following transitions:

(i) $x_i \mapsto q_{v,i}$ for $v \in V$ and $1 \leq i \leq k$,
(ii) $g(q_{v_1,i}, q_{v_2,j}) \mapsto q_{i,j}$ for $v_1, v_2 \in V$ such that $\{v_1, v_2\} \in E$,
(iii) $f(q_{1,2}, q_{1,3}, \ldots, q_{k,k-1}) \mapsto q_F$.

Note that $t = f(g(x_1, x_2), g(x_1, x_3), \ldots, g(x_{k-1}, x_k))$ is the only term that can be accepted by $\mathscr{A}$.

Let $k' = |t| = 1 + k(k-1) + 2k(k-1) = 3k(k-1) + 1$. The size of the automaton $\mathscr{A}$ depends polynomially on the size of the graph G. Moreover, the parameter k' depends polynomially on the parameter k. What is more, one can easily show that in G there is a clique of k elements if and only if $\mathscr{A}$ accepts the term t.

The proof of the fact that the examined problem is in W[1] is similar to the proof of Theorem 3. □

MEMBERSHIP. By using similar methods as in Theorem 3 (for the membership in W[1]) and in Theorem 6 (for W[1]-hardness) one can prove the following theorem.

Theorem 7. *For t-DAG automata the MEMBERSHIP problem parameterized by $|t| + |\Sigma|$ is W[1]-complete.*

Theorem 8. *For t-DAG automata the MEMBERSHIP problem parameterized by a number of states is para-NP-complete.*

Sketch of proof. In order to show para-NP-hardness of the problem we propose a reduction of p-COLORABILITY. Consider an instance of the this problem: a graph $G = (V, E)$ with n vertices $v_1, v_2, \ldots, v_n$ and m edges $e_1, e_2, \ldots, e_m$ and a number $k \in \mathbb{N}$. We construct a t-DAG automaton $\mathscr{A} = \langle \Sigma, Q, F, \delta \rangle$ and a t-dag t such that G is k-colorable if and only if $\mathscr{A}$ accepts t. Let $\Sigma = \{v_1, v_2, \ldots, v_n, e_1, e_2, \ldots, e_m, f\}$, where v_i is a constant symbol, e_i is a symbol of a binary function and f is a n-ary function symbol. Let $Q = \{q_1, q_2, \ldots, q_k, q, q_F\}$ and $F = \{q_F\}$. States $q_1, q_2, \ldots, q_k$ symbolize k colors. Let δ consist of the following transitions.

(i) $v \mapsto q_i$ for $i \in \{1, 2, \ldots, k\}$,
(ii) $e_l(q_i, q_j) \mapsto q$ for $1 \leq l \leq m$ and $1 \leq i \neq j \leq k$,
(iii) $f(q, q, \ldots, q) \mapsto q_F$.

Let $t = f(e_1(v_{11}, v_{12}), e_2(v_{21}, v_{22}), \ldots, e_m(v_{m1}, v_{m2}))$, where v_{i1} and v_{i2} are vertices incident to an edge e_i. The size of the automaton $\mathscr{A}$ and the size of the t-dag t depend polynomially on k and on the size of the graph G. Moreover, $|Q| = k+2$. One can easily show that G is k-colorable if and only if $\mathscr{A}$ accepts t.

The fact that examined problem belongs to the para-NP class is quite obvious. Indeed, it is solvable by the algorithm that for each vertex of t guesses a state from Q and then checks in polynomial time if this mapping is an accepting run of $\mathscr{A}$ on t. □

Table 2. Summary of our results

problem	parameter	TA	RTA	TAGED	t-DAG
EMPTINESS	$\|Q\| + \|\Sigma\|$	PTIME	PTIME		FPT
	$\|Q\|$				W[P]
k-EMPTINESS	$k + \|\Sigma\|$	FPT	W[1]	W[1]	W[1]-complete
$\cap$-EMPTINESS	$\|A\| + \|Q\|$	FPT			
	$\|A\|$	W[1]-hard	W[1]-hard	W[1]-hard	W[1]-hard
	$\|Q\|$	W[2]-hard	W[2]-hard	W[2]-hard	W[2]-hard
k-$\cap$-EMPTINESS	$\|\Sigma\| + \|Q\| + k$	FPT			
	$\|A\| + \|Q\|$	FPT			
	$\|A\| + k$	W[1]-hard	W[1]-hard	W[1]-hard	W[1]-hard
	$\|Q\| + k$	W[2]-hard	W[2]-hard	W[2]-hard	W[2]-hard
MEMBERSHIP	$\|t\| + \|\Sigma\|$	PTIME	W[1]	W[1]	W[1]-complete
	$\|t\|$		W[1]	W[1]	W[1]-complete
	$\|Q\|$		W[2]-hard	W[2]-hard	para-NP-compl
UNIVERSALITY	$\|Q\|$	para-co-NP-compl	undecidable	undecidable	undecidable
INCLUSION	$\|\mathscr{A}_2\|$	FPT	undecidable	undecidable	undecidable
	$\|Q_1\| + \|Q_2\|$	para-co-NP-compl			

5 Conclusion

We have studied parameterized complexity of several decision problems for the following classes of automata on finite trees: TA automata, RTA automata, TAGED automata and t-DAG automata. Results of our studies are presented in Table 2 (some of values "PTIME" and "undecidable" are rewritten from already presented Table 1).

These results were quite surprising for us — we had expected more of these problems to be fixed-parameter tractable. However, a lot of the examined problems turn to be hard even for such a big parameter as the number of states of an automaton.

As one can see there are still gaps in the presented table. Moreover, some issues are partially examined as we have only proved that they belong to some complexity class or that they are hard in that class. In addition, it seems interesting to check how the complexity will change if one uses binary (instead of unary) encoding of a number k in k-EMPTINESS and k-$\cap$-EMPTINESS problems. In consequence, there are left some open questions that we hope to examine in future.

References

1. Anantharaman, S., Narendran, P., Rusinowitch, M.: Closure properties and decision problems of dag automata. Inf. Process. Lett. 94(5), 231–240 (2005)
2. Barecka, A., Charatonik, W.: The parameterized complexity of chosen problems for finite automata on trees (2010), http://www.math.uni.wroc.pl/~barecka/papers/param_aut.pdf

3. Barguñó, L., Creus, C., Godoy, G., Jacquemard, F., Vacher, C.: The emptiness problem for tree automata with global constraints. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LICS 2010), pp. 263–272. IEEE Computer Society Press, Edinburgh (2010)
4. Charatonik, W.: Automata on DAG representations of finite trees. Technical Report MPI-I-1999-2-001, Max-Planck-Institut für Informatik (1999)
5. Charatonik, W., Pacholski, L.: Set constraints with projections. Journal of the ACM 57(4), 1–37 (2010)
6. Comon-Lundh, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (November 2007), http://www.grappa.univ-lille3.fr/tata/
7. Downey, R.G., Fellows, M.: Parameterized complexity. Monographs in Computer Science. Springer, New York (1999)
8. Filiot, E., Talbot, J.M., Tison, S.: Tree automata with global constraints. Int. J. Found. Comput. Sci. 21(4), 571–596 (2010)
9. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag New York, Inc., Secaucus (2006)
10. Jacquemard, F., Klay, F., Vacher, C.: Rigid tree automata. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 446–457. Springer, Heidelberg (2009)
11. Lichtenstein, O., Pnueli, A.: Checking that finite state concurrent programs satisfy their linear specification. In: POPL, pp. 97–107 (1985)
12. Wareham, T.: The parameterized complexity of intersection and composition operations on sets of finite-state automata. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 302–310. Springer, Heidelberg (2001)

Recognizing Shuffled Languages

Martin Berglund, Henrik Björklund, and Johanna Högberg

Department of Computing Science
Umeå University, Sweden
{mbe,henrikb,johanna}@cs.umu.se

Abstract. Language models that use interleaving, or shuffle, operators have applications in various areas of computer science, including system verification, plan recognition, and natural language processing. We study the complexity of the membership problem for such models, i.e., how difficult it is to determine if a string belongs to a language or not. In particular, we investigate how interleaving can be introduced into models that capture the context-free languages.

Keywords: interleaving, shuffle languages, membership problems.

1 Introduction

We study the membership problem for various language classes that make use of the shuffle operator $\odot$. When applied to a pair of strings u and v, the operator returns the set of all possible interleavings of the symbols in u and v. For example, the shuffle of ab and cd is $\{abcd, acbd, acdb, cabd, cadb, cdab\}$. The operator is lifted to languages by defining $\mathcal{L}_1 \odot \mathcal{L}_2$ to be the set $\bigcup\{u \odot v \mid u \in \mathcal{L}_1, v \in \mathcal{L}_2\}$. We also consider the shuffle closure operator, whose relationship to the shuffle operator resembles that of the Kleene star to concatenation.

Various aspects of shuffling have been studied in the theory of formal languages, see, e.g., [15,17,4,13,5,11,20,18,9,7,3]. In this paper, we take the *shuffle languages* considered by Gischer [15] and by Jedrzejowicz and Szepietowski [17] as the starting point. These are the languages defined by regular expressions augmented with the shuffle and the shuffle closure operators.

Shuffling of languages is of interest in a number of different areas:

- In the *modelling* and *verification* of systems, shuffling is, as argued by Garg and Ragunath [12], useful for modelling the interleaving of processes. There is a close connection between shuffle languages and Petri nets [15,12,6].
- The shuffle operator (often called interleaving) is used in *XML database systems* for schema definitions, see, e.g., Gelade et al. [13].
- In *plan recognition*, the objective is to identify an agent's goal or plan, based on observations of the agent's actions [8,23]. In a generalised version, a number independent agents that perform their actions in an interleaved fashion. To model this multi-agent scenario one could combine shuffle operators and context-free grammars [16]. For this approach to be tractable, the membership problem for the resulting languages must remain efficiently solvable.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 142–154, 2011.

Table 1. Summary of results for the membership problem. The shuffle languages are abbreviated by Sh, the regular by Reg, and the context-free by CF. The results of this paper appear in bold face.

	Sh	Reg $\odot$ CF	Sh $\odot$ CF	CF $\odot$ CF	CFSA
Non-Uniform	P	**P**	**P**	**NPC**	**NPC**
Uniform	NPC / **W[1]-hard**	**P**	NPC	**NPC**	**NPC**

– In *natural language processing*, there is a growing interest in linguistic models for languages with relatively free word ordering. Recent work in this direction includes parse algorithms for so-called dependency grammars [22,19].

A number of fundamental questions regarding the complexity of the membership problem for various models remain unanswered. We answer some of them in this paper. In particular, we are interested in language classes that capture the context-free languages. Among the above application areas, such languages are primarily of interest in plan recognition and natural language processing.

It is important to distinguish the *uniform* and the *non-uniform* version of the membership problem. In the uniform version, both the string and a representation of the language is given as input. Thus it is important *how* the language is represented. In the non-uniform version, only the string to be tested is considered as input. The language is fixed, and thus its representation is not important.

Contributions. To facilitate the study of languages that combine restricted forms of recursion and interleaving, we define *Concurrent Finite State Automata* (CFSA). We show that the emptiness problem for CFSA is solvable in polynomial time, list the automata's closure properties, and identify the language classes that correspond to certain syntactic restrictions.

Our complexity-results for the membership problems of various language classes are summarized in Table 1. For the full class of languages recognized by CFSA, we show that both the uniform and the non-uniform membership problem are NP-complete. For the *shuffle languages* (as used in [15,17]), the *uniform membership problem* is NP-complete [1,21], while the *non-uniform membership problem* can be decided in polynomial time [17]. We shed further light on the complexity of the membership problem by showing that the uniform version, parameterized by the number of shuffle operations, is hard for the complexity class W[1]. This indicates a strong dependence on the number of shufflings.

For the interleaving of a regular language and a context-free language, we show that the uniform (and thus also the non-uniform) membership problem can be solved in polynomial time. For the shuffling of a shuffle language and a context-free language, the uniform problem is NP-hard, since this holds already for the shuffle languages. The non-uniform problem is, however, solvable in polynomial time. For the shuffling of two context-free languages, we show that already the non-uniform version of the membership problem is NP-hard.

It should be noted that we only investigate which broad complexity classes the problems belong to. In particular, for the problems that belong to P, our aim has not been to find optimal algorithms.

Due to space limitations, we only provide proof sketches and intuitions. Full proofs of our results can be found in [2].

2 Preliminaries

Sets and numbers. If S is a set, then S^* is the set of all finite sequences of elements of S, and $\mathit{precl}(S)$ is the set of all finite prefix-closed subsets of S^*. In other words, for every $S' \in \mathit{precl}(S)$, if $uv \in S'$ for some $u, v \in S^*$ then $u \in S'$. We write $\mathbb{N}$ for the natural numbers. For $k \in \mathbb{N}$, we write $[k]$ for $\{1, \ldots, k\}$. Note that $[0] = \emptyset$. The domain of a mapping f is denoted $\mathit{dom}(f)$.

An *alphabet* is a a finite nonempty set. Let Σ be an alphabet and let ε be the empty string, then $\Sigma \cup \{\varepsilon\}$ is denoted by Σ_ε. The length of a string $w = \alpha_1 \cdots \alpha_n$ is written $|w|$, and for every $\alpha \in \Sigma$, $|w|_\alpha = |\{i \in [n] \mid \alpha_i = \alpha\}|$.

Trees. The set T_Σ of *(unranked) trees* over the alphabet Σ consists of all mappings $t\colon D \to \Sigma$, where $D \in \mathit{precl}(\mathbb{N})$. The *empty tree*, denoted t_ε, is the unique tree such that $\mathit{dom}(t) = \emptyset$. We henceforth refer to $\mathit{dom}(t)$ as the *nodes of* t and write $\mathit{nodes}(t)$ rather than $\mathit{dom}(t)$. The *size* of t, denoted $|t|$, is $|\mathit{nodes}(t)|$.

For a tree $t \in T_\Sigma$ and a node $v \in \mathit{nodes}(t)$, the *subtree of* t *rooted at* v is denoted by t/v. It is defined by $\mathit{nodes}(t/v) = \{v' \in \mathbb{N}^* \mid vv' \in \mathit{nodes}(t)\}$ and, for all $v' \in \mathit{nodes}(t/v)$, $(t/v)(v') = t(vv')$. The *leaves* of t is the set $\mathit{leaves}(t) = \{v \in \mathbb{N}^* \mid \nexists i \in \mathbb{N}\ s.t.\ vi \in \mathit{nodes}(t)\}$. The *substitution* of t' into t at node v is denoted $t[\![v \leftarrow t']\!]$. It is defined by

$$\mathit{nodes}(t[\![v \leftarrow t']\!]) = (\mathit{nodes}(t) \setminus \{vu \mid u \in \mathbb{N}^*\}) \cup \{vu \mid u \in \mathit{nodes}(t')\} \ ;$$

and, for every $u \in \mathit{nodes}(t[\![v \leftarrow t']\!])$, if $u = vv'$ for some $v' \in \mathit{nodes}(t')$ then $t[\![v \leftarrow t']\!](u) = t'(v')$, otherwise $t[\![v \leftarrow t']\!](u) = t(u)$.

For a tree $t \in T_\Sigma$ let $v_1, \ldots, v_k \in \mathit{nodes}(t)$ be the immediate child nodes of the root ordered by numeric value. That is, $\{v_1, \ldots, v_k\} = \{v \in \mathit{nodes}(t) \mid |v| = 1\}$, ordered such that $v_i < v_{i+1}$ for all $i \in [k-1]$. Then we will write t as $f[t_1, \ldots, t_k]$, where $f = t(\varepsilon)$ and $t_j = t/v_j$ for all $j \in [k]$. In the special case where $k = 0$ (i.e., when $\mathit{nodes}(t) = \{\varepsilon\}$), the brackets may be omitted, thus denoting t as f.

Shuffle operations and shuffle expressions. We recall the definitions of the operations shuffle and shuffle closure, and of shuffle expressions, from [15,17].

The *shuffle* operation $\odot$ is inductively defined as follows: for every $u \in \Sigma^*$ it is given by $u \odot \varepsilon = \varepsilon \odot u = \{u\}$, and by

$$\alpha_1 u_1 \odot \alpha_2 u_2 = \{\alpha_1 w \mid w \in (u_1 \odot \alpha_2 u_2)\} \cup \{\alpha_2 w \mid w \in (\alpha_1 u_1 \odot u_2)\} \ ,$$

for every $\alpha_1, \alpha_2 \in \Sigma$, and $u_1, u_2 \in \Sigma^*$. The operation extends to languages with

$$\mathcal{L}_1 \odot \mathcal{L}_2 = \bigcup_{u_1 \in \mathcal{L}_1, u_2 \in \mathcal{L}_2} u_1 \odot u_2 \ .$$

The *shuffle closure* of a language $\mathcal{L} \in \Sigma^*$, denoted $\mathcal{L}^\odot$, is

$$\mathcal{L}^\odot = \bigcup_{i=0}^{\infty} \mathcal{L}^{\odot i}, \text{ where } \mathcal{L}^{\odot 0} = \{\varepsilon\} \text{ and } \mathcal{L}^{\odot i} = \mathcal{L} \odot \mathcal{L}^{\odot i-1} \ .$$

Shuffle expressions are regular expressions that can additionally use the shuffle operators. The shuffle expressions over the alphabet Σ are as follows. The empty string ε, the empty set $\emptyset$, and every $\alpha \in \Sigma$ is a shuffle expression. If s_1 and s_2 are shuffle expressions, then so are $(s_1 \cdot s_2), (s_1 + s_2), (s_1 \odot s_2), s_1^*$, and $s_1{}^{\odot}$, where $\cdot$ denotes concatenation and $+$ denotes disjunction. Shuffle expressions that do not use the shuffle closure operator are called *closure free* shuffle expressions. The language $\mathcal{L}(s)$ of a shuffle expression s is defined in the usual way. *Shuffle languages* are the languages defined by shuffle expressions.

3 Concurrent Finite-State Automata

In this section, we introduce *concurrent finite-state automata* (CFSA). They are inspired by *recursive Markov models*, but differ in that recursive calls can be made in parallel. This allows an unbounded number of invocations to be executed simultaneously, but each symbol can only be read by one invocation. In Defintion 1, $p^{\odot}$ is to be read as single symbol. In the later definition of CFSA semantics, transitions of the form $(q, \alpha, q'[p^{\odot}])$ will be interpreted as rule schema.

Definition 1 (CFSA). A *Concurrent FSA* is a tuple $M = (Q, \Sigma, \delta, I)$, where
- Q is a finite set of *states*;
- Σ is an alphabet of *input symbols*;
- $\delta \subseteq Q \times \Sigma_\varepsilon \times T$ is a set of *transitions*, where T is the finite set

$$\{q, q[p], q[p, p'], q[p^{\odot}] \mid q, p, p' \in Q\} \cup \{t_\varepsilon\} \ .$$

  A transition $(q, \alpha, t) \in \delta$ is
  - *terminal* if $|nodes(t)| = 0$,
  - *horizontal* if $|nodes(t)| = 1$, and
  - *vertical* if $|nodes(t)| > 1$.
- $I \subseteq Q$ is a set of *initial* states. □

Remark. For simplicity, we henceforth assume, without loss of generality, that the terminal transitions form a subset of $Q \times \{\varepsilon\} \times \{t_\varepsilon\}$.

Whereas a FSA is in a single state at a time, a concurrent FSA maintains a branching call-stack of states, represented as an unranked tree. In each step, exactly one leaf node of the state tree is rewritten. Vertical transitions model the invocation of child processes; horizontal transitions the continued execution within a process; and terminal transitions the completion of a process. A CFSA accepts a string if, upon reading the string, it can reach a configuration in which every processes has been completed, i.e., the state tree is empty.

Definition 2 (Concurrent FSA semantics). A *configuration* of the CFSA $M = (Q, \Sigma, \delta, I)$ is a tuple $(w, t) \in \Sigma^* \times T_Q$. The set of all configurations of M is denoted $\Delta(M)$. A configuration $(w, t) \in \Delta(M)$ is *initial* (with respect to the string $w \in \Sigma^*$) if $t \in I$.

In its initial configuration, M is in the unique initial state q_0 and has yet to consume any input symbol.

$$(\ \lfloor\lfloor\lceil\rfloor\lfloor\rceil\rfloor\rfloor, \ \ q_0)$$

To proceed, M must nondeterministically choose the transition $(q_0, \varepsilon, q_0'[q_1, q_2])$.

$$(\ \lfloor\lfloor\lceil\rfloor\lfloor\rceil\rfloor\rfloor, \ \ q_0'[q_1, q_2] \ \ )$$

By the transition $(q_1, \lfloor, q_1'[q_1])$, M reaches the configuration

$$(\ \lfloor\lceil\rfloor\lfloor\rceil\rfloor\rfloor, \ \ q_0'[q_1'[q_1], q_2] \ \ )$$

and by $(q_1, \lfloor, q_1'[q_1])$ and $(q_2, \lceil, q_2'[q_2])$ the configuration

$$(\ \rfloor\lfloor\rceil\rfloor\rfloor, \ \ q_0'[q_1'[q_1'[q_1]], q_2'[q_2]] \ \ )$$

Now M nondeterministically guesses that it is time to read the symbol $\rfloor$. It prepares by deleting the leaf labelled q_1 using transition $(q_1, \varepsilon, t_\varepsilon)$ to get

$$(\ \rfloor\lfloor\rceil\rfloor\rfloor, \ \ q_0'[q_1'[q_1'], q_2'[q_2]] \ \ )$$

and then $(q_1', \rfloor, q_1)$ to get

$$(\ \lfloor\rceil\rfloor\rfloor, \ \ q_0'[q_1'[q_1], q_2'[q_2]] \ \ )$$

Again, $(q_1, \lfloor, q_1'[q_1])$ lets M read $\lfloor$,

$$(\ \rceil\rfloor\rfloor, \ \ q_0'[q_1'[q_1'[q_1]], q_2'[q_2]] \ \ )$$

and $(q_2, \varepsilon, t_\varepsilon), (q_2', \rceil, q_2)$ produces

$$(\ \rfloor\rfloor, \ \ q_0'[q_1'[q_1'[q_1]], q_2] \ \ ) \ .$$

Thereafter, applying the transition sequence $(q_1, \varepsilon, t_\varepsilon), (q_1', \rfloor, q_1)$ twice yields

$$(\ \varepsilon, \ \ q_0'[q_1, q_2] \ \ ) \ .$$

Although the entire input has been read, M does not accept until the state tree has been reduced to the empty tree. This can be done by applying $(q_1, \varepsilon, t_\varepsilon), (q_2, \varepsilon, t_\varepsilon)$ to get

$$(\ \varepsilon, \ \ q_0' \ \ ) \ ,$$

and finally $(q_0', \varepsilon, t_\varepsilon)$ to reach

$$(\ \varepsilon, \ \ t_\varepsilon \ \ ) \ .$$

Fig. 1. The CFSA M of Example 1 accepts the input string $\lfloor\lfloor\lceil\rfloor\lfloor\rceil\rfloor\rfloor$

Let $(w, t), (w', t') \in \Delta(M)$. There is a *transition step* from (w, t) to (w', t'), written $(w, t) \to (w', t')$, if there is a transition $(q, \alpha, s) \in \delta$ and node $v \in \mathit{nodes}(t)$ such that $w = \alpha w'$, $t/v = q$ (so v is a leaf), and either

- $s \in T_Q$ and $t' = t[\![v \leftarrow s]\!]$, or
- $s = p'[p^{\odot}]$ and $t' = t[\![v \leftarrow p'[\underbrace{p, \ldots, p}_{n}]]\!]$ for some for $p, p' \in Q$ and $n \in \mathbb{N}$.

As usual, the reflexive and transitive closure of $\to$ is denoted $\xrightarrow{*}$. The *language recognised by* M is $\mathcal{L}(M) = \{w \in \Sigma^* \mid \exists q \in I : (w, q) \xrightarrow{*} (\varepsilon, t_\varepsilon)\}$. □

For the sake of brevity only the state-tree part of a configuration, called a *configuration tree*, may be shown in cases where the string is irrelevant.

Example 1. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the Dyck languages[1] over the symbol pairs $\lfloor, \rfloor$ and $\lceil, \rceil$, respectively. Their shuffle $\mathcal{L} = \mathcal{L}_1 \odot \mathcal{L}_2$ is recognised by the concurrent FSA $M = (\{q_0, q_1, q_1', q_2, q_2'\}, \{\lfloor, \rfloor, \lceil, \rceil\}, \delta, \{q_0\})$, where

$$\delta = \{ \ (q_0, \varepsilon, q'[q_1, q_2]), \quad (q_0', \varepsilon, t_\varepsilon), \quad (q_1, \lfloor, q_1'[q_1]), \quad (q_1', \rfloor, q_1), \\ (q_1, \varepsilon, t_\varepsilon), \quad (q_2, \lceil, q_2'[q_2]), \quad (q_2', \rceil, q_2), \quad (q_2, \varepsilon, t_\varepsilon) \quad \} \ .$$

[1] A Dyck language consists of all well-balanced strings over a given set of parentheses.

To illustrate the automaton's semantics, we step through an accepting run of M on the string $w = \lfloor\lfloor\lceil\rfloor\lfloor\rceil\rfloor\rfloor$ (see Figure 1). Note that since $w \in w_1 \odot w_2$ for $w_1 = \lfloor\lfloor\rfloor\lfloor\rfloor\rfloor \in \mathcal{L}_1$ and $w_1 = \lceil\rceil \in \mathcal{L}_2$, it follows that $w \in \mathcal{L}_1 \odot \mathcal{L}_2$. □

It is known that $\mathcal{L}_1 = \{a^n b^n \mid n \in \mathbb{N}\}$ is a context-free language, but it is not a shuffle language. Conversely, $\mathcal{L}_2 = \{w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c\}$ is a shuffle language but is not context-free. Both $\mathcal{L}_1$ and $\mathcal{L}_2$ is recognized by a CFSA, and so is $\mathcal{L}_1 \cup \mathcal{L}_2$, which is neither a context-free nor a shuffle language. Thus the CFSA languages properly extend both the context-free languages and the shuffle languages. They also have comparatively nice closure properties.

Theorem 1. *The languages recognised by CFSA are closed under union, concatenation, Kleene star, shuffle and shuffle closure. They are not closed under intersection with a regular language or complementation.*

Restrictions and expressive power. The restrictions considered here are as follows. A CFSA $M = (Q, \Sigma, \delta, I)$ is

- *horizontal* if δ contains no vertical transitions;
- *non-branching* if every vertical transition is in $Q \times \Sigma \times \{q'[q] \mid q, q' \in Q\}$;
- *finitely branching* if no vertical transition is in $Q \times \Sigma \times \{q'[q^{\odot}] \mid q, q' \in Q\}$;
- *acyclic* if there is no configuration $(w,t) \in \Delta(M)$ and state $q \in Q$ such that q appears twice on a path from the root of t to a leaf.

Theorem 2. *A language is:*

- *regular if and only if it is recognised by a horizontal CFSA;*
- *context-free if and only if it is recognised by a non-branching CFSA;*
- *a shuffle language if and only if it is recognised by an acyclic CFSA;*
- *a closure-free shuffle language if and only if it is recognised by an acyclic and finitely branching CFSA.*

Since the closure free shuffle languages are regular [14], we can conclude that acyclic and finitely branching CFSA also recognize the regular languages.

CFSA do not provide the full power of linear bounded Turing machines:

Theorem 3. *The languages recognised by CFSA are properly contained in the context-sensitive languages.*

Since not all CFSA-languages are context-free (e.g., there are non-context-free shuffle languages), we conclude that their expressive powers lies strictly between that of context-free grammars and that of context-sensitive grammars.

Also unlike linear bounded Turing machines, CFSA can be efficiently checked for emptiness.

Theorem 4. *The emptiness problem for CFSA is decidable in polynomial time.*

4 Membership Problems

The membership problem for unrestricted CFSA is intractable, both in the uniform and the non-uniform case.

Theorem 5. *Both the uniform and the non-uniform membership problem for CFSA is NP-complete.*

NP-hardness for the uniform membership problem for shuffle expressions is already known; see, e.g., [1,21]. We postpone the hardness proof for the non-uniform case until Theorem 9 which shows it for a subclass of the CFSA. The following lemma establishes that the membership problem for CFSA is in NP.

Lemma 1. *Given a CFSA $M = (Q, \Sigma, \delta, I)$ and a string $w \in \Sigma^*$ it is possible to determine if $w \in \mathcal{L}(M)$ in nondeterministic polynomial time.*

Proof sketch. We rewrite M so that if $(\varepsilon, q) \xrightarrow{*} (\varepsilon, t)$ for some $q \in Q$ and $t \in \{s \in T_Q \mid |s| \leq 2\}$ then $(q, \varepsilon, t) \in \delta$. The transitions that must be added to δ can be identified in polynomial time by iterating over every $q \in Q$ and every tree $t \in \{s \in T_Q \mid |s| \leq 2\}$. To determine if $(\varepsilon, q) \xrightarrow{*} (\varepsilon, t_\varepsilon)$ corresponds to the emptiness test (which is polynomial by Theorem 4) on $M' = (Q, \Sigma, \delta', \{q\})$ where $\delta' \subseteq \delta$ contains only the transitions that reads ε.

The extension of δ allow us to assume that a run of M never needs to create "unnecessary" nodes only to later delete them without reading any symbols from the input string. Consider any $w \in \mathcal{L}(M)$ and notice that in the shortest accepting run of M on w, no intermediary configuration tree ever has more than $|w|$ leaves. Otherwise one of those leaves would consume the empty string and be superfluous. Similarly, no such tree ever has height greater than $|Q|(|w|+1)$. Otherwise some state q must occur more than $|w|+2$ times on the longest path, so at least one section of the path delimited by q nodes reads the empty string, which means that the run could be shortened by omitting this loop on q.

This establishes that all configuration trees in a shortest accepting run are polynomial in size of the input string. Since no unnecessary nodes are generated, only a polynomial number of transitions are needed to go from one configuration tree to another. Together this means that there is a polynomial that bounds the length of the shortest accepting run on every string in $\mathcal{L}(M)$. Thus, a nondeterministic algorithm can be constructed by guessing an accepting run and then verifying that it respects the transitions in δ. □

We now turn to the membership problem for acyclic CFSA.

Corollary 1. *For acyclic CFSA*

1. *the non-uniform membership problem is solvable in polynomial time, and*
2. *the uniform membership problem is NP-complete.*

The uniform membership problem is NP-complete already for acyclic and finitely branching CFSA, which only recognise regular languages. This is not too surprising, since, e.g., the similar NFA(&) from [13], which also recognize the regular languages, has PSPACE-complete uniform membership. For some languages,

CFSA offer a more succinct form of representation than nondeterministic finite automata (and than the shuffle automata from [17]). One example is the language family $\{\{a^n\} \mid n \in \mathbb{N}\}$, for which the smallest NFAs (and shuffle automata) have sizes linear in n, while the smallest CFSAs are logarithmic in n.

Corollary 1 states that the problem is polynomial for a fixed automaton but NP-hard if the automaton is considered input. The question then remains whether the size of the automaton merely influences the coefficients of the polynomial or if it affects the degree itself. We give a partial answer by showing that when parameterized by the maximal size of a configuration tree for the automaton, the uniform membership problem for acyclic and finitely branching CFSAs is *not fixed-parameter tractable*, unless FPT = W[1]. This class equivalence is considered very unlikely and would have far-reaching complexity-theoretic implications. For more on parameterized complexity theory, see, e.g., [10].

We state the result for acyclic and finitely branching CFSA, but it could be equivalently stated for closure-free shuffle expressions. We first define the parameterized version of the problem.

Definition 3. An instance of the parameterized uniform membership problem for acyclic and finitely branching CFSA is a pair (M, w) where M is an acyclic and finitely branching CFSA over a finite alphabet Σ and w is a string in Σ^*. The parameter is the maximal size of any configuration tree for M. The question is whether $w \in \mathcal{L}(M)$. □

For acyclic and finitely branching CFSA, the maximal size of the configuration trees depends only on the automaton. If the problem was fixed-parameter tractable, it would have an algorithm with running time $f(k) \cdot n^c$, where f is a computable function, k is the parameter (the maximal tree size), n is the instance size, and c is a constant. Theorem 6 gives strong evidence to the contrary.

Theorem 6. *The parameterized uniform membership problem for acyclic and finitely branching CFSA is W[1]-hard.*

The proof is by a fixed-parameter reduction from parameterized clique, which is known to be $W[1]$-complete [10].

Definition 4. An instance of k-Clique is a pair (G, k), where $G = (V, E)$ is an undirected graph and k is an integer. The question is whether there is a set $C \subseteq V$ of size k such that the subgraph of G induced by C is complete. The parameter is k. □

Proof sketch. Let $(G = (V, E), k)$ be an instance of k-Clique, and let $n = |V|$ and $m = |E|$. We assume that the vertices are named $v_1, \ldots, v_n$ and that the edges are named $e_{i,j}$, where $i < j$, and construct an alphabet $\Sigma = V \cup E$. Next, we construct a word $w_G = v_1^k \cdot v_2^k \cdots v_n^k \cdot \mathit{edges}$, where *edges* is any enumeration of the edges in E. It remains to construct an acyclic and finitely branching CFSA M_G such that $w_G \in \mathcal{L}(M_G)$ if and only if G has a clique of size k and such that the maximum configuration tree size for M_G depends only on k. The idea is to let M_G read the shuffle of

- a regular language $s = (v_1^k + v_2^k + \cdots + v_n^k)^{n-k}$ that consumes all copies of $n - k$ vertex names;
- a regular language $t = V^* \cdot E^*$ that does the "garbage collection"; and
- $k(k-1)/2$ copies of a regular language $u = \Sigma_{e_{i,j} \in E}(v_i \cdot v_j \cdot e_{i,j})$.

Each instance of u will thus consume (one instance each of) the names of two vertices and the name of the edge that connects the two vertices. Since only k vertex names are represented in w_G after s has consumed all copies of $n - k$ of them, this means that for all copies of u to be matched, there must be $k(k-1)/2$ edges in G whose endpoints are all in a set of vertices of size k. This, in turn, means that G has a clique of size k. Constructing an acyclic and finitely branching CFSA M_G that does this is straightforward. It is also clear that M_G can be constructed in such a way that the maximum size of a configuration tree is bounded by $O(k^2)$, which makes this a fixed-parameter reduction. □

The following corollary is immediate.

Corollary 2. *The uniform membership problem for closure-free shuffle expressions, parameterized by the number of shuffle operators, is W[1]-hard.*

We next show that the shuffle of a context-free language and a regular language is efficiently recognizable, even if the language descriptions are part of the input.

Theorem 7. *The uniform membership problem for the shuffle of two languages, one represented by context-free grammar and one represented by a nondeterministic finite automaton, is solvable in polynomial time.*

Proof sketch. Let $G = (N, \Sigma, \delta, S)$ and $M = (Q, \Sigma, \gamma, I, F)$ be a context-free grammar on Chomsky normal form and an NFA, respectively.

To test membership in $\mathcal{L}(G) \odot \mathcal{L}(M)$, we extend the CYK algorithm for context-free grammars. A *parse triple* for G and M over a string $w = a_1 \cdots a_m$ is a triple $(A, q_1, q_2) \in (N \cup \{\varepsilon\}) \times Q \times Q$ such that $w \in \mathcal{L}(M_{q_1,q_2}) \odot \mathcal{L}(G_A)$, where $M_{q_1,q_2} = (Q, \Sigma, \gamma, \{q_1\}, \{q_2\})$, and $G_A = (N, \Sigma, \delta, A)$, unless $A = \varepsilon$ in which case $\mathcal{L}(G_\varepsilon) = \{\varepsilon\}$.

Like in the CYK algorithm the parse triples are computed for each substring, starting with the substrings of length 1 and then combining triples to form new triples for successively longer strings. For example, if (A, q, q') and (B, q', q'') are triples for the strings w' and w'' respectively, then (C, q, q'') is a triple for $w'w''$ if G contains the rule $C \to AB$. Since there are at most $(|N| + 1) \cdot |Q|^2$ distinct parse triples and $O(|w|^2)$ substrings, this can be done in polynomial time. In the end, $w \in \mathcal{L}(G) \odot \mathcal{L}(M)$ if and only if there is a parse triple (S, q_I, q_F) for the whole of w such that S is the start symbol of G, $q_I \in I$, and $q_F \in F$. □

Since acyclic and finitely branching CFSA only contribute a more compact representation of the regular languages, Theorem 7 extends to non-uniform membership for the shuffle of a context-free language and a closure-free shuffle language:

Corollary 3. *The non-uniform membership problem for the shuffle of two languages, one represented by a context-free grammar and one represented by an acyclic and finitely branching CFSA, is solvable in polynomial time.*

Extending Theorem 7 with techniques inspired by [17], we get the following:

Theorem 8. *The non-uniform membership problem for the shuffle of a shuffle language and a context-free language is solvable in polynomial time.*

Proof sketch. Assume that the languages are represented by an acyclic CFSA M and a context-free grammar G. Just as in the proof of Theorem 7, we extend the CYK algorithm to work with triples. The only difference is that each triple consists of a nonterminal from G and two configuration trees for M.

For the algorithm to run in polynomial time, the number of configuration trees that need to be stored must be polynomially bounded. It can be shown that this is the case by taking advantage of symmetries in the configuration trees. Intuitively, the shuffle-closure should be applied as sparingly as possible. □

Next, we show that the uniform membership problem for $\mathcal{L}(A_1) \odot \mathcal{L}(A_2)$, where $\mathcal{L}(A_1)$ and $\mathcal{L}(A_2)$ are context-free languages, is NP-complete. The construction makes use of push-down automata (PDA), which are well known to be equivalent to context-free grammars and can be obtained from them in polynomial time. It also uses two-stack PDA, which have two independent stacks, and are known to be equivalent to Turing machines. NP-hardness is demonstrated by constructing two reductions. The first is a polynomial reduction which takes an arbitrary nondeterministic Turing machine and constructs two context-free languages, $\mathcal{L}(A_{sim})$ and $\mathcal{L}(A_{comp})$, which depend *only* on the Turing machine. The second reduction takes any string w and constructs a string w' so that $w' \in \mathcal{L}(A_{sim}) \odot \mathcal{L}(A_{comp})$ if and only if w is accepted by A. This second reduction will be polynomial in $|w| + n$, where n is the number of steps that A takes to accept or reject w, which means that A can only be simulated a polynomial number of steps with a polynomial reduction, but this is sufficient to solve all problems in NP.

The nondeterministic Turing machine A is assumed to actually be represented as a nondeterministic two-stack push-down automaton. We assume, without loss of generality, that A always starts by reading its entire input onto its first stack, and that the input alphabet is $\{0, 1\}$. Additionally, we assume that there is a polynomial P such that A accepts or rejects a string w in at most $P(|w|)$ steps.

The input string reductions takes any valid input w for A and constructs

$$w' = w \cdot \$\$ \underbrace{[[\mathrm{push}_0\mathrm{push}_1\mathrm{pop}_0\mathrm{pop}_1]][[\mathrm{push}_0\mathrm{push}_1\mathrm{pop}_0\mathrm{pop}_1]]\ldots.}_{P(|w|)\text{ copies}}$$

The output alphabet of the reduction is $\{0, 1, \mathrm{push}_0, \mathrm{push}_1, \mathrm{pop}_0, \mathrm{pop}_1,], [, \$\}$.

The push-down automaton A_{sim} is constructed to simulate A. A string is a *valid stack run* (VSR) if it is of the form $[\mathrm{push}_0][\mathrm{push}_1][\mathrm{pop}_1]$, that is, the push/pop symbols surrounded by brackets, such that each pop_x, $x \in \{0, 1\}$, corresponds to a push_x earlier in the string. The stack discipline must be maintained; $[\mathrm{push}_1][\mathrm{push}_0][\mathrm{pop}_1]$ is not a VSR. A_{sim} is constructed from A so that:

a string w is accepted by A if and only if there exists some VSR s such that $w \cdot \$ \cdot s \in \mathcal{L}(A_{sim})$. The trick is that A_{sim} simulates the first stack of A on its own (only) stack, and then exploits this VSR suffix string to simulate the second stack of A. With this idea in place, the simulation is straightforward. By assumption, A starts by reading all of its input onto its first stack. Thus A_{sim} reads all input up until the \$ symbol onto its stack and then discards \$. At this point, A and A_{sim} are in equivalent states. If A contains a rule of the form "if in q we may pop 0 from the first stack and 1 from the second, in which case we push 1 onto the second stack and go to state q'", then A_{sim} can, from state q, pop 0 from its stack, read the string "$[\text{pop}_1][\text{push}_0]$", and go to state q'.

The context-free language $\mathcal{L}(A_{comp})$ is in the shuffle to read the "extraneous" symbols from the constructed input string so that the remainder becomes exactly what A_{sim} expects. It always starts by reading \$, then, for each $[[\text{push}_0 \dots]]$ template, it reads the extra bracket pair, and three of the stack operations, so that the remainder is any possible VSR (as described above). For example, since $[\text{push}_1][\text{push}_0][\text{pop}_0]$ is a VSR there is a string

$$\$[\text{push}_0\text{pop}_0\text{pop}_1][\text{push}_1\text{pop}_0\text{pop}_1][\text{push}_0\text{push}_1\text{pop}_1] \in \mathcal{L}(A_{comp}).$$

Constructing this grammar is trivial (start with a Dyck language).

Theorem 9. *For the shuffle of two context-free languages, the non-uniform membership problem is NP-complete.*

Proof sketch. The problem is trivially in NP, whereas NP-hardness follows from the construction above. Take a nondeterministic Turing machine A and construct A_{sim} and A_{comp} as above. Take any string w and construct $w' = w \cdot \$\$ \cdot [[\dots$. What then happens is that A_{comp} will from w' read one dollar-sign, and then for each double-bracketed substring read one bracket-pair and three of the four stack operation, in such a way that the remainder of the input will have the form $w \cdot \$ \cdot s$ for all possible VSR s. By construction, there exists such a string that is a member of $\mathcal{L}(A_{sim})$ if and only if w is accepted by A.

A_{comp} is a constant language, A_{sim} depends only on A, $|w'|$ is polynomial in the size of the input string w, and the number of steps taken by A (assumed to be polynomial), establishing non-uniform NP-completeness. □

5 Conclusions and Future Work

Concurrent finite-state automata combine the expressive power of context-free and shuffle languages. The CFSA languages are properly included in the context-sensitive languages, and minor restrictions of the device suffice to obtain the regular, context-free, and shuffle languages, respectively. CFSA have comparatively nice closure properties, and can be sanity-checked in polynomial time.

To be of practical use, at least the non-uniform membership problem needs to be efficiently decidable. This is known to be true for the shuffle languages, but our

analysis shows that the efficiency depends heavily on the number of shuffle operations used. We also obtain that the non-uniform membership problem remains polynomial for the shuffle of a shuffle language and a context-free language. For the shuffle of two context-free languages, however, it is NP-complete.

Ideally, also the uniform membership problem should be solvable in polynomial time. The only language class we studied for which this is the case, unless P=NP, is the interleaving of a regular language and a context-free language.

Future work will strive to determine the complexity of the non-uniform membership problem for further restrictions of CFSA. If even very sparse use of shuffling has a large negative impact on the complexity, one could consider replacing the shuffle operator with weaker alternatives, such as unordered shuffle.

References

1. Barton, G.E.: On the complexity of ID/LP parsing 1. Comput. Linguist. 11(4), 205–218 (1985)
2. Berglund, M., Björklund, H., Högberg, J.: Recognizing shuffled languages. Technical Report UMINF 11.01, Computing Sci., Umeå University (2011), http://www8.cs.umu.se/research/uminf/index.cgi
3. Biegler, F., Daley, M., McQuillan, I.: On the shuffle automaton size for words. In: DCFS, pp. 79–89 (2009)
4. Björklund, H., Bojańczyk, M.: Shuffle expressions and words with nested data. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 750–761. Springer, Heidelberg (2007)
5. Bloom, S.B., Ésik, Z.: Axiomatizing shuffle and concatenation in languages. Information and Comptuation 139(1), 62–91 (1997)
6. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: Proc. LICS 2006, pp. 7–16 (2006)
7. 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)
8. Carberry, S.: Techniques for plan recognition. User Modeling and User-Adapted Interaction 11(1-2), 31–48 (2001)
9. Daley, M., Domaratzki, M., Salomaa, K.: Orthogonal concatenation: Language equations and state complexity. J. Universal Comp. Sci. 16(5), 653–675 (2010)
10. Downey, R., Fellows, M.: Parameterized Complexity. Springer, Heidelberg (1999)
11. Ésik, Z., Bertol, M.: Nonfinite axiomatizability of the equational theory of shuffle. Acta Informatica 35(6), 505–539 (1998)
12. Garg, V., Ragunath, M.: Concurrent regular expressions and their relationship to petri nets. Theor. Comput. Sci. 96(2), 285–304 (1992)
13. Gelade, W., Martens, W., Neven, F.: Optimizing schema languages for XML: Numerical constraints and interleaving. SIAM J. on Comp. 39(4), 1486–1530 (2009)
14. Ginsburg, S.: The Mathematical Theory of Context Free Languages. McGraw-Hill, New York (1966)
15. Gischer, J.: Shuffle languages, petri nets, and context-sensitive grammars. Comm. ACM 24(9), 597–605 (1981)
16. Högberg, J., Kaati, L.: Weighted unranked tree automata as a framework for plan recognition. In: Proc. Fusion (2010) (to appear)
17. Jedrzejowicz, J., Szepietowski, A.: Shuffle languages are in P. Theor. Comput. Sci. 250(1-2), 31–53 (2001)

18. Kari, L., Sosík, P.: Aspects of shuffle and deletion on trajectories. Theor. Comput. Sci. 332, 47–61 (2005)
19. Kuhlmann, M., Satta, G.: Treebank grammar techniques for non-projective dependency parsing. In: Proc. EACL, pp. 478–486 (2009)
20. Mateescu, A., Rozenberg, G., Salomaa, A.: Shuffle on trajectories: syntactic constraints. Theor. Comput. Sci. 197, 1–56 (1998)
21. Mayer, A., Stockmeyer, L.: Word problems – this time with interleaving. Inform. and Comput. 115, 293–311 (1994)
22. Nivre, J.: Non-projective dependency parsing in expected linear time. In: Proc. ACL-IJCNLP 2009, pp. 351–359 (2009)
23. Schmidt, C., Sridharan, N., Goodson, J.: The plan recognition problem: An intersection of psychology and artificial intelligence. A.I. 11(1,2) (1978)

Unary Pattern Avoidance in Partial Words Dense with Holes*

Francine Blanchet-Sadri[1], Kevin Black[2], and Andrew Zemke[3]

[1] Department of Computer Science, University of North Carolina,
P.O. Box 26170, Greensboro, NC 27402–6170, USA
blanchet@uncg.edu
[2] Department of Mathematics, Harvey Mudd College,
301 Platt Blvd., Claremont, CA, 91711, USA
[3] School of Mathematical Sciences, Rochester Institute of Technology,
85 Lomb Memorial Drive, Rochester, NY, 14623-5603, USA

Abstract. A *partial word* is a sequence of symbols over a finite alphabet that may have some undefined positions, called *holes*, that match every letter of the alphabet. Previous work completed the classification of all unary patterns with respect to partial word avoidability, as well as the classification of all binary patterns with respect to non-trivial partial word avoidability. In this paper, we pose the problem of avoiding patterns in partial words very dense with holes. We define the concept of hole sparsity, a measure of the frequency of holes in a partial word, and determine the minimum hole sparsity for all unary patterns in the context of trivial and non-trivial avoidability.

1 Introduction

Notions and techniques related to patterns such as repetitions in strings find applications in several areas of theoretical and applied computer science, notably in text processing, data compression, computational biology, string and pattern matching algorithms (see [7] for an overview on repetitions in strings). In pattern matching, several algorithms take advantage of the repetitions of the pattern to speed up the search of its occurrences in a text. On the other hand, non-repetitive sequences, those avoiding patterns such as squares, or square-free words, have been used to build several counterexamples in context-free languages, groups, lattice of varieties, partially ordered sets, semigroups, symbolic dynamics, to name a few (see [8] for a survey on pattern avoidance). For example, Main, Bucher and Haussler found applications of an infinite square-free co-CFL language [11]. They proved several conjectures on context-free languages by using the set of all words that are not prefixes of the Thue-Morse infinite sequence, well known to

* This material is based upon work supported by the National Science Foundation under Grant No. DMS–0754154. The Department of Defense is also gratefully acknowledged. We thank Robert Mercaş for very valuable help in the writing of this paper. A World Wide Web server interface has been established at www.uncg.edu/cmp/research/patterns for automated use of the programs.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 155–166, 2011.

be square-free, which provides a CFL language with an infinite square-free complement. Words avoiding more general patterns than squares find applications in algorithmic problems on algebraic structures.

The topic of *(un)avoidable patterns* in words has been extensively studied, providing a framework for better understanding properties of repetitive and non-repetitive sequences. A pattern p is a non-empty word over an alphabet of *variables*, which are usually denoted by α, β, γ, etc. The terminology of avoidable pattern was introduced by Bean, Ehrenfeucht and McNulty [2] and by Zimin [21]. They proved the fundamental result that it is decidable whether a pattern p is avoidable; in fact, if p is over m variables, then p is avoidable if and only if w_m avoids p, where w_m is recursively defined by $w_1 = 1$ and $w_m = w_{m-1}mw_{m-1}$, $m > 1$. However, the complexity of deciding avoidability has remained open. The problem "Is it decidable, given a pattern p and an integer k, whether p is k-avoidable (or avoidable over a k-letter alphabet)?" has also remained open. An alternative is the problem of classifying all the patterns over a fixed number of variables m, that is, to find the smallest k such that p is k-avoidable, called the *avoidability index* of p, where p is such pattern. In the context of full words, the case $m = 1$ of the unary patterns, or powers of a variable α, were investigated by Thue [19,20]: α is unavoidable, $\alpha\alpha$ is 2-unavoidable but 3-avoidable, and α^n with $n \geq 3$ is 2-avoidable. The case $m = 2$ has been completely classified [5,10,16,17,18], the case $m \geq 3$ has also been the subject of investigation [5,10,13].

An understanding of avoidable patterns is needed in the more general context of partial words, which allow for incomplete or corrupted data. A partial word is a sequence of symbols over a finite alphabet that may have some undefined positions, called holes, denoted by $\diamond$'s. Here $\diamond$ is *compatible* with, or matches, every letter of the alphabet. In this context, in order for a pattern p to occur in a partial word, for each variable α of p, all of its substituted partial words be pairwise compatible. For the case $m = 1$, both α and $\alpha\alpha$ are unavoidable, and so their avoidability indices in partial words is ∞. In [12], the case of $\alpha^n, n \geq 3$ was considered, the avoidability index in partial words being two, settling the classification of the unary patterns. For the case $m = 2$, it turns out that with respect to *non-trivial avoidability*, in which no variable is substituted by only one hole, the avoidability index of a binary pattern, one over two variables α and β, coincides for both the partial and the full word cases [4]; the avoidability indices of *almost all* binary patterns in terms of (not restricted to non-trivial) avoidability have also been found.

In [14], the authors introduce the concepts of c-approximate and α-similarity. Here a word uv with $|u| = |v|$, is considered to be a c-approximate square, if u and v differ on at most c positions, and a α-similar square if the ratio between the number of positions u and v agree on and the distance of u is at least α. As we can see one of the notions is "additive," while the other "multiplicative," but both definitions are based on the *Hamming distance* representing the number of positions on which u and v differ.

Following all this, the next problem is how "dense" in holes can a partial word defined over a fixed alphabet be, while still avoiding a given pattern. In this paper, we define a new concept, that of hole sparsity, representing the minimum length each factor of an infinite word should have, such that it contains at least a hole. Based on this notion, we compute the minimum hole sparsity necessary for a word defined over an alphabet of size k to avoid a unary pattern α^n.

The contents of our paper is as follows: In Section 2, we recall the basic definitions regarding partial words and patterns in partial words. In Section 3, we introduce the concept of hole sparsity, which measures the frequency of holes in a partial word. Moreover, we pose the problem of determining the minimum hole sparsity a partial word can have while still avoiding a pattern, and present some tools that are useful for deciding the avoidability of a pattern over a given alphabet with a given hole sparsity. In Sections 4 and 5, we determine this minimum hole sparsity for unary patterns, for words over alphabets of all sizes, in the context of trivial and non-trivial avoidability. Finally in Section 6, we conclude with our classification of all unary patterns with respect to hole sparsity.

2 Preliminaries

For more information on partial words, the reader is referred to [3]. For patterns in full (resp., partial) words, he/she is referred to Chapter 3 of [10] (resp., [4]).

Throughout this paper, A is a fixed non-empty finite alphabet. A finite *partial word* of length n over A can be defined as a function $u : \{0, \ldots, n-1\} \to A_\diamond$, where $A_\diamond = A \cup \{\diamond\}$. We call the elements of A *letters* (the symbol $\diamond$ is not called a letter). Write $|u|$ for the length of u. For $0 \leq i < n$, if $u(i) \in A$, then i belongs to the *domain* of u, denoted $D(u)$, and if $u(i) = \diamond$, then i belongs to the *set of holes* of u, denoted $H(u)$. Whenever $H(u)$ is empty, u is a *full* word. Refer to an occurrence of the symbol $\diamond$ as a *hole*. Denote by A^* (resp., $A_\diamond^*$) the set of all finite words (resp., partial words) over A. Abbreviate $A^* \backslash \{\varepsilon\}$ by A^+ and $A_\diamond^* \backslash \{\varepsilon\}$ by $A_\diamond^+$. Under the concatenation operation, A^* and $A_\diamond^*$ form monoids whose identities are the empty word denoted by ε. An infinite partial word over A is a function $u : \mathbb{N} \to A_\diamond$.

A partial word v is a *factor* of the partial word u if there exist x, y such that $u = xvy$. If $x = \varepsilon$, then v is a *prefix* of u; if $y = \varepsilon$, then v is a *suffix* of u. The powers of a finite partial word u are defined recursively by $u^0 = \varepsilon$ and for $n \geq 1$, $u^n = uu^{n-1}$. Two partial words u and v of equal length are *compatible*, denoted $u \uparrow v$, if $u(i) = v(i)$ whenever $i \in D(u) \cap D(v)$. If u, v are non-empty and compatible, then uv is called a *square*. The partial word u is *contained* in v, denoted $u \subset v$, if $|u| = |v|$ and $u(i) = v(i)$ for all $i \in D(u)$.

Let E be a non-empty finite set of symbols, $E \cap A = \emptyset$, whose elements are denoted by α, β, γ, etc. Symbols in E are called *variables*, and finite words over E are called *patterns*. The pattern $p = \alpha_0 \cdots \alpha_{n-1}$, where each α_i is a variable, *occurs* in a partial word w (or w *meets* p) if there is a factor $u_0 \cdots u_{n-1}$ in w, where u_i, u_j are non-empty and compatible whenever $\alpha_i = \alpha_j$; otherwise, w *avoids* p or w is *p-free*. An occurrence $u_0 \cdots u_{n-1}$ of p is *non-trivial* if $u_i \neq \diamond$ for all $i = 0, \ldots, n-1$. Otherwise, the occurrence is called *trivial*. We call w

non-trivially p-free if w contains no non-trivial occurrence of p. For instance, the pattern $\alpha\beta\beta\alpha$ occurs in $ab\underline{\diamond b}\ \underline{a}\ \underline{\diamond}\ \underline{bb}a$, while $\diamond babbbaaab$ avoids $\alpha\beta\beta\alpha$ (the underlined occurrence of $\alpha\beta\beta\alpha$ is trivial).

A pattern p is *k-avoidable* if there are infinitely many partial words in $A_\diamond^*$ with h holes, for any integer $h > 0$, that avoid p, where A is any alphabet of size k. Note that if there is a partial word over A with infinitely many holes that avoids p, then p is obviously k-avoidable. On the other hand, if, for some integer $h \geq 0$, p occurs in every long enough partial word in $A_\diamond^*$ with h holes, then p is *k-unavoidable* (it is also called unavoidable over A). Finally, a pattern p which is k-avoidable for some k is simply called *avoidable*, and a pattern which is k-unavoidable for every k is called *unavoidable*. The *avoidability index* of p is the smallest integer k such that p is k-avoidable, or is ∞ if p is unavoidable.

If a pattern p occurs in a pattern q, then p *divides* q, denoted by $p \mid q$. For example, $\alpha\alpha \nmid \alpha\beta\alpha$ but $\alpha\alpha \mid \alpha\beta\alpha\beta$. When both $p \mid q$ and $q \mid p$ hold, p and q are *equivalent*; for instance, $\alpha\alpha$ and $\beta\beta$ are equivalent.

3 Hole Sparsity

Results for pattern avoidance in partial words have thus far been obtained by hole insertions in selected positions of full words (the letters in selected positions are replaced by holes). In [4], for example, a binary partial word with infinitely many holes that avoids the pattern $\alpha\beta\alpha\beta\alpha$ is constructed, and has at least 80 letters between any two consecutive holes. We are interested in determining how frequently holes can appear in a word that avoids a given pattern. First, we give a precise definition of the notion of "frequency of holes."

Definition 1. *The hole sparsity of a partial word w is the smallest positive integer λ such that every factor of w of length λ contains at least one hole. In this case, we call w λ-sparse.*

For example, $ab\diamond b\diamond ac\diamond bc\diamond a$ is 3-sparse, since every factor of length three contains at least one hole and there is a factor of length two that has no holes.

For a fixed pattern p and fixed alphabet A of size k, we want to determine the smallest λ so that an infinite λ-sparse word over A avoids p.

Definition 2. *Let $p \in E^+$ be a pattern.*

- *Define the non-trivial minimum hole sparsity for p over an alphabet of size k, denoted $\chi_k(p)$, to be the smallest positive integer λ such that there exists an infinite λ-sparse word w over a k-letter alphabet that avoids all non-trivial occurrences of p. If no such integer exists, then $\chi_k(p) = \infty$.*
- *Define the minimum hole sparsity for p over an alphabet of size k, denoted $\chi_k^*(p)$, to be the smallest positive integer λ such that there exists an infinite λ-sparse word w over a k-letter alphabet that avoids all occurrences of p (including trivial occurrences). If no such integer exists, then $\chi_k^*(p) = \infty$.*

We have the following lemma regarding (non-trivial) minimum hole sparsity.

Lemma 1. *For $p, q \in E^+$ and integer $k > 0$, the following statements hold:*

1. $\chi_k(p) \geq 2$;
2. *If $k < k'$, then $\chi_k(p) \geq \chi_{k'}(p)$;*
3. *If $p \mid q$, then $\chi_k(p) \geq \chi_k(q)$;*
4. *If p is k-unavoidable, then $\chi_k(p) = \infty$.*

Statements 1–4 hold for the χ_k^* function as well.

Proof. We prove the lemma for the χ_k function. Statement 4 is trivial, while for Statement 1, we point out that if an infinite word is 1-sparse, then it consists only of holes, and therefore meets every pattern. For Statement 2, if $k < k'$, then a λ-sparse word avoiding p over the k-letter alphabet will also avoid p over the k'-letter alphabet. We prove Statement 3 similarly by noting that if $p \mid q$ then a word that avoids p also avoids q. □

Moreover, the backtracking algorithm given in Chapter 2 of [6] can be easily adapted to list all the λ-sparse words over an alphabet of size k that trivially or non-trivially avoid p. The following useful definition differs from the usual preimage definition in that, for all $y \in \varphi^{-1}(x)$, we do not require that $x = \varphi(y)$, but only that x is a factor of $\varphi(y)$.

Definition 3. *Let A and B be alphabets, let $\varphi : A^* \to B^+$, and let p be a pattern. We define the preimage of p under φ, denoted $\varphi^{-1}(p)$, to be the set of all $w \in A^*$ for which $\varphi(w)$ meets p.*

4 Non-trivial Minimum Hole Sparsity for Unary Patterns

In this section, we consider all non-trivial occurrences of α^n, where $n \geq 1$. Since α is unavoidable over a k-letter alphabet and α^n is unavoidable over the unary alphabet, $\chi_k(\alpha) = \infty$ and $\chi_1(\alpha^n) = \infty$ for all $k, n \geq 1$. The pattern α^2 is unavoidable over the binary alphabet [19], hence, $\chi_2(\alpha^2) = \infty$. Moreover, for all $k \geq 1$, $\chi_k(\alpha^2) > 3$. The latter is due to the fact that $a\diamond\diamond b$, $a\diamond b\diamond c\diamond$, $\diamond ab\diamond$ are compatible with the non-trivial squares, $(ab)^2$, $(acb)^2$ and $(ba)^2$.

Lemma 2. *For all $k \geq 4$, $\chi_k(\alpha^2) = 4$.*

Proof. Let $A = \{a, b, c, d\}$ and consider the morphisms $\rho : A^* \to A^*$ defined by $\rho(a) = ad, \rho(b) = bc, \rho(c) = ab, \rho(d) = ba$, and $\sigma : A^* \to A_\diamond^*$ defined by $\sigma(a) = dca\diamond, \sigma(b) = bca\diamond, \sigma(c) = dba\diamond, \sigma(d) = bda\diamond$. Note that the morphism ρ has been previously studied; see, for example, exercise 33(c) of 1.6 in [1] (after a permutation of letters $d \mapsto c \mapsto b \mapsto d$). We show that the 4-sparse word $\sigma(\rho^\omega(a))$ avoids non-trivial squares.

Let us first show that $\rho^\omega(a)$ is square-free. By contradiction, assume that $\rho^\omega(a)$ contains a square, and let u^2 be the smallest factor in $\rho^\omega(a)$ that is a square. It is trivial to note that $|u^2| \geq 4$. Checking all four-length factors of $\rho^\omega(a)$ we see that every such factor must contain a c or a d, and therefore u^2 must contain

either two c's or two d's. Since c's and d's only occur at odd positions in $\rho^\omega(a)$, it follows that $|u|$ must be even.

Suppose u begins with a, the case in which u starts with b is symmetrical. Note that, the second letter of u is either b or d. If it is d, then $u^2 = adu'adu'$, for some $u' \in A^*$. According to the definition of ρ and the fact that this morphism is prolongable, there exists a factor $x \in A^*$ of $\rho^\omega(a)$, such that $u^2 = \rho(x)^2$. This is a contradiction with the minimality of u. If the second letter of u is b, then ab is an image of either c or db (since $\rho(db) = babc$). If the preimage is c, we reach another contradiction in the same manner. If the preimage is db, it follows that $bu^2 = babu'babu'b$ is a factor of $\rho^\omega(a)$, for some $u' \in A^*$. Once more we reach a contradiction. Now let us suppose u begins with c, the case when u starts with d is symmetrical. Since b always preceeds c, we have $bu^2 = bcu'bcu'a$ is a factor of $\rho^\omega(a)$, for some $u' \in A^*$. A contradiction is reached similarly to the previous cases, and we conclude that $\rho^\omega(a)$ is square-free.

Assume now, to the contrary, that $\sigma(\rho^\omega(a))$ contains a factor uv where $u \uparrow v$ with $|u| = |v|$. Furthermore, every fourth symbol of $\sigma(\rho^\omega(a))$ is an a. If $|u| \leq 3$, then the only combinations of images of σ that contain squares are $\diamond dba \diamond b$ and $\diamond bda \diamond d$, but none of their preimages, xcb, xcd, xda or xdc, occur in $\rho^\omega(a)$. Thus, $|u| \geq 4$, and we show that $|u|$ is divisible by 4. Since $|u| \geq 4$, it must be that both u and v contain a's. If the a's occur in corresponding positions, then $|u|$ is divisible by 4. If not, then without loss of generality we assume that an a in u corresponds to a $\diamond$ in v. But then the a preceding the hole in v corresponds to the symbol that precedes the a in u, which is neither a nor $\diamond$, a contradiction. Note that if u begins with a, since v begins with a hole, it must be that u ends in a, and this correspondz to a hole at the end of v, a contradiction according to the previous case. It follows that $u \uparrow v$ implies $u = v$, since $\diamond$'s in u occur in the same positions as those in v. Hence, we may refer to uv as u^2.

Let us look at the position of the first hole in u. If $u(3) = \diamond$, then u starts with one of the images of σ, and since $|u|$ is divisible by four, it follows that for some $x \in A^*$ we have $u^2 = \sigma(xx)$, which is a contradiction with the fact that $\rho^\omega(a)$ is square-free. If $u(0) = \diamond$, we consider the square that begins at $u(1)$ and ends with the $\diamond$ that follows u^2, which leads to a contradiction as shown above. If $u(1) = \diamond$ then u begins with $a\diamond$ and u^2 must be followed by $a\diamond$. Considering the square that starts at $u(2)$ a contradiction follows as shown above. Finally, assume $u(2) = \diamond$. If u starts with b or d, then we know the letter that precedes u, and we find a new square starting with that letter and obtain a contradiction. If u starts with c, then $u^2 = ca\diamond u'zca\diamond u''z$, where $u' = u'' \in A_\diamond^*$ and $z \in \{b, d\}$. Suppose u^2 is preceded by d, the case for b is similar. In order to avoid a contradiction similar to the previous cases, we have $z = b$. Since the preimage of $dca\diamond$ under σ is a, and, in $\rho^\omega(a)$, a is always followed by b or d, it follows that the first letter of u' is b. However, by similar reasoning the first letter in u'' is a d, a contradiction since the latest d corresponds to the b in the first copy of u. □

Next we consider $\chi_3(\alpha^2)$. The backtracking algorithm gives us the bound of $\chi_3(\alpha^2) \geq 7$. We show that this bound is tight.

Lemma 3. *The equality* $\chi_3(\alpha^2) = 7$ *holds.*

Proof. Let $A = \{a, b, c, d\}$ and $B = \{a, b, c\}$, and the morphism ρ defined as in Lemma 2. Define $\pi : A^* \to B_\diamond^*$ by $\pi(a) = abcbac\diamond b\diamond acbabc\diamond, \pi(b) = bacabc\diamond a\diamond bcabac\diamond, \pi(c) = abcacb\diamond$, and $\pi(d) = bacbca\diamond$. We show that $\pi(\rho^\omega(a))$ avoids non-trivial squares. Suppose to the contrary that $\Pi = \pi(\rho^\omega(a))$ contains a factor uv where $u \uparrow v$. Note that, every length two factor of $\rho^\omega(a)$ has an image under π of length at least 23. Moreover, all squares of length at most 24 are contained in the image of a factor of length three of $\rho^\omega(a)$. There are ten such factors $aba, abc, adb, bab, bad, bca, cab, cad, dba$ and dbc, and, by exhaustive checking, we conclude that their images under π are square-free. Thus, it must be that $|uv| > 25$.

Let us now show that $u = v$. In particular, we show that there are no two non-identical factors of length nine that are compatible. Looking at all 54 length nine factors of Π obtained from the ten factors of $\rho^\omega(a)$ described above, we get that no two are compatible. Hence, all compatible factors of Π of length nine or greater must be equal. Since $|u| = |v| > 12$, we must have $u = v$.

Looking again at the position of the first hole in u and using an approach similar to the one in the proof of Lemma 2, the conclusion follows. □

Let us now move on to α^3 and first show that $\chi_2(\alpha^3) = 3$. The backtracking algorithm gives the lower bound $\chi_2(\alpha^3) \geq 3$.

Lemma 4. *The equality $\chi_2(\alpha^3) = 3$ holds.*

Proof. Let $A = \{a, b, c\}$ and $B = \{a, b\}$, and define the morphisms $\delta : A^* \to A^*$ by $\delta(a) = ab$, $\delta(b) = bc$ and $\delta(c) = ab$, and $\upsilon : A^* \to B_\diamond^*$ by $\upsilon(a) = aa\diamond$, $\upsilon(b) = ab\diamond$ and $\upsilon(c) = bb\diamond$. Note that replacing each c in δ with a yields the Thue-Morse morphism. It is well-known that the Thue-Morse word avoids the patterns α^3 and $\alpha\beta\alpha\beta\alpha$ [9]. Thus, $\delta^\omega(a)$ also avoids these patterns.

We show that $\upsilon(\delta^\omega(a))$ avoids non-trivial cubes. Suppose towards a contradiction, that there exists a non-trivial cube $u_1u_2u_3$ in $\upsilon(\delta^\omega(a))$, where $u_1, u_2, u_3 \subset u$ for some $u \in B_\diamond^*$ and $u_i \neq \diamond$ for all i. We first look at the possible starting letters of u_1, u_2 and u_3 in order to show $|u| \equiv 0 \bmod 3$.

If $|u| \equiv 1$ or $2 \bmod 3$ and $|u| > 3$, then, because of the compatibility, the u_i's must begin one with $xy\diamond$, one with $x\diamond z$, and one with $\diamond yz$, where $x, y, z \in B$. If $xy = aa$, then the factor $a\diamond z$ implies $z = a$. The factor $aa\diamond$ is followed by a, while $a\diamond a$ is followed by b, yielding a contradiction. If $xy = ab$, then the factor $a\diamond z$ implies $z = a$, but the factor $\diamond bz$ implies $z = b$, a contradiction. Finally, if $xy = bb$, then the factor $\diamond bz$ implies $z = b$. The factor $b\diamond b$ is followed by b, while $bb\diamond$ is followed by a, a contradiction. Since $|u| = 1$ implies the occurrence of a trivial cube, this leaves $|u| = 2$ in the case when $|u| \not\equiv 0 \bmod 3$. In this case, u^3 is contained in the image of a factor in $\delta^\omega(a)$ of length three. It is sufficient to compute all such factors and check their images to see that none of their images under υ contains a cube.

For $|u| \equiv 0 \bmod 3$, we note that $u = u_1 = u_2 = u_3$. The conclusion follows similarly to the proof of Lemma 2, based on the position of the first $\diamond$ in u. □

Lemma 5. *For all $k, n \geq 3$, $\chi_k(\alpha^n) = 2$.*

Proof. From Lemma 1 we have $\chi_3(\alpha^3) \geq 2$, so it suffices to show that $\chi_3(\alpha^3) = 2$. Let $A = \{a, b, c\}$ and consider $\varphi^\omega(a)$, where $\varphi : A_\diamond^* \to A_\diamond^*$ is given by $\varphi(a) = a\diamond b\diamond a\diamond c$, $\varphi(b) = a\diamond b$, $\varphi(c) = a\diamond c$ and $\varphi(\diamond) = \diamond$. Observe that $\varphi^\omega(a)$ alternates letters and holes, and hence is 2-sparse. Suppose towards a contradiction, that $\varphi^\omega(a)$ has a factor $u_1u_2u_3$ with $u_1, u_2, u_3 \subset u$, for some $u \in A_\diamond^*$. If $|u|$ is even, since holes must occur in the same positions in the u_i's, it follows that $u = u_1 = u_2 = u_3$. Moreover, a appears in $\varphi^\omega(a)$ exactly once every four symbols. Since $|u^3| \geq 6$ and u^3 is a cube, u^3 contains an a in each of the three copies of u. Thus, $|u| \equiv 0 \bmod 4$.

Assume that u^3 begins with an a. The other cases are similar since either the second symbol in u^3 is an a, there is an a immediately to the left of u^3, or there is an a before the symbol that precedes u^3. In all these cases u^3 shifts so that it starts with a. Break u^3 into factors of length four, each of which is either $s = a\diamond b\diamond$ or $t = a\diamond c\diamond$. We can therefore rewrite $\varphi^\omega(a)$ in terms of s and t, such that the cubes in $\varphi^\omega(a)$ correspond to cubes in the new word. Note that $\varphi(s) = a\diamond b\diamond a\diamond c\diamond a\diamond b\diamond = sts$ and $\varphi(t) = a\diamond b\diamond a\diamond c\diamond a\diamond c\diamond = stt$, and that this new word is equivalent to $\varphi'^\omega(s)$, where $\varphi' : \{s, t\}^* \to \{s, t\}^*$ is given by $\varphi'(s) = sts$ and $\varphi'(t) = stt$. We reach the desired contradiction using a result of Richomme and Wlazinski, who show in [15] that a morphism φ'' on $\{s, t\}^*$ avoids cubes if and only if $\varphi''(sstttststtstsstststsstt)$ is cube-free. It is straightforward to compute $\varphi'(sstttststtstsstststsstt)$ and check that it does not have any cubes.

If $n = |u|$ is odd, then $u_1 \uparrow u_3$ implies $u_1 = u_3$. Hence, we refer to $u_1u_2u_3$ as xyx, where $x \uparrow y$. Moreover, we may suppose that $x(0) = \diamond$, since otherwise, there is another cube, of equal length, starting at $x(1) = \diamond$. Since $x(0) = \diamond$, and $|x|$ is odd, we have $x = \diamond x'\diamond$ with $x' \in A_\diamond^*$. Since we can break $\varphi^\omega(a)$ into factors of length four that are either $a\diamond b\diamond$ or $a\diamond c\diamond$, by induction, we deduce that between any two arbitrary identical letters there are oddly-many letters. Counting the number of letters between the first letter in the first occurrence of x and the one in the second occurrence, we get that this number is even. This is a contradiction with the previous remark, hence, the conclusion follows. □

Next, we show that for $\alpha^n, n \geq 4$, a binary alphabet is enough for constructing an infinite word having a hole every two symbols.

Lemma 6. *For all $n \geq 4$, $\chi_2(\alpha^n) = 2$.*

Proof. Let $A = \{a, b\}$ and define $\mu : A_\diamond^* \to A_\diamond^*$ to be $\mu(a) = a\diamond b$, $\mu(b) = a\diamond a$ and $\mu(\diamond) = \diamond$. We show that $\mu^\omega(a)$ is free of fourth powers. Assume that $\mu^\omega(a)$ has a factor $u_1u_2u_3u_4$ such that $u_1, u_2, u_3, u_4 \subset u$, for some $u \in A_\diamond^*$.

If $|u|$ is even, since the holes align, we have $u = u_1 = u_2 = u_3 = u_4$. Assume that u^4 begins with a letter. If it is not the case, then there is a fourth power of equal length that starts at the second symbol of u^4, which must be a letter. Break u^4 into factors of length two, each of which is either $s = a\diamond$ or $t = b\diamond$. We can therefore write $\mu^\omega(a)$ in terms of s and t, and the fourth powers in $\mu^\omega(a)$ correspond to fourth powers in the new word. Since $\mu(s) = a\diamond b\diamond = st$ and $\mu(t) = a\diamond a\diamond = ss$, the new word is equivalent to $\mu'^\omega(s)$, where $\mu' : \{s, t\}^* \to \{s, t\}^*$ is given by $\mu'(s) = st$ and $\mu'(t) = ss$. We now show that $\mu'^\omega(s)$ is 4th powers free.

Assume $\mu'^{\omega}(s)$ has a factor v^4 and that v^4 is the smallest fourth power in $\mu'^{\omega}(s)$. Since, from the definition of μ', between two consecutive t's there are either one or three s's, by induction, there are an odd number of letters between any two t's. Moreover, $v^4 \geq 4$ and v^4 must contain a t and, hence, each copy of v must contain a t. Thus, $|v|$ must be even. Let $v' \in \{s,t\}^*$. If $v = stv'$, then the preimage of v^4 under μ' is unique and contains a fourth power. Since μ' is prolongable, this contradicts the minimality of v. If v starts with t, then v is preceded by s, and there exists again a fourth power which is covered above. If $v = sstv'$, then v is preceded by ts, and there is also a fourth power $(tssstv')^4$, covered above. Finally, if $v = ssstv'$, then v is preceded by t, and there is a fourth power $(tssstv')^4$ in $\mu'^{\omega}(s)$, which is covered above.

Next, suppose that $n = |u|$ is odd. Note that, $u_1 \uparrow u_3$ implies $u_1 = u_3$ and $u_2 \uparrow u_4$ implies $u_2 = u_4$, and a b occurs at least every 8 symbols. Since $|u^4| \geq 8$, there is at least one b in u^4. Assume without loss of generality that b occurs in u_1. Then, there is a corresponding b in u_3, and between the two b's there are $2n-1$ symbols. The remainder of the proof is identical to that of Lemma 5. □

5 Minimum Hole Sparsity for Unary Patterns

Note that $\chi_k^*(\alpha) = \chi_1^*(\alpha^n) = \infty$ for all $k, n \geq 1$. Since factors of the form $a\diamond$ or $\diamond a$ appear in all infinite partial words having holes, $\chi_k^*(\alpha^2) = \infty$ for all $k \geq 1$.

Looking at trivial cubes, any 2-sparse word meets the pattern α^3, since any letter a is preceded and followed by a hole. Thus, for all $k \geq 1$, we have the lower bound $\chi_k^*(\alpha^3) \geq 3$. For $k \geq 3$, we show that the bound is tight.

Lemma 7. *For all $k \geq 3$, $\chi_k^*(\alpha^3) = 3$.*

Proof. Let $A = \{a,b\}$ and $B = \{a,b,c\}$, and define the morphisms $\xi : A^* \to A^*$ by $\xi(a) = ab$ and $\xi(b) = ba$, and $\tau : B^* \to B_\diamond^*$ by $\tau(a) = ab\diamond$ and $\tau(b) = ac\diamond$.

We show that $\tau(\xi^\omega(a))$ avoids cubes. Suppose to the contrary that $\tau(\xi^\omega(a))$ has a factor $u_1u_2u_3$ where $u_1, u_2, u_3 \subset u$, for some $u \in A_\diamond^*$. If $|u| \equiv 1$ or 2 mod 3 and $|u| > 3$, then, again due to compatibility, the u_i's must begin one with $xy\diamond$, one with $x\diamond z$, and one with $\diamond yz$, where $x, y, z \in B$. Furthermore, since $\diamond$ is always followed by a, we must have $x = y = z = a$, which is absurd. If $|u| \leq 2$, then u^3 is contained in the image of a factor in $\xi^\omega(a)$ of length 3. It is sufficient to compute all such factors and check their images under τ for cubes. None of the images of these under τ contains a cube.

For $|u| \equiv 0$ mod 3, we note that $u = u_1 = u_2 = u_3$.

If $u = au'$ for $u' \in B_\diamond^*$, then the preimage of u^3 in $\xi^\omega(a)$ contains a cube, a contradiction since ξ is the cube-free Thue-Morse morphism. If, instead, $u = xu'$, $x \in \{b, c, \diamond\}$, then we can easily find another perfect cube that begins one symbol to the left, in the case $x = b$ or $x = c$, or to the right, in the case $x = \diamond$, of $u(0)$. The new perfect cube has the form $(au')^3$, which leads to a contradiction as shown above. □

We complete the computation of $\chi_k^*(\alpha^3)$, $k \geq 1$, with the following lemma.

Lemma 8. *The equality* $\chi_2^*(\alpha^3) = 7$ *holds.*

Proof. The backtracking algorithm gives the lower bound of 7. Let $A = \{a, b\}$, $\xi : A^* \to A^*$ the Thue-Morse morphism from the proof of Lemma 7, and define $\zeta : A^* \to A_\diamond^*$ by $\zeta(a) = babaab\diamond abbaba\diamond$ and $\zeta(b) = baabba\diamond$.

We show that $\zeta(\xi^\omega(a))$ avoids cubes. Suppose towards a contradiction that $\zeta(\xi^\omega(a))$ contains a factor $u_1u_2u_3$, where $u_1, u_2, u_3 \subset u$ for $u \in A_\diamond^*$. Since every factor v of $\xi^\omega(a)$ with $|v| \geq 7$ has an image $\zeta(v)$ such that $|\zeta(v)| \geq 70$, every length 63 factor of $\zeta(\xi^\omega(a))$ is contained in the image of a factor w of length eight of $\xi^\omega(a)$. There are 22 possible such factors and by exhaustive checking we find out that none contains cubes. Thus, we have $|u| \geq 21$.

The same as in the proof of Lemma 3 we show that in $\zeta(\xi^\omega(a))$ there are no two non-identical factors of length 21 that are compatible. In particular, we check that each of the 51 factors of length 21 of $\zeta(\xi^\omega(a))$ is not compatible with any other length 21 factor. It follows that any two compatible factors of $\zeta(\xi^\omega(a))$ of length 21 or greater must be equal. Since $|u| \geq 21$, we have $u = u_1 = u_2 = u_3$.

Since u^3 is sufficiently long, it contains the factor $\zeta(b) = baabba\diamond$. If $\zeta(b)$ is a factor of u then $u = u'\zeta(b)u''$ with $u''u' = \zeta(w)$, for $w \in A^*$. Hence, the Thue-Morse word, contains the factor $bwbwb$, a contradiction since this word is overlap-free. If $\zeta(b)$ is not a factor of u, then $u = u'u''u'''$ with $u'''u' = \zeta(b)$ and $u'' = \zeta(w)$, for $w \in A^*$. This implies that the Thue-Morse word contains the factor $wbwbw$, again a contradiction. □

The following two lemmas settle α^4.

Lemma 9. *The equality* $\chi_2^*(\alpha^4) = 3$ *holds.*

Proof. The backtracking algorithm provides the lower bound. Let $A = \{a, b\}$, $\xi : A^* \to A^*$ the Thue-Morse morphism from the proof of Lemma 7, and define $\kappa : A^* \to A_\diamond^*$ by $\kappa(a) = ab\diamond ab\diamond ba\diamond ba\diamond$ and $\kappa(b) = ab\diamond ab\diamond ab\diamond ba\diamond ba\diamond$.

We show that $\kappa(\xi^\omega(a))$ avoids fourth powers. Suppose to the contrary that $\kappa(\xi^\omega(a))$ contains a factor $u_1u_2u_3u_4$ with $u_1, u_2, u_3, u_4 \subset u$, for some $u \in A_\diamond^*$. Every factor v of $\xi^\omega(a)$, $|v| \geq 2$, has an image $\kappa(v)$ such that $|\kappa(v)| \geq 24$. Hence, every length 24 factor of $\kappa(\xi^\omega(a))$ is contained in the image of a factor of length 3 of $\xi^\omega(a)$. Since none of the images of these six factors contains fourth powers, we conclude that $|u| \geq 7$.

Moreover, suppose that $|u| \not\equiv 0 \bmod 3$. Since every third symbol is $\diamond$, at least one of the u_i's must begin with $xy\diamond$, where $x, y \in A$ and $x \neq y$. To maintain compatibility, another u_i must begin with $\diamond yx$, while the third must begin with $x\diamond x$. Furthermore, the factor $x\diamond x$ must be followed by $y\diamond$, while the factor $xy\diamond$ must be followed by $yx\diamond$ and the factor $\diamond yx$ must be followed by $\diamond xy$. We conclude that one of the u_i's should start with $x\diamond xy\diamond y$, but since neither $a\diamond ab\diamond b$ nor $b\diamond ba\diamond a$ appear in $\kappa(\xi^\omega(a))$, this is a contradiction. Therefore, we have $7 \leq |u| \equiv 0 \bmod 3$.

Next, if $|u| \equiv 0 \bmod 3$, then $u = u_1 = u_2 = u_3 = u_4$, and since $|u^4| \geq 28$, it must have the factor $ba\diamond ba\diamond$ at least once. Because u^4 has four occurrences of u, the factor $ba\diamond ba\diamond$ must appear at least three times. Taking the preimages of the last three (identical) factors that end with $ba\diamond ba\diamond$, we obtain a cube in $\xi^\omega(a)$, a contradiction since the Thue-Morse word $\xi^\omega(a)$ is cube-free. □

k	α	α^2	α^3	α^4	α^5	$\cdots$
1	∞	∞	∞	∞	∞	$\cdots$
2	∞	∞	3	2	2	$\cdots$
3	∞	7	2	2	2	$\cdots$
4	∞	4	2	2	2	$\cdots$
5	∞	4	2	2	2	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Fig. 1. $\chi_k(\alpha^n)$

k	α	α^2	α^3	α^4	α^5	α^6	α^7	$\cdots$
1	∞	∞	∞	∞	∞	∞	∞	$\cdots$
2	∞	∞	7	3	3	2	2	$\cdots$
3	∞	∞	3	2	2	2	2	$\cdots$
4	∞	∞	3	2	2	2	2	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Fig. 2. $\chi_k^*(\alpha^n)$

Lemma 10. *For all $k \geq 3$, $\chi_k^*(\alpha^4) = 2$.*

Proof. Lemma 1 gives the lower bound. Let $A = \{a, b, c\}$ and define a new function $\mu' : A_\diamond^* \to A_\diamond^*$ as follows $\mu'(a) = a\diamond b$, $\mu'(b) = a\diamond c$, $\mu'(c) = a\diamond b$ and $\mu'(\diamond) = \diamond$. Furthermore, note that the infinite word $\mu'^\omega(a)$ is simply a copy $\mu^\omega(a)$ from Lemma 6, in which some a's have been replaced with c's. Since a non-trivial fourth power in $\mu'^\omega(a)$ implies the presence of a non-trivial fourth power in $\mu^\omega(a)$, which is impossible, we have that $\mu'^\omega(a)$ is non-trivially 4th power free. It is easy to check that $\mu'^\omega(a)$ avoids trivial fourth powers as well, since no two consecutive letters of $\mu'^\omega(a)$ are the same. □

The next two lemmas consider α^n, where $n \geq 5$.

Lemma 11. *The equality $\chi_2^*(\alpha^5) = 3$ holds.*

Lemma 12. *For all $n \geq 6$ and $k \geq 2$, $\chi_k^*(\alpha^n) = 2$.*

Proof. We show that the Thue-Morse word $\xi^\omega(a)$ from Lemma 7 with a hole inserted between every two letters is 6th power free. Suppose to the contrary that the new word w contains a factor $u_0 \cdots u_5$, where $u_i \subset u$ for some $u \in \{a, b\}_\diamond^*$ and $i \in \{0, \ldots, 5\}$. Moreover, assume that $u_0 \cdots u_5$ begins with a letter, for if it begins with a $\diamond$ then we consider the sixth power in w that begins with the second symbol of $u_0 \cdots u_5$ and ends with the hole that follows $u_0 \cdots u_5$.

Let $v_0 = u_0u_1$, $v_1 = u_2u_3$, $v_2 = u_4u_5$, and $v = u^2$ so that $v_0, v_1, v_2 \subset v$. We have that $|v| = 2|u|$ is even, and so $v_0 = v_1 = v_2$ since the $\diamond$'s occur in corresponding positions in all v's. We break $v_1v_2v_3$ into factors of length two, each of which are $a\diamond$ or $b\diamond$. Since $v_1 = v_2 = v_3$, removing the holes from each factor of length two of w preserves the cube in the resulting word. However, this word is $\xi^\omega(a)$, which is known to avoid cubes. The conclusion follows. □

6 Conclusion

Using Lemma 1 and the results from Sections 4 and 5, we conclude with the following theorem which gives $\chi_k(\alpha^n)$ and $\chi_k^*(\alpha^n)$ for all $k, n \geq 1$.

Theorem 1. *The values of $\chi_k(\alpha^n)$ are given in Figure 1, while the values of $\chi_k^*(\alpha^n)$ are given in Figure 2.*

References

1. Allouche, J.P., Shallit, J.: Automatic Sequences: Theory, Applications, Generalizations. Cambridge University Press, Cambridge (2003)
2. Bean, D.R., Ehrenfeucht, A., McNulty, G.: Avoidable patterns in strings of symbols. Pacific Journal of Mathematics 85, 261–294 (1979)
3. Blanchet-Sadri, F.: Algorithmic Combinatorics on Partial Words. Chapman & Hall/CRC Press, Boca Raton (2008)
4. Blanchet-Sadri, F., Mercaş, R., Simmons, S., Weissenstein, E.: Avoidable binary patterns in partial words. Acta Informatica 48, 25–41 (2011)
5. Cassaigne, J.: Unavoidable binary patterns. Acta Informatica 30, 385–395 (1993)
6. Cassaigne, J.: Motifs évitables et régularités dans les mots. PhD thesis, Paris VI (1994)
7. Crochemore, M., Ilie, L., Rytter, W.: Repetitions in strings: Algorithms and combinatorics. Theoretical Computer Science 410, 5227–5235 (2009)
8. Currie, J.D.: Pattern avoidance: themes and variations. Theoretical Computer Science 339 (2005)
9. Lothaire, M.: Combinatorics on Words. Cambridge University Press, Cambridge (1997)
10. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2002)
11. Main, M.G., Bucher, W., Haussler, D.: Applications of an infinite squarefree co-CFL. Theoretical Computer Science 49, 113–119 (1987)
12. Manea, F., Mercaş, R.: Freeness of partial words. Theoretical Computer Science 389, 265–277 (2007)
13. Ochem, P.: A generator of morphisms for infinite words. RAIRO-Theoretical Informatics and Applications 40, 427–441 (2006)
14. Ochem, P., Rampersad, N., Shallit, J.: Avoiding aproximate squares. International Journal of Foundations of Computer Science 19, 633–648 (2008)
15. Richomme, G., Wlazinski, F.: Some results on k-power-free morphisms. Theoretical Computer Science 273, 119–142 (2002)
16. Roth, P.: Every binary pattern of length six is avoidable on the two-letter alphabet. Acta Informatica 29, 95–106 (1992)
17. Schmidt, U.: Motifs inévitables dans les mots. Technical Report LITP 86–63 Thèse, Paris VI (1986)
18. Schmidt, U.: Avoidable patterns on two letters. Theoretical Computer Science 63, 1–17 (1989)
19. Thue, A.: Über unendliche Zeichenreihen. Norske Vid. Selsk. Skr. I, Mat. Nat. Kl. Christiana 7, 1–22 (1906)
20. Thue, A.: Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen. Norske Vid. Selsk. Skr. I, Mat. Nat. Kl. Christiana 1, 1–67 (1912)
21. Zimin, A.I.: Blocking sets of terms. Mathematics of the USSR-Sbornik 47, 353–364 (1984)

Characterizing Compressibility of Disjoint Subgraphs with NLC Grammars*

Robert Brijder[1] and Hendrik Blockeel[1,2]

[1] Leiden Institute of Advanced Computer Science,
Universiteit Leiden, The Netherlands
rbrijder@liacs.nl
[2] Department of Computer Science, Katholieke Universiteit Leuven, Belgium
hendrik.blockeel@cs.kuleuven.be

Abstract. We consider compression of a given set $\mathcal{S}$ of isomorphic and disjoint subgraphs of a graph G using node labelled controlled (NLC) graph grammars. Given $\mathcal{S}$ and G, we characterize whether or not there exists a NLC graph grammar consisting of exactly one rule such that (1) each of the subgraphs $\mathcal{S}$ in G are compressed (i.e., replaced by a nonterminal) in the (unique) initial graph I, and (2) the set of generated terminal graphs is the singleton $\{G\}$.

1 Introduction

Graph grammars define languages over graphs, in the same way that string or tree grammars define languages over strings (trees). Several types of graph grammars have been proposed and studied (e.g., [8,1]). A well-studied and well-understood type are the NLC grammars (Node Labeled Controlled grammars) [8]. An NLC grammar consists of a set of node rewriting rules that indicate how, in the process of generating graphs from an initial graph using the grammar, a node with a particular label can be rewritten into a subgraph, and how this subgraph is connected to the neighbors of the node being rewritten. These so-called connection rules are in an important difference with string or tree grammars, which do not need such rules.

While NLC grammars (and the broader class of (e,d)-NCE grammars that they belong to) have been studied in detail regarding the properties of the languages they can define, the complexity of parsing their elements, etc., there has been little research on the induction of such grammars from example graphs. This is not specific for NLC grammars: induction of graph grammars has in general not received much interest. This is somewhat remarkable because the learning of graph languages from example graphs is generally considered an important topic in machine learning: it has been used, for instance, to classify molecules [6], analyse network structures [12], recognize objects in images [13], etc. The induction of graph grammars has the additional advantage that the grammar

* This research is supported by the Netherlands Organization for Scientific Research (NWO), project "Annotated graph mining".

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 167–178, 2011.

rules define transformations of graphs and as such can be used to describe the dynamic behavior of graphs, which is of interest in the context of, for instance, social network analysis.

Several approaches to the induction of graph grammars have been proposed, with significant practical success; however, there is little theoretical understanding of the learnability of certain types of graph grammars. In the context of NLC grammars, some initial work has been done that describes how grammar rules can be learned from example graphs. In this paper, we continue that thread of research.

Previous work [4] showed how, given a graph G (or a set of graphs — but such a set can always be considered a single graph with multiple connected components, so there is no loss of generality in considering a single graph), one can find a single NLC grammar rule r and a graph I that is as small as possible (following the "maximum compression" or "minimum description length" (MDL) principle that is commonly used in machine learning), with the property that I uniquely determines G through repeated application of r. This work was limited by the fact that the subgraphs generated by the rule r cannot "touch", that is, there cannot be edges connecting two such subgraphs. In [3], it was shown that when the subgraphs can touch, this may lead to non-confluency: the order in which nodes are rewritten can influence the outcome of the rewriting process (thus, I no longer uniquely determines G unless this order is fixed). *In this paper, we characterize the exact conditions under which the NLC grammar rule will be confluent.*

This result is useful for two reasons. From the point of view of graph compression, it makes a better compression algorithm possible because the conditions under which subgraphs can be compressed into a single node can be relaxed. From the point of view of graph grammar induction (and its applications in machine learning), it allows for a broader class of grammar rules to be learned.

2 Related Work

As mentioned in the Introduction, there has been little research on graph grammar induction. An important line of work in this area was started by Cook and Holder with their work on Subdue [5], an algorithm for learning from graphs that, in several variants, led to classification, clustering, and compression of graphs, and later on to induction of graph grammars [10,11]. This work is mostly motivated by practical applications, rather than theory on grammar induction or graph grammars, and as such these grammars sometimes lack desirable properties. For instance, node rewriting rules do not always indicate how the new graph should be connected to the neighborhood of the node being substituted; as a result, graph compression is not lossless, and the graph grammar lacks a certain expressive power that other grammars have. Also, potential problems with non-confluency, overlapping or touching subgraphs, etc., are not studied.

Another line of work is the induction of probabilistic graph grammars in [7,14]. This work builds mostly on methods for constructing probabilistic grammars. It does not focus on issues such as lossless compression or confluency, which are not very relevant in that context.

In [9] the potential of edge replacement grammars for machine learning is discussed, but we are not aware of further work in that direction.

The most closely related work is [4] and [2,3] on the induction of NLC grammars; this paper builds directly on it. As said in the introduction, the work by [4] had certain limitations, and the possible effects of lifting these limitations was studied by [3]. In this paper, we characterize under what conditions the mentioned limitations can be lifted without undesirable consequences. Before we can go into detail about this, we need to introduce some terminology and background.

3 Notation and Terminology

We consider simple graphs $G = (V, E)$, where V is a finite set of nodes and $E \subseteq \{\{x, y\} \mid x, y \in V, x \neq y\}$ is the set of edges — hence no loops or parallel edges are allowed. We denote $V(G) = V$ and $E(G) = E$. For $S \subseteq V$, the induced subgraph of G is the graph (S, E') where $E' \subseteq E$ and for each $e \in E$ we have $e \in E'$ iff $e \subseteq S$. We consider only induced subgraphs, and therefore we will write "subgraph" instead of induced subgraph. The neighborhood of $S \subseteq V$ in G, denoted by $N_G(S)$, is $\{v \in V \backslash S \mid \{s, v\} \in E \text{ for some } s \in S\}$. If $S = \{x\}$ is a singleton, then we also write $N_G(x) = N_G(S)$. A labelled graph is a triple $G = (V, E, l)$ where (V, E) is a simple graph and $l : V \to L$ is a node labelling function, where L is a finite set of labels. We write $l(G) = l(V) = \{l(v) \mid v \in V\}$. As usual, graphs are considered isomorphic if they are identical modulo the identity of the nodes. It is important to realize that for labelled graphs, nodes identified by an isomorphism have identical labels. In graphical depictions of labelled graphs we always represent the nodes by their labels. As we consider solely labelled graphs from now on, we will often write simply *graph* to denoted labelled graphs.

Subgraphs S_1 and S_2 are called *disjoint* (or *non-overlapping*) if $V(S_1)$ and $V(S_2)$ are disjoint. They are called *touching* if they are disjoint and there is an edge $e \in E(G)$ with one node in S_1 and the other in S_2.

4 NLC Graph Grammars

In this section we briefly recall the notions and definitions concerning NLC grammars used in this paper, and refer to [8] for a gentle and more detailed introduction to these grammars.

A NLC graph grammar is an ordered 5-tuple $Q = (L, \bar{L}, I, P, E)$, where L is a finite set of *node labels*, where the elements of $\bar{L} \subseteq L$ ($L \setminus \bar{L}$, resp.) are called *terminal* (*nonterminal*, resp.) node labels, I is a labelled graph with the nodes labelled by L called the *initial graph*, and $E \subseteq L^2$ (where $L^2 = L \times L$) is called an *embedding relation*. Finally, P is a set of tuples (N, S), called *productions*, where $N \in L \setminus \bar{L}$ and S is a labelled graph with nodes labelled by L. A production (N, S) is also denoted by $N \to S$.

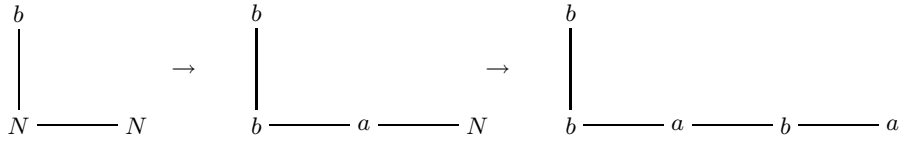


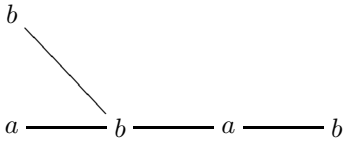
Fig. 1. The derivation of graph G (right-hand side) from I (left-hand side) in Example 1 using the derivation rule $p = N \to a - b$ and embedding relation $E = \{(b, a), (b, b), (a, N)\}$

The semantics of an NLC graph grammar Q are as follows. Let G be a graph containing a node x labelled by a nonterminal N. Then a production p of the form $N \to S$ is applicable to (defined on) G. The result after applying p to G on x is another graph G'. The graph G' is obtained from G by removing node x (and all edges adjacent to x) and replacing it by (a copy of) S. Moreover, edges are created between the nodes of S and of $N_G(x)$ according to the embedding relation E: for $y \in V(S)$ and $z \in N_G(x)$, we have $\{y, z\} \in E(G')$ iff $(l(y), l(z)) \in E$ (where l is the labelling function of G'). Now, the set of graphs with only terminal nodes obtainable from the initial graph I by iteratively applying productions from P is the *language* $L(Q)$ of Q.

In this paper, we consider NLC grammars containing exactly one production $N \to S$, i.e., $|P| = 1$. Therefore we (may) assume without loss of generality that $L \setminus \bar{L} = \{N\}$. Hence we specify Q by the tuple $(L, I, (N, S), E)$. Also, we may assume without loss of generality that the node labels of S do not contain N. Indeed, if the node labels of S contain N, and assuming I contains a node labelled by N, then $L(Q) = \varnothing$ as no graph with only terminal labels can be generated.

Example 1. Let G be the graph on the left-hand side of Figure 1. Let $L = \{a, b, N\}$ be the set of labels (N is the nonterminal), $E = \{(b, a), (b, b), (a, N)\}$ be the embedding, and $p = N \to S$ be the production, where S is the graph a —— b . Note that formally we have only defined S up to isomorphism, however as we have seen this is not an objection.

Figure 1 now depicts a derivation from an initial graph I where first production p is applied to the node labelled by N on the left-hand side, and then p is applied on the remaining node labelled by N. The obtained graph G (on the right-hand side of the figure) has only terminal nodes. The derivation from I by applying p in the other order on the nonterminal nodes obtains the following graph.



This example will be a running example in this paper.

5 Compatibility and Confluency

In this section we recall the basic notions of compatibility and confluency, and recall the results from [4] that are necessary for this paper.

We fix some graph G and some graph S. We let $\mathcal{S}$ be the set of subgraphs of G isomorphic to S. We let $\bar{L} = l(G)$ and, as defined in the previous section $L = \bar{L} \cup \{N\}$ with N the nonterminal node label.

First we recall the notion of compatibility.

Definition 2. Let G be a graph, $\mathcal{S}$ be a set of subgraphs of G isomorphic to a graph S, and $C = (S_1, S_2, \ldots, S_n)$ be a linear ordering of $\mathcal{S}$. We say that $E \subseteq L \times L$ is *compatible* with C (in G) if there are graphs $G_0, \ldots, G_n$ such that $G_n = G$ and for each $i \in \{1, \ldots, n\}$, G_i is obtained from G_{i-1} by producing S_i using NLC grammar $Q = (L, G_0, (N, S), E)$.

In case $\mathcal{S} = \{S\}$ is a singleton, we also say that E is compatible with S instead of (S).

Since the embedding of the new subgraph in the old graph is determined by node labels, it is impossible that two nodes with the same label in the subgraph are connected to the neighborhood of the subgraph in a different way. This is formally expressed as follows.

Lemma 3. *Let G be a graph, and $S_1, \ldots, S_n$ subgraphs of G. If $E \subseteq L^2$ is compatible with $(S_1, \ldots, S_n)$, then for any $i \in \{1, \ldots, n\}$ and $x, y \in V(S_i)$ with $l(x) = l(y)$, we have $N_G(x) \setminus V(S_i) = N_G(y) \setminus V(S_i)$.*

To characterize the notion of compatibility, we need the notions of *inset* and *outset*.

Definition 4. Let $Z \subseteq V(G)^2$. We define the *inset* of Z, denoted by I_Z, as the set $\{(l(x), l(y)) \mid \{x, y\} \in E(G), (x, y) \in Z\}$, and *outset* of Z, denoted by O_Z, as the set $\{(l(x), l(y)) \mid \{x, y\} \notin E(G), (x, y) \in Z\}$.

Let S be an induced subgraph of G. Then the *inset* (*outset*, resp.) of S, denoted by I_S (O_S, resp.), is defined to be the inset (outset, resp.) of $Z = V(S) \times N_G(V(S))$.

The following lemma from [4, Section 4.1] characterizes compatibility for a single graph S in terms of the inset and outset of S: the inset are tuples that *should* be in E, while the outset are tuples that *should not* be in E.

Lemma 5 ([4]). *Let S be an induced subgraph of G, and let $E \subseteq L \times L$. Then E is compatible with S iff $I_S \subseteq E \subseteq L^2 \setminus O_S$ (i.e., E separates I_S from O_S).*

Hence, there is an E compatible with S in G iff $I_S \cap O_S = \varnothing$.

Definition 6. Let G be a graph and let $\mathcal{S}$ be a set of mutually isomorphic subgraphs of G. We say that $E \subseteq L^2$ is *confluent* for $\mathcal{S}$ if E is compatible with any ordering C of $\mathcal{S}$.

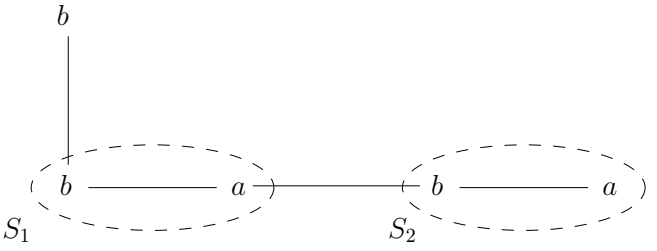


Fig. 2. Graph G from Example 1 including a depiction of its subgraphs S_1 and S_2

Example 7. We continue Example 1. Let us denote the first and second subgraph introduced in the derivation from I to G by S_1 and S_2, respectively, see Figure 2. Example 1 illustrates that the given embedding E is compatible with (S_1, S_2). Moreover, one may verify that $I_{S_1} = O_{S_1} = \{(a,b),(b,b)\}$, $I_{S_2} = \{(b,a)\}$, and $O_{S_2} = \{(a,a)\}$. Note that by Lemma 5, E is not compatible with S_1. In fact, there is no embedding compatible with S_1 as $I_{S_1} \cap O_{S_1} \neq \varnothing$. The fact that E is compatible with (S_1, S_2) but not with S_1 implies that S_1 could not be generated by E in its current context, but could be generated if S_2 was introduced after S_1.

As noted in [4], Lemma 5 can be trivially generalized to a set $\mathcal{S}$ of mutually disjoint, non-touching, and isomorphic subgraphs of G.

Lemma 8 ([4]). *Let $\mathcal{S}$ be a set of mutually disjoint, non-touching, and isomorphic subgraphs of a graph G. Then $E \subseteq L \times L$ is confluent for $\mathcal{S}$ iff $\cup_i I_{S_i} \subseteq E \subseteq L^2 \setminus (\cup_i O_{S_i})$ iff E is compatible for* some *ordering C of $\mathcal{S}$.*

6 Touching Subgraphs

We now consider the case where subgraphs touch. Compatibility for the case where two subgraphs touch is characterized in [3, Lemma 12] (see also [2]). We recall this result, cf. Lemma 12 below, and use it to establish another, similar characterization which we will be useful in the next section.

We distinguish three kinds of insets and outsets associated to pairs of touching subgraphs.

Definition 9. Let S_1 and S_2 be touching graphs in G. For $Z_1 = V(S_2) \times (V(S_1) \cap N_G(S_2))$, we denote I_{Z_1} and O_{Z_1} by $I_{(S_1,S_2)}$ and $O_{(S_1,S_2)}$, respectively. Moreover, for $Z_2 = V(S_2) \times V(S_1)$, we denote I_{Z_2} and O_{Z_2} by $I_{((S_1,S_2))}$ and $O_{((S_1,S_2))}$, respectively. Finally, for $Z_3 = V(S_2) \times (N_G(S_2) \setminus V(S_1))$, we denote I_{Z_3} and O_{Z_3} by $I_{S_2 \setminus S_1}$ and $O_{S_2 \setminus S_1}$, respectively.

We remark that we assume no interpretation of "$S_2 \setminus S_1$" in $I_{S_2 \setminus S_1}$ and $O_{S_2 \setminus S_1}$. Note that $Z_1 \cup Z_3 = V(S_2) \times N_G(S_2)$, and therefore $I_{(S_1,S_2)} \cup I_{S_2 \setminus S_1} = I_{S_2}$ and similarly for outset. Also note that $(a,b) \in I_{(S_1,S_2)}$ iff $(b,a) \in I_{(S_2,S_1)}$. Moreover, we have always $I_{(S_1,S_2)} = I_{((S_1,S_2))}$ and $O_{(S_1,S_2)} \subseteq O_{((S_1,S_2))}$.

Notice that $I_{(S_1,S_2)}$ and $O_{(S_1,S_2)}$ (and also $I_{((S_1,S_2))}$ and $O_{((S_1,S_2))}$) are concerned with the tuples going *from* S_2 *to* S_1. This is because these tuples are important in the second step in the derivation to G; the step which creates S_2.

Example 10. We continue Examples 1 and 7. One may verify that $I_{(S_1,S_2)} = I_{((S_1,S_2))} = \{(b,a)\}$, $O_{(S_1,S_2)} = \{(a,a)\}$, $O_{((S_1,S_2))} = \{(a,a),(b,b),(a,b)\}$, and $I_{S_2\setminus S_1} = O_{S_2\setminus S_1} = \varnothing$. Moreover, $I_{(S_2,S_1)} = I_{((S_2,S_1))} = \{(a,b)\}$, $O_{(S_2,S_1)} = \{(b,b)\}$, $O_{((S_2,S_1))} = \{(a,a),(b,a),(b,b)\}$, $I_{S_1\setminus S_2} = \{(b,b)\}$, and $O_{S_1\setminus S_2} = \{(a,b)\}$.

We now state an implicit result of [3].

Lemma 11. *Let S_1 and S_2 be touching subgraphs of G, and let $x \in V(S_1)$ and $y \in V(S_2)$ with $l(x) = a$ and $l(y) = b$. Moreover, $E \subseteq L^2$ be compatible with (S_1,S_2). Then $\{x,y\}$ is an edge iff $(b,a) \in E$ and $(a,N) \in E$.*

We now recall [3, Lemma 12] (it is slightly reformulated here).

Lemma 12 ([3]). *Let S_1 and S_2 be touching subgraphs of G. Then $E \subseteq L^2$ is compatible with (S_1,S_2) iff the following conditions hold:*

1. *E is compatible with S_2,*
2. *$I_{S_1\setminus S_2} \subseteq E \subseteq L^2\setminus O_{S_1\setminus S_2}$,*
3. *$\{(a,N) \mid a \in l(V(S_1) \cap N_G(S_2))\} \subseteq E$, and*
4. *If $(b,a) \in O_{((S_1,S_2))}$, then $(a,N) \notin E$ or $(b,a) \notin E$ (or both).*

Note that the fact that E is compatible with (S_1,S_2) does *not* imply that E is compatible with S_1. In fact, Example 7 illustrates that we may have $I_{S_1} \cap O_{S_1} \neq \varnothing$ — hence an embedding compatible with S_1 may not even exist.

We now obtain the following consequence of Lemma 12 by noticing that the sets $l(V(S_1) \cap N_G(S_2))$ and $l(V(S_1) \setminus N_G(S_2))$ "should" be treated differently.

Lemma 13. *Let S_1 and S_2 be touching subgraphs of G and $E \subseteq L \times L$. Then E is compatible with (S_1,S_2) iff the following conditions hold:*

1. *E is compatible with S_2,*
2. *$I_{S_1\setminus S_2} \subseteq E \subseteq L^2\setminus O_{S_1\setminus S_2}$,*
3. *$\{(a,N) \mid a \in l(V(S_1) \cap N_G(S_2))\} \subseteq E$, and*
4. *If $(a,N) \in E$ with $a \in l(V(S_1)\setminus N_G(S_2))$, then $(b,a) \notin E$ for all $b \in l(V(S_2))$.*

If this is the case, then $l(V(S_1) \cap N_G(S_2)) \cap l(V(S_1) \setminus N_G(S_2)) = \varnothing$.

Proof. We obtain the "if and only if" part by showing now that Condition 4, referred to as (I), is equivalent with Condition 4 of Lemma 12, referred to as (II), under the assumption that the other three conditions hold.

First we show that (I) implies (II). Let $(b,a) \in O_{((S_1,S_2))}$ and $(b,a) \in E$. We show that $(a,N) \notin E$. Hence there is an $x \in V(S_2)$ and $y \in V(S_1)$ with $l(x) = b$ and $l(y) = a$ such that $\{x,y\}$ is *not* an edge of G. Assume $y \in N_G(S_2)$. Then $(b,a) \in O_{(S_1,S_2)}$, and by Condition 1, $(b,a) \notin E$ — a contradiction. Therefore $y \notin N_G(S_2)$, and by (I), we have $(a,N) \notin E$ (as $(b,a) \in E$), and we are done.

We now show that (II) implies (I). Assume the contrary and let $(a,N) \in E$ and $(b,a) \in E$ for some $a \in l(V(S_1) \setminus N_G(S_2))$ and $b \in l(V(S_2))$. Since $(b,a) \in E$ we have by Condition 1 that $(b,a) \notin O_{((S_1,S_2))}$. As we assume that (II) and the other three conditions of Lemma 12 hold, we have by Lemma 12 that E is compatible for (S_1,S_2). By Lemma 11, we obtain that there is an edge $\{x,y\}$ in G

with $x \in V(S_1)$, $y \in V(S_2)$, $l(x) = a$, and $l(y) = b$. Clearly, $x \in V(S_1) \setminus N_G(S_2)$. Consequently, $(b, a) \in O_{((S_1,S_2))}$ — a contradiction.

Finally, let E be compatible with (S_1, S_2), and assume that $a \in l(V(S_1) \cap N_G(S_2)) \cap l(V(S_1) \setminus N_G(S_2))$. Let $x \in V(S_1) \cap N_G(S_2)$ and $y \in V(S_1) \setminus N_G(S_2)$ such that $l(x) = l(y) = a$. As $x, y \in V(S_1)$, we have by Lemma 3, $N_G(x) \setminus V(S_1) = N_G(y) \setminus V(S_1)$ — a contradiction as $x \in N_G(S_2)$ while $y \notin N_G(S_2)$. □

For an embedding relation $E \subseteq L^2$, and $L' \subseteq L$, we define the *restriction* of E to L' by $E[L'] = \{(x, y) \in E \mid x, y \in L'\}$. The next corollary follows from Lemma 13.

Corollary 14. *Let S_1 and S_2 be touching subgraphs of G, let $\bar{L} = L \setminus \{N\}$, and let $E' \subseteq \bar{L}^2$. Then there is an $E \subseteq L^2$, with $E[\bar{L}] = E'$, compatible with (S_1, S_2) iff the following conditions hold:*

1. *E' is compatible with S_2,*
2. *$I_{S_1 \setminus S_2} \subseteq E' \subseteq \bar{L}^2 \setminus O_{S_1 \setminus S_2}$,*
3. *$l(V(S_1) \cap N_G(S_2)) \cap l(V(S_1) \setminus N_G(S_2)) = \varnothing$*

Moreover, if this is the case, then $E = E' \cup \{(a, N) \mid a \in l(V(S_1) \cap N_G(S_2))\}$ is compatible with (S_1, S_2).

Consider again Corollary 14. Note that E compatible with S_1 (i.e., $I_{S_1} \subseteq E \subseteq L^2 \setminus O_{S_1}$) implies $I_{S_1 \setminus S_2} \subseteq E \subseteq L^2 \setminus O_{S_1 \setminus S_2}$. Moreover, we show now that if E is compatible with S_1, then the third condition of Corollary 14 also holds.

Lemma 15. *Let S_1 and S_2 be touching subgraphs of G and $E \subseteq L \times L$. If E is compatible with S_1, then $l(V(S_1) \cap N_G(S_2)) \cap l(V(S_1) \setminus N_G(S_2)) = \varnothing$.*

Proof. Assume to the contrary that $a \in l(V(S_1) \cap N_G(S_2)) \cap l(V(S_1) \setminus N_G(S_2))$. Let $x \in V(S_1) \cap N_G(S_2)$ and $y \in V(S_1) \setminus N_G(S_2)$ such that $l(x) = l(y) = a$. As $x, y \in V(S_1)$, we have by Lemma 3, $N_G(x) \setminus V(S_1) = N_G(y) \setminus V(S_1)$ — a contradiction as $x \in N_G(S_2)$ while $y \notin N_G(S_2)$. □

Hence, by Corollary 14 and Lemma 15 we notice now that an $E' \subseteq \bar{L}^2$ compatible for both S_1 and S_2 can be extended to an $E \subseteq L^2$ compatible for both (S_1, S_2) and (S_2, S_1). However, as we recall again, it may be the case that E is compatible with (S_1, S_2), while E is *not* compatible with S_1, see Example 7.

7 Symmetric Connections

In this section we consider the case where the connections between subgraphs S_1 and S_2 are symmetric, i.e., $I_{(S_1,S_2)}$ is symmetric as a relation (if $(a, b) \in I_{(S_1,S_2)}$, then also $(b, a) \in I_{(S_1,S_2)}$). Clearly, $I_{(S_1,S_2)}$ is symmetric iff $I_{(S_1,S_2)} = I_{(S_2,S_1)}$.

The next lemma is easy to verify.

Lemma 16. *Let S_1 and S_2 be isomorphic touching subgraphs of G where $I_{(S_1,S_2)}$ is symmetric. Then $l(V(S_1)) = l(V(S_2))$, $l(V(S_1) \cap N_G(S_2)) = l(V(S_2) \cap N_G(S_1))$, and $l(V(S_1) \setminus N_G(S_2)) = l(V(S_2) \setminus N_G(S_1))$.*

Proof. As S_1 and S_2 are isomorphic we have $l(V(S_1)) = l(V(S_2))$. Since $I_{(S_1,S_2)}$ is symmetric, we have moreover $l(V(S_1) \cap N_G(S_2)) = l(V(S_2) \cap N_G(S_1))$ and $l(V(S_1) \setminus N_G(S_2)) = l(V(S_2) \setminus N_G(S_1))$. □

We show now a key result.

Lemma 17. *Let S_1 and S_2 be isomorphic touching subgraphs of G where $I_{(S_1,S_2)}$ is symmetric. If $E \subseteq L^2$ is compatible for (S_1, S_2), then E is compatible for S_1.*

Proof. Recall that we have $I_{S_1} = I_{(S_2,S_1)} \cup I_{S_1 \setminus S_2}$ (and similar for O_{S_1}). Hence to show that E is compatible for S_1, i.e., $I_{S_1} \subseteq E \subseteq L^2 \setminus O_{S_1}$, it suffices to show that (1) $I_{S_1 \setminus S_2} \subseteq E \subseteq L^2 \setminus O_{S_1 \setminus S_2}$, and (2) $I_{(S_2,S_1)} \subseteq E \subseteq L^2 \setminus O_{(S_2,S_1)}$.

The former, (1), follows by Lemma 13 as E is compatible for (S_1, S_2).

We now show the latter, i.e., (2). As $I_{(S_2,S_1)} = I_{(S_1,S_2)}$ and $I_{(S_1,S_2)} \subseteq E$ since E is compatible for (S_1, S_2), we obtain $I_{(S_2,S_1)} \subseteq E$.

To show that $E \subseteq L^2 \setminus O_{(S_2,S_1)}$, it suffices to show that $O_{(S_2,S_1)} = O_{(S_1,S_2)}$ (as $E \subseteq L^2 \setminus O_{(S_1,S_2)}$).

We have by definition of inset and outset, $I_{(S_1,S_2)} \cup O_{(S_1,S_2)} = K \times K'$ where K is the set of labels of the vertices of S_2 and K' is the set of labels of the vertices in $V(S_1) \cap N_G(S_2)$. As S_1 and S_2 are isomorphic and $I_{(S_2,S_1)} = I_{(S_1,S_2)}$, we have by Lemma 16, $l(V(S_1) \cap N_G(S_2)) = l(V(S_2) \cap N_G(S_1))$ and so $I_{(S_2,S_1)} \cup O_{(S_2,S_1)} = K \times K'$. Hence $I_{(S_1,S_2)} \cup O_{(S_1,S_2)} = I_{(S_2,S_1)} \cup O_{(S_2,S_1)}$.

Since E is compatible for (S_1, S_2), we have $I_{(S_1,S_2)} \cap O_{(S_1,S_2)} = \varnothing$. We show now that $I_{(S_2,S_1)} \cap O_{(S_2,S_1)} = \varnothing$. Assume to the contrary that $(b, a) \in I_{(S_2,S_1)} \cap O_{(S_2,S_1)}$. As $(b, a) \in I_{(S_2,S_1)}$, there is an edge $\{x, y\}$ with $y \in V(S_1)$ and $x \in V(S_2)$ with $l(y) = b$ and $l(x) = a$. More specifically, $y \in V(S_1) \cap N_G(S_2)$ (and $x \in V(S_2) \cap N_G(S_1)$). As E is compatible for (S_1, S_2), we have $l(V(S_1) \cap N_G(S_2)) \cap l(V(S_1) \setminus N_G(S_2)) = \varnothing$ by Lemma 13. Consequently, since $(b, a) \in O_{(S_2,S_1)}$ and $b \in l(V(S_1) \cap N_G(S_2))$, we have $b \notin l(V(S_1) \setminus N_G(S_2))$ and therefore there is an $y' \in V(S_1) \cap N_G(S_2)$ and $x' \in V(S_2)$ with $l(y') = b$ and $l(x') = a$ such that there is no edge between y' and x'. By definition of $O_{(S_1,S_2)}$ we have now that $(a, b) \in O_{(S_1,S_2)}$. As $I_{(S_1,S_2)} \cap O_{(S_1,S_2)} = \varnothing$, we have therefore $(a, b) \notin I_{(S_1,S_2)} = I_{(S_2,S_1)}$ — a contradiction. Therefore $I_{(S_2,S_1)} \cap O_{(S_2,S_1)} = \varnothing$.

We thus obtain that $\{I_{(S_1,S_2)}, O_{(S_1,S_2)}\}$ and $\{I_{(S_2,S_1)}, O_{(S_2,S_1)}\}$ are both partitions of $K \times K'$. Since $I_{(S_2,S_1)} = I_{(S_1,S_2)}$ we obtain $O_{(S_2,S_1)} = O_{(S_1,S_2)}$. □

We are ready now to show that if an embedding E is compatible for one ordering of touching graphs S_1 and S_2, then E is compatible for the other ordering precisely when $I_{(S_1,S_2)}$ is symmetric.

Theorem 18. *Let S_1 and S_2 be isomorphic touching subgraphs of G and let $E \subseteq L^2$ be compatible for (S_1, S_2). Then E is compatible for (S_2, S_1) iff $I_{(S_1,S_2)}$ is symmetric.*

Proof. We first show the forward implication. Let E be compatible for both (S_1, S_2) and (S_2, S_1). Let $x_1 \in V(S_1)$, $y_1 \in V(S_1)$, $x_2 \in V(S_2)$, $y_2 \in V(S_2)$ with $l(x_1) = l(x_2) = a$ and $l(y_1) = l(y_2) = b$ (a may be equal to b). We need that show that if $\{x_1, y_2\}$ is an edge of G, then $\{x_2, y_1\}$ is an edge of G. If $\{x_1, y_2\}$ is

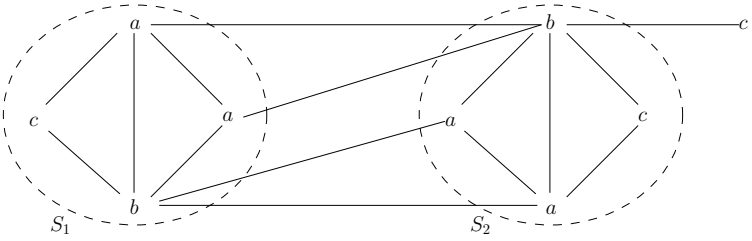


Fig. 3. Graph G from Example 20 including a depiction of its subgraphs S_1 and S_2

an edge of G, then we have by Lemma 11 $(b, a) \in E$ and $(a, N) \in E$. Since E is compatible for (S_2, S_1), we have again by Lemma 11 (and since $(b, a) \in E$ and $(a, N) \in E$) that $\{x_2, y_1\}$ is an edge of G.

We now show the reverse implication. Let $I_{(S_1,S_2)}$ be symmetric. We show that the conditions of Lemma 13 are fulfilled for (S_2, S_1). As E is compatible for (S_1, S_2), E is compatible for S_2, and therefore $I_{S_2 \setminus S_1} \subseteq E \subseteq L^2 \setminus O_{S_2 \setminus S_1}$ (Condition 2 of Lemma 13) holds.

By Lemma 16, we have $l(V(S_1) \cap N_G(S_2)) = l(V(S_2) \cap N_G(S_1))$ and therefore Condition 3 of Lemma 13 holds. Similarly, by Lemma 16, $l(V(S_1)) = l(V(S_2))$ and $l(V(S_1) \setminus N_G(S_2)) = l(V(S_2) \setminus N_G(S_1))$ holds and therefore Condition 4 of Lemma 13 holds (and using again that $I_{(S_1,S_2)}$ is symmetric).

Next, E is compatible for S_1 (Condition 1 of Lemma 13) by Lemma 17. Hence we obtain that E is compatible for (S_2, S_1). □

Example 19. Consider again Example 1. Recall that the given E is compatible for (S_1, S_2) (with S_1 and S_2 as in Figure 2). Now as $I_{(S_1,S_2)} = \{(b, a)\}$ (see Example 10) is not symmetric, we have by Theorem 18 that E is not compatible for (S_2, S_1).

Example 20. Consider now graph G and its subgraphs S_1 and S_2 given in Figure 3. One may verify that $E = \{(a, b), (b, a), (b, c), (a, N), (b, N)\}$ is compatible with (S_1, S_2). We have $I_{(S_1,S_2)} = \{(a, b), (b, a)\}$, and thus $I_{(S_1,S_2)}$ is symmetric. By Theorem 18, E is also compatible with (S_2, S_1).

Remark 21. One may wonder in view of Theorem 18 whether or not it holds that E compatible for both (S_1, S_2) and S_1 implies that $I_{(S_1,S_2)}$ is symmetric (recall that E being compatible for (S_2, S_1) implies that E is compatible for S_1). It is straightforward to verify that $E = \{(a, b), (b, a), (b, N)\}$ and S_1 and S_2 discrete graphs of two vertices labelled by a and b with one edge $\{x, y\}$ with $x \in V(S_1)$, $y \in V(S_2)$, $l(x) = a$, and $l(y) = b$ form a counterexample. □

We now generalize Theorem 18 where two subgraphs S_1 and S_2 are considered to the case where n subgraphs $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ of G are considered. It *characterizes* the notion of confluency of an embedding relation E if $\mathcal{S}$ is a set of mutually isomorphic and disjoint subgraphs of G.

Theorem 22. *Let G be a graph, let $\mathcal{S}$ be a set of mutually isomorphic and disjoint subgraphs of G, and let $E \subseteq L^2$ be compatible for some ordering C of $\mathcal{S}$.*

Then E is confluent for $\mathcal{S}$ iff (1) every E is compatible with $S_i \in \mathcal{S}$, and (2) if S_j and S_k in $\mathcal{S}$ are touching, then $I_{(S_j,S_k)}$ is symmetric.

Proof. Let $\mathcal{S} = \{S_1, \ldots, S_n\}$. We first prove the forward implication. Assume that E is confluent for $\mathcal{S}$. Let $S_i \in \mathcal{S}$. Let C be an ordering of $\mathcal{S}$ where S_i is at the last position in C. Then, by definition, S_i is compatible with E. Let now S_j and S_k in $\mathcal{S}$ be touching. Again, there are orderings C of $\mathcal{S}$ where S_j and S_k are at the last two positions, in any order, in C. Hence by Theorem 18 $I_{(S_j,S_k)}$ is symmetric.

We now prove the reverse implication. Let E be compatible with $C = (S_{i_1}, \ldots, S_{i_n})$ where $\{i_1, \ldots, i_n\} = \{1, \ldots, n\}$. Assume moreover that the conditions (1) and (2) of the theorem hold for $\mathcal{S}$. It suffices to prove that for any $k \in \{1, \ldots, n-1\}$, E is compatible with the ordering C' of $\mathcal{S}$ obtained from C by interchanging S_k and S_{k+1} in C; indeed any ordering of $\mathcal{S}$ may be obtained by iteration of this argument, and the theorem then holds.

As E is compatible with C in G, we have that E is compatible with $(S_{i_1}, \ldots, S_{i_k}, S_{i_{k+1}})$ in G', where G' is the usual "intermediate graph". More precisely, G' is the unique graph such that G is obtained from G' by iteratively applying the NLC grammar rule to create $S_{i_{k+2}}, \ldots, S_{i_n}$ (in this order). We have that E is compatible for $(S_{i_k}, S_{i_{k+1}})$ in G', and since $I_{(S_j,S_k)}$ is symmetric (also in G'), E is compatible for $(S_{i_{k+1}}, S_{i_k})$ in G'. Therefore, E is compatible with $(S_{i_1}, \ldots, S_{i_{k+1}}, S_{i_k})$ in G' and we have that E is compatible with $(S_{i_1}, \ldots, S_{i_{k+1}}, S_{i_k}, S_{i_{k+2}}, \ldots, S_{i_n})$ in G. This completes the theorem. □

8 Discussion

To compress a set $\mathcal{S}$ of disjoint and isomorphic subgraphs of G, we used NLC graph grammars consisting of a single production. As decompression should obtain precisely graph G (i.e., no other graphs), we need to ensure that graph grammar is confluent. Theorem 22 characterizes confluency (in our case under consideration) in terms of symmetry in the connections between any two touching subgraphs which are to be generated by the graph grammar. Future research could focus on extending the results to more rules. In this case one may consider recursive applications of graph grammar productions.

References

1. Bauderon, M., Courcelle, B.: Graph expressions and graph rewritings. Mathematical Systems Theory 20(2-3), 83–127 (1987)
2. Blockeel, H., Brijder, R.: Learning non-confluent NLC graph grammar rules. In: Ambos-Spies, K., Löwe, B., Merkle, W. (eds.) 5th Conference on Computability in Europe (CiE 2009), Mathematical Theory and Computational Practice, Abstract Booklet, pp. 60–69 (2009)
3. Blockeel, H., Brijder, R.: Non-confluent NLC graph grammar inference by compressing disjoint subgraphs. Accepted for Journal of Logic and Computation. Preprint version: [arXiv:0901.4876] (2010)

4. Blockeel, H., Nijssen, S.: Induction of node label controlled graph grammar rules. In: Proceeding of the 6th International Workshop on Mining and Learning with Graphs, MLG 2008 (2008)
5. Cook, D., Holder, L.: Substructure discovery using minimum description length and background knowledge. Journal of Artificial Intelligence Research 1, 231–255 (1994)
6. Cook, D., Holder, L., Su, S., Maglothin, R., Jonyer, I.: Structural mining of molecular biology data. IEEE Engineering in Medicine and Biology 20(4), 67–74 (2001)
7. Doshi, S., Huang, F., Oates, T.: Inferring the structure of graph grammars from data. In: Proceedings of the International Conference on Knowledge-Based Computer Systems, KBCS 2002 (2002)
8. Engelfriet, J., Rozenberg, G.: Graph grammars based on node rewriting: An introduction to NLC graph grammars. In: Graph-Grammars and Their Application to Computer Science, pp. 12–23 (1990)
9. Florencio, C., Ramon, J., Daenen, J., van den Bussche, J., van Dyck, D.: Context-free graph grammars as a language-bias mechanism for graph pattern mining. In: Blockeel, H., Borgwardt, K., Yan, X. (eds.) 7th International Workshop on Mining and Learning with Graphs, Extended Abstracts (2009)
10. Kukluk, J., Holder, L., Cook, D.: Inference of node replacement graph grammars. Intelligent Data Analysis 11(4), 377–400 (2007)
11. Kukluk, J., Holder, L., Cook, D.: Inference of edge replacement graph grammars. International Journal on Artificial Intelligence Tools 17(3), 539–554 (2008)
12. Kukluk, J., You, C., Holder, L., Cook, D.: Learning node replacement graph grammars in metabolic pathways. In: Arabnia, H., Yang, M., Yang, J. (eds.) BIOCOMP, pp. 44–50. CSREA Press (2007)
13. Lin, L., Wu, T., Porway, J., Xu, Z.: A stochastic graph grammar for compositional object representation and recognition. Pattern Recognition 42(7), 1297–1307 (2009)
14. Oates, T., Doshi, S., Huang, F.: Estimating maximum likelihood parameters for stochastic context-free graph grammars. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 281–298. Springer, Heidelberg (2003)

Partial Derivatives of an Extended Regular Expression

Pascal Caron, Jean-Marc Champarnaud, and Ludovic Mignot

LITIS, Université de Rouen, 76801 Saint-Étienne du Rouvray Cedex, France
{pascal.caron,jean-marc.champarnaud,ludovic.mignot}@univ-rouen.fr

Abstract. The notion of expression derivative due to Brzozowski leads to the construction of a deterministic automaton from an extended regular expression, whereas the notion of partial derivative due to Antimirov leads to the construction of a non-deterministic automaton from a simple regular expression. In this paper, we generalize Antimirov partial derivatives to regular expressions extended to complementation and intersection. For a simple regular expression with n symbols, Antimirov automaton has at most $n+1$ states. As far as an extended regular expression is concerned, we show that the number of states can be exponential.

1 Introduction

Regular expressions are a basic tool for describing patterns in a text. This is the reason why they are used in numerous domains that involve pattern specification or pattern matching, such as electronical document processing, bio-informatics or data bases. An additional advantage of regular expressions is that they can be transformed into an equivalent machine, called a finite automaton, that makes it possible to automatically decide whether a word belongs to the language denoted by an expression or not.

Simple regular expressions only contain sum, concatenation product and Kleene star operators whereas extended regular expressions in addition contain boolean operators such as complementation or intersection. Simple regular expressions have been extensively investigated. Numerous algorithms have been designed in particular for converting a regular expression into a finite automaton. These algorithms can be partitioned into two main categories. The algorithms of the first category are based on the notion of position (of a symbol occurrence in the expression). It is the case of the algorithm due to Glushkov [9] and to McNaughton and Yamada [12] that computes a non-deterministic automaton with $n + 1$ states from a n-symbol occurrence expression. Let us notice that, under some assumptions, the inductive algorithm [11,4] computes the same automaton. The algorithms of the second category are based on the computation of expression derivatives that is similar to the computation of language quotients [13,14]. It is the case of the algorithm of expression derivatives due to Brzozowski [3] that computes a deterministic automaton and of the algorithm of partial derivatives due to Antimirov [1] that computes a non-deterministic automaton with

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 179–191, 2011.

at most $n+1$ states if there are n symbol occurrences in the expression. Let us remark that the relations that exist between the two notions of position and of expression derivative have been studied in [2,5].

As far as extended regular expressions are concerned, many complexity studies [6] have been realized; we will consider in the following the results of Gelade and Neven [8] about the succinctness of the operations of complementation and intersection. On the opposite, there exist few algorithms performing the conversion of an extended regular expression into an automaton. Actually, boolean operators (except for the sum) are not compatible with the notion of position and thus Glushkov algorithm cannot be extended from simple to extended regular expressions. The inductive algorithm still works for extended regular expressions, but its performance is penalized by the determinization steps required by the automaton-implementation of some boolean operations. Actually, extended regular expressions are handled by the algorithm of Brzozowski [3]; however, a deterministic automaton is computed, which can be a drawback regarding to space complexity. As far as partial derivatives are concerned, here is what Antimirov reports in the conclusion of his paper [1]: "It would be useful to find an appropriate definition of partial derivatives of *extended* regular expressions (with intersection, complementation, and other operations). Then, in particular, our NFA construction would directly extend to this class of regular expressions."

In this paper, we extend the computation of partial derivatives in order to handle the operations of complementation and intersection. It allows us to generalize the two automaton constructions designed by Antimirov: the non-deterministic derivated term automaton and the deterministic partial derivative automaton. We also show that the partial derivative automaton may have a number of states exponential with respect to the size of the expression.

The main notions used in this paper as well as the computation of expression derivatives and partial derivatives are recalled in the next section. A generalization of the computation of partial derivatives to extended regular expressions is introduced in Section 3. Section 4 and Section 5 are respectively devoted to the construction of the derivated term automaton and of the partial derivative automaton that both recognize the language denoted by a given extended regular expression.

2 Preliminaries

A *finite automaton* A is a 5-tuple $(\Sigma, Q, I, F, \delta)$ with Σ the *alphabet* (a finite set of symbols), Q a finite set of *states*, $I \subset Q$ the set of *initial states*, $F \subset Q$ the set of *final states* and $\delta \subset Q \times \Sigma \times Q$ the set of *transitions*. The set δ is equivalent to the function from $Q \times \Sigma$ to 2^Q defined by: $q' \in \delta(q,a)$ if and only if $(q,a,q') \in \delta$. The domain of the function δ is extended to $2^Q \times \Sigma^*$ as follows: $\forall P \subset Q$, $\delta(P,\varepsilon) = P$, $\delta(P,a) = \bigcup_{p \in P} \delta(p,a)$ and $\delta(P, a \cdot w) = \delta(\delta(P,a), w)$. The automaton A *recognizes* the language $L(A) = \{w \in \Sigma^* \mid \delta(I,w) \cap F \neq \emptyset\}$. The automaton A is *deterministic* if $\#I = 1 \ \wedge \ \forall (q,a) \in Q \times \Sigma$, $\#\delta(q,a) \leq 1$. A deterministic automaton is *complete* if $\forall (q,a) \in Q \times \Sigma$, $\#\delta(q,a) = 1$.

For every automaton A, there exists a complete deterministic automaton A' such that $L(A') = L(A)$ [15].

Let f be a boolean operator of arity k. An *extended regular expression* E over an alphabet Σ is inductively defined by $E = \emptyset$, $E = \varepsilon$, $E = a$, $E = \mathrm{f}(E_1, \ldots, E_k)$, $E = (F \cdot G)$, $E = (F^*)$ where $a \in \Sigma$ and F, G, $E_1, \ldots, E_k$ are extended regular expressions. The regular expression E is said to be a *simple expression* if it only contains sum, concatenation product and Kleene star operators.

The *language denoted by* the expression E is inductively defined by

$$L(\emptyset) = \emptyset,\ L(\varepsilon) = \{\varepsilon\},\ L(a) = \{a\}\ \forall a \in \Sigma,\ L(F \cdot G) = L(F) \cdot L(G),$$
$$L(F^*) = L(F)^*,\ L(\mathrm{f}(E_1, \ldots, E_k)) = \mathrm{f}_L(L(E_1), \ldots, L(E_k)),$$

where f_L is the language operator associated with the operator[1] f.

A language L is *regular* if and only if there exists a simple regular expression E such that $L(E) = L$. It has been proved by Kleene [10] that a language is regular if and only if it is recognized by a finite automaton. Moreover, given a complete deterministic automaton A, a deterministic automaton A' such that $L(A') = \neg L(A)$ can be constructed by setting non final the final states and vice versa. Therefore, the set of regular languages is closed under any boolean operator.

The *quotient of a language* L w.r.t. a symbol a is the language $a^{-1}(L) = \{w \in \Sigma^* \mid aw \in L\}$. It can be recursively computed as follows:

$$a^{-1}(\emptyset) = a^{-1}(\{\varepsilon\}) = \emptyset,\ a^{-1}(\{b\}) = \{\varepsilon\} \text{ if } a = b,\ \emptyset \text{ otherwise},$$
$$a^{-1}(L_1 \cup L_2) = a^{-1}(L_1) \cup a^{-1}(L_2),\ a^{-1}(L_1^*) = a^{-1}(L_1) \cdot L_1^*$$
$$a^{-1}(L_1 \cdot L_2) = \begin{cases} a^{-1}(L_1) \cdot L_2 \cup a^{-1}(L_2) & \text{if } \varepsilon \in L_1, \\ a^{-1}(L_1) \cdot L_2 & \text{otherwise,} \end{cases}$$
$$a^{-1}(\mathrm{f}_L(L_1, \ldots, L_k)) = \mathrm{f}_L(a^{-1}(L_1), \ldots, a^{-1}(L_k)).$$

The quotient $w^{-1}(L)$ of L w.r.t. the word $w \in \Sigma^*$ is the language $\{w' \in \Sigma^* \mid w \cdot w' \in L\}$. It can be recursively computed as follows: $\varepsilon^{-1}(L) = L$, $(aw')^{-1}(L) = w'^{-1}(a^{-1}(L))$ with $a \in \Sigma$ and $w' \in \Sigma^*$. Myhill-Nerode theorem [13,14] states that a language L is regular if and only if the set of quotients $\{u^{-1}(L) \mid u \in \Sigma^*\}$ is finite.

The notion of derivative of an expression has been introduced by Brzozowski [3]. Let E be an extended regular expression over an alphabet Σ and let a and b be two distinct symbols of Σ. The *derivative of* E w.r.t. a is the expression $\frac{d}{da}(E)$ inductively computed as follows[2]:

$$\tfrac{d}{da}(\emptyset) = \tfrac{d}{da}(\varepsilon) = \tfrac{d}{da}(b) = \emptyset,\ \tfrac{d}{da}(a) = \varepsilon,$$
$$\tfrac{d}{da}(\mathrm{f}(E_1, \ldots, E_k)) = \mathrm{f}(\tfrac{d}{da}(E_1), \ldots, \tfrac{d}{da}(E_k)),\ \tfrac{d}{da}(F^*) = \tfrac{d}{da}(F) \cdot F^*,$$
$$\tfrac{d}{da}(F \cdot G) = \begin{cases} \frac{d}{da}(F) \cdot G + \frac{d}{da}(G) & \text{if } \varepsilon \in L(F), \\ \frac{d}{da}(F) \cdot G & \text{otherwise.} \end{cases}$$

The derivative of E w.r.t. a word w of Σ^* is defined by:

$$\tfrac{d}{dw}(E) = \begin{cases} \frac{d}{dw'}(\frac{d}{da}(E)) & \text{if } w = a \cdot w' \text{ with } a \in \Sigma \text{ and } w' \in \Sigma^* \\ E & \text{if } w = \varepsilon. \end{cases}$$

[1] For instance $L(\neg E) = \neg L(E)$, $L(E + F) = L(E) \cup L(F)$.

[2] This notation is used in order to distinguish the derivative of an expression from the quotient of a language.

The set of derivatives of an expression E is not necessarily finite. It has been proved by Brzozowski [3] that it is sufficient to use the ACI equivalence (that is based on the associativity, the commutativity and the idempotence of the sum) to obtain a finite set of derivatives: the set $\mathcal{D}_E$ of *dissimilar derivatives*. Given a class of ACI-equivalent expressions, a unique representative can be obtained after deleting parenthesis (associativity), ordering terms of each sum (commutativity) and deleting redundant subexpressions (idempotence). Let $E_{\sim_s}$ be the unique representative of the class of the expression E. The set of dissimilar derivatives can be computed as follows:

$$\frac{d'}{d'_a}(\emptyset) = \frac{d'}{d'_a}(\varepsilon) = \frac{d'}{d'_a}(b) = \emptyset,\ \frac{d'}{d'_a}(a) = \varepsilon,$$
$$\frac{d'}{d'_a}(\mathrm{f}(E_1, \ldots, E_k)) = (\mathrm{f}(\frac{d'}{d'_a}(E_1), \ldots, \frac{d'}{d'_a}(E_k)))_{\sim_s},$$
$$\frac{d'}{d'_a}(F^*) = \frac{d'}{d'_a}(F) \cdot F^*,$$
$$\frac{d'}{d'_a}(F \cdot G) = \begin{cases} (\frac{d'}{d'_a}(F) \cdot G + \frac{d'}{d'_a}(G))_{\sim_s} & \text{if } \varepsilon \in L(F), \\ (\frac{d'}{d'_a}(F) \cdot G)_{\sim_s} & \text{otherwise.} \end{cases}$$

Example 1. Let us consider the expression $E = a^* \cdot a^*$ over the alphabet $\Sigma = \{a\}$. The set of derivatives of E is infinite since for every $w \in \Sigma^*$, $\frac{d}{d_{wa}}(E) = \frac{d}{d_w}(E) + a^*$. On the opposite, since $\frac{d}{d_{aa}}(E) = a^* \cdot a^* + a^* + a^* \sim_s a^* \cdot a^* + a^* = \frac{d}{d_a}(E)$, it holds $\frac{d'}{d'_{aa}}(E) = \frac{d'}{d'_a}(E)$. Thus the set of dissimilar derivatives of E is $\mathcal{D}_E = \{a^*a^*, a^*a^* + a^*\}$.

The *derivative automaton* $B = (\Sigma, Q, q_0, F, \delta)$ of an extended regular expression E over an alphabet Σ is defined by $Q = \mathcal{D}_E$, $q_0 = E$, $F = \{q \in Q \mid \varepsilon \in L(q)\}$, $\delta = \{(q, a, q') \in Q \times \Sigma \times Q \mid \frac{d'}{d'_a}(q) = q'\}$. The automaton B is deterministic and it recognizes the language $L(E)$. Its size can be exponentially larger than the number of symbols of E (see Example 2).

Example 2. Consider the expression $E = (a+b)^*a(a+b)^n$. The set of its derivatives can be computed as follows, where $\Sigma = a + b$:

$\frac{d'}{d'_a}(E) = E + \Sigma^n$ $\quad \frac{d'}{d'_{aa}}(E) = E + \Sigma^n + \Sigma^{n-1}$ $\quad \frac{d'}{d'_a}(\Sigma^k) = \frac{d'}{d'_b}(\Sigma^k) = \Sigma^{k-1}$
$\frac{d'}{d'_b}(E) = E$ $\quad \frac{d'}{d'_{ab}}(E) = E + \Sigma^{n-1}$

The set $\mathcal{D}_E$ of dissimilar derivatives of E is equal to the set $E \cup \{E + \sum_{F \in \mathcal{F}} F \mid \mathcal{F} \subset \{\Sigma^n, \ldots, \Sigma, \varepsilon\}\}$. Consequently, $\#\mathcal{D}_E = 1 + 2^{n+1}$. The derivative automaton of E is presented in Figure 1.

Antimirov algorithm [1] constructs a non-deterministic automaton from a simple regular expression E. It is based on the *partial derivative* computation. The partial derivative of a simple regular expression E w.r.t. a symbol a is the set $\frac{\partial}{\partial_a}(E)$ of expressions defined as follows:

$$\frac{\partial}{\partial_a}(\emptyset) = \frac{\partial}{\partial_a}(\varepsilon) = \frac{\partial}{\partial_a}(b) = \emptyset,\ \frac{\partial}{\partial_a}(a) = \{\varepsilon\},$$
$$\frac{\partial}{\partial_a}(F + G) = \frac{\partial}{\partial_a}(F) \cup \frac{\partial}{\partial_a}(G),\ \frac{\partial}{\partial_a}(F^*) = \frac{\partial}{\partial_a}(F) \cdot F^*,$$
$$\frac{\partial}{\partial_a}(F \cdot G) = \begin{cases} \frac{\partial}{\partial_a}(F) \cdot G \cup \frac{\partial}{\partial_a}(G) & \text{if } \varepsilon \in L(F), \\ \frac{\partial}{\partial_a}(F) \cdot G & \text{otherwise,} \end{cases}$$

with for a set $\mathcal{E}$ of expressions, $\mathcal{E} \cdot F = \bigcup_{E \in \mathcal{E}} E \cdot F$.

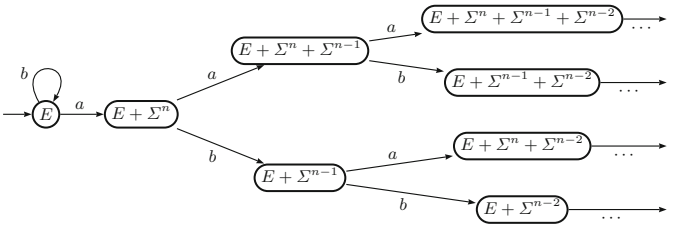


Fig. 1. The derivative automaton of $E = (a + b)^* a(a + b)^n$

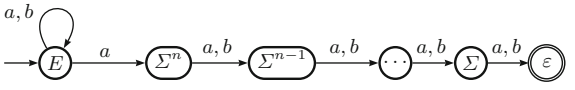


Fig. 2. The derivated term automaton of $E = (a + b)^* a(a + b)^n$

The partial derivative of E w.r.t. a word w of Σ^* is computed as follows:

$$\frac{\partial}{\partial_w}(E) = \begin{cases} \frac{\partial}{\partial_{w'}}(\frac{\partial}{\partial_a}(E)) & \text{if } w = a \cdot w' \text{ with } a \in \Sigma \text{ and } w' \in \Sigma^*, \\ \frac{\partial}{\partial_w}(E) = \{E\} & \text{if } w = \varepsilon, \end{cases}$$

with for a set $\mathcal{E}$ of expressions, $\frac{\partial}{\partial_a}(\mathcal{E}) = \bigcup_{E \in \mathcal{E}} \frac{\partial}{\partial_a}(E)$.

Every element of a partial derivative is called a *derivated term* of E. It has been shown by Antimirov [1] that the set $\mathcal{D}'_E$ of the derivated terms of E is such that $\#\mathcal{D}'_E \leq n + 1$. The *derivated term automaton* $A = (\Sigma, Q, q_0, T, \delta)$ of a simple regular expression E is defined as follows: $Q = \mathcal{D}'_E$, $q_0 = E$, $F = \{q \in Q \mid \varepsilon \in L(q)\}$, $\delta = \{(q, a, q') \in Q \times \Sigma \times Q \mid q' \in \frac{\partial}{\partial_a}(q)\}$. The automaton A recognizes the language $L(E)$.

Example 3. Consider the expression $E = (a + b)^* a(a + b)^n$ of Example 2. The derivated terms of E are computed as follows:

$$\begin{aligned} \frac{\partial}{\partial_a}((a + b)^* a(a + b)^n) &= \frac{\partial}{\partial_a}((a + b)^*) \cdot a(a + b)^n \cup \frac{\partial}{\partial_a}(a(a + b)^n) \\ &= \{E\} \cup \{\Sigma^n\}, \\ \frac{\partial}{\partial_b}(E) = \{E\}, \quad \frac{\partial}{\partial_a}(\Sigma^k) &= \frac{\partial}{\partial_b}(\Sigma^k) = \{\Sigma^{k-1}\}. \end{aligned}$$

The set of derivated terms of E is $\mathcal{D}'_E = \{E, \Sigma^n, \Sigma^{n-1}, \ldots, \varepsilon\}$; the number of derivated terms is equal to $n + 2$. The derivated term automaton of E is represented in Figure 2.

3 Partial Derivatives of an Extended Expression

This section describes two extensions of the computation of partial derivatives for extended regular expressions. The first one is a natural extension in which the simple operators ($+$, $\cdot$ and *) activate the standard partial derivative computation, whereas the boolean operators (except for the sum) activate the Brzozowski derivative computation. The interest of this technique is to make easier the

understanding of the second extension, where partial derivatives of a new type are computed.

3.1 A Natural Extension

A natural extension of the partial derivative computation is achieved by assuming that the partial derivative of a boolean operator (except for the sum) is the singleton equal to the derivative of this operator.

Definition 1. *The partial derivative of an extended regular expression w.r.t. a symbol a is the set $\frac{\partial}{\partial_a}(E)$ of expressions computed as follows:*

$$\frac{\partial}{\partial_a}(\emptyset) = \frac{\partial}{\partial_a}(\varepsilon) = \frac{\partial}{\partial_a}(b) = \emptyset,\ \frac{\partial}{\partial_a}(a) = \{\varepsilon\},$$
$$\frac{\partial}{\partial_a}(F+G) = \frac{\partial}{\partial_a}(F) \cup \frac{\partial}{\partial_a}(G),\ \frac{\partial}{\partial_a}(F^*) = \frac{\partial}{\partial_a}(F)\cdot F^*,$$
$$\frac{\partial}{\partial_a}(F\cdot G) = \begin{cases} \frac{\partial}{\partial_a}(F)\cdot G \cup \frac{\partial}{\partial_a}(G) & \text{if } \varepsilon \in L(F), \\ \frac{\partial}{\partial_a}(F)\cdot G & \text{otherwise,} \end{cases}$$
$$\frac{\partial}{\partial_a}(\mathrm{f}(E_1,\ldots,E_k)) = \{\tfrac{d'}{d'_a}(\mathrm{f}(E_1,\ldots,E_k))\}$$

Example 4. This example illustrates the worst case of the natural extension, where partial derivative computation reduces to derivative computation. Let us consider the three expressions F, G and E defined by:

$$F = (a+b)^*a(a+b)^n,\ G = (\neg(\neg a \cap \neg b))^*a(a+b)^n,\ E = F \cap G.$$

According to Definition 1, $\frac{\partial}{\partial_a}(E) = \{\frac{d'}{d'_a}(E)\}$. Since the intersection operator is at the highest level of the syntax tree of E, for all word w in Σ^*, the partial derivative of E w.r.t. w contains a unique derivated term: the derivative of E w.r.t. w. The expression F has an exponential number of dissimilar derivatives (*cf.* Example 2) and $\frac{d'}{d'_a}(F) \neq \emptyset \Leftrightarrow \frac{d'}{d'_a}(G) \neq \emptyset$. Consequently, E has an exponential number of derivated terms.

3.2 Set of Sets Extension

The main strength of the partial derivative computation for a simple regular expression is to break the partial derivative of a sum into a union of derivated terms. A partial derivative is a set of simple regular expressions, the language of which is the union of the languages denoted by these expressions. As far as recognizers are concerned, the advantage is that the derivated term automaton is a non-deterministic one, and then it may be exponentially smaller than the deterministic derivative automaton. For extended regular expressions, Example 4 shows that the computation of a partial derivative is ineffective for any derivated term for which the highest operator is different from the sum.

We now show how to break the partial derivative of an intersection expression into a union of derivated terms. Consider the Example 4, where:

$$F = (a+b)^*a(a+b)^n,\ G = (\neg(\neg a \cap \neg b))^*a(a+b)^n \text{ and } E = F \cap G.$$

Let us set: $\mathcal{H}_1 = \{F, \Sigma^n, \ldots, \Sigma^{n-k+1}\}$ and $\mathcal{H}_2 = \{G, \Sigma^n, \ldots, \Sigma^{n-k+1}\}$.

In the natural extension, the partial derivative of E w.r.t. a^k contains a unique derivated term $T = (F + \Sigma^n + \ldots + \Sigma^{n-k+1}) \cap (G + \Sigma^n + \ldots + \Sigma^{n-k+1})$.

By distributivity of $\cap$ over $+$, the expression T is equivalent to the expression $T' = \sum_{(H_1,H_2)\in\mathcal{H}_1\times\mathcal{H}_2} H_1 \cap H_2$.

Let us set: $\mathcal{K} = \{F, \Sigma^n, \ldots, \Sigma^0\} \times \{G, \Sigma^n, \ldots, \Sigma^0\}$. It can be checked that every derivated term of E is equivalent to a sum of expressions $H_i \cap H_j$, where (H_i, H_j) is an element in $\mathcal{K}$. There exist $(n+2)^2$ distinct expressions $H_i \cap H_j$, and each one will be transformed into a derivated term in our second extension. As a result of breaking the partial derivative of an intersection just like Antimirov breaks the partial derivative of a sum, the number of derivated terms can be exponentially smaller than the number of dissimilar derivatives. Finally, for an extended regular expression, a partial derivative is a set of derivated terms (the language of which is a union of languages), and a derivated term is a set of expressions (the language of which is an intersection of languages). Thus a partial derivative is a particular set of expression sets. We now study the properties of sets of expression sets.

Let $\mathbb{E}$ be a set of expression sets, with: $\mathbb{E} = \bigcup_{\mathcal{E}\in\mathbb{E}} \mathcal{E}$ and, for all $\mathcal{E} \in \mathbb{E}$, $\mathcal{E} = \bigcup_{E\in\mathcal{E}} E$. The language of $\mathbb{E}$ is defined by: $L(\mathbb{E}) = \bigcup_{\mathcal{E}\in\mathbb{E}} L(\mathcal{E})$ and, for all $\mathcal{E} \in \mathbb{E}$, $L(\mathcal{E}) = \bigcap_{E\in\mathcal{E}} L(E)$.

The following properties are satisfied by sets of expression sets.

Lemma 1. *Let $\mathbb{E}$ and $\mathbb{E}'$ be two sets of expression sets, $\mathcal{E}$ and $\mathcal{E}'$ be two expression sets, and F be an expression.*

(1) $L(\mathcal{E}) \cap L(\mathcal{E}') = L(\mathcal{E} \cup \mathcal{E}')$,
(2) $L(\mathbb{E}) \cap L(\mathbb{E}') = \bigcup_{\mathcal{E}\in\mathbb{E},\mathcal{E}'\in\mathbb{E}'} L(\mathcal{E} \cup \mathcal{E}')$,
(3) $\neg L(\mathbb{E}) = L(\bigcap_{\mathcal{E}\in\mathbb{E}} \sum_{E\in\mathcal{E}} \neg E)$,
(4) $L(\mathbb{E}) \cdot L(F) = L(\sum_{\mathcal{E}\in\mathbb{E}}((\bigcap_{E\in\mathcal{E}} E) \cdot F))$.

Proof. According to definitions and properties of language operators:

(1) $L(\mathcal{E}) \cap L(\mathcal{E}') = \bigcap_{E\in\mathcal{E}} L(E) \cap \bigcap_{E'\in\mathcal{E}'} L(E')$
$= \bigcap_{E\in\mathcal{E}\cup\mathcal{E}'} L(E) \qquad = L(\mathcal{E} \cup \mathcal{E}')$

(2) $L(\mathbb{E}) \cap L(\mathbb{E}') = \bigcup_{\mathcal{E}\in\mathbb{E}} L(\mathcal{E}) \cap \bigcup_{\mathcal{E}'\in\mathbb{E}'} L(\mathcal{E}')$
$= \bigcup_{\mathcal{E}\in\mathbb{E},\mathcal{E}'\in\mathbb{E}'} L(\mathcal{E}) \cap L(\mathcal{E}') \quad = \bigcup_{\mathcal{E}\in\mathbb{E},\mathcal{E}'\in\mathbb{E}'} L(\mathcal{E} \cup \mathcal{E}')$

(3) $\neg L(\mathbb{E}) = \neg \bigcup_{\mathcal{E}\in\mathbb{E}} L(\mathcal{E}) \qquad = \neg \bigcup_{\mathcal{E}\in\mathbb{E}} \bigcap_{E\in\mathcal{E}} L(E)$
$= \bigcap_{\mathcal{E}\in\mathbb{E}} \bigcup_{E\in\mathcal{E}} \neg L(E) \; = \bigcap_{\mathcal{E}\in\mathbb{E}} \bigcup_{E\in\mathcal{E}} L(\neg E)$
$= \bigcap_{\mathcal{E}\in\mathbb{E}} L(\sum_{E\in\mathcal{E}} \neg E) = L(\bigcap_{\mathcal{E}\in\mathbb{E}} \sum_{E\in\mathcal{E}} \neg E)$

(4) $L(\mathbb{E}) \cdot L(F) = (\bigcup_{\mathcal{E}\in\mathbb{E}} L(\mathcal{E})) \cdot L(F) \qquad = (\bigcup_{\mathcal{E}\in\mathbb{E}} L(\mathcal{E}) \cdot L(F))$
$= (\bigcup_{\mathcal{E}\in\mathbb{E}} L(\bigcap_{E\in\mathcal{E}} E) \cdot L(F))$
$= (\bigcup_{\mathcal{E}\in\mathbb{E}} L((\bigcap_{E\in\mathcal{E}} E) \cdot F))$
$= L(\sum_{\mathcal{E}\in\mathbb{E}}((\bigcap_{E\in\mathcal{E}} E) \cdot F))$

□

Let $\mathbb{E}$ and $\mathbb{F}$ be two sets of expression sets and G be an expression. The three following operators are defined: $\mathbb{E} \odot G = \bigcup_{\mathcal{E}\in\mathbb{E}}((\bigcap_{E\in\mathcal{E}} E) \cdot G)$, $\mathbb{E} \owedge \mathbb{F} = \bigcup_{\mathcal{E}\in\mathbb{E},\mathcal{F}\in\mathbb{F}}(\mathcal{E} \cup \mathcal{F})$, $\ominus\mathbb{E} = \owedge_{\mathcal{E}\in\mathbb{E}}(\bigcup_{E\in\mathcal{E}} \neg E)$. According to Lemma 1, it holds: $L(\mathbb{E} \odot F) = L(\mathbb{E}) \cdot L(F)$, $L(\mathbb{E} \owedge \mathbb{F}) = L(\mathbb{E}) \cap L(\mathbb{F})$ and $L(\ominus\mathbb{E}) = \neg L(\mathbb{E})$. These operators are used to define partial derivatives of an extended regular expression.

Definition 2. *The partial derivative of an extended regular expression E w.r.t. a symbol a is the set $\frac{\partial}{\partial_a}(E)$ of expression sets defined by:*

$$\textbf{(1)}\ \tfrac{\partial}{\partial_a}(\emptyset) = \tfrac{\partial}{\partial_a}(\varepsilon) = \tfrac{\partial}{\partial_a}(b) = \emptyset, \quad \textbf{(2)}\ \tfrac{\partial}{\partial_a}(a) = \{\{\varepsilon\}\}$$

$$\textbf{(3)}\ \tfrac{\partial}{\partial_a}(E+F) = \tfrac{\partial}{\partial_a}(E) \cup \tfrac{\partial}{\partial_a}(F), \quad \textbf{(4)}\ \tfrac{\partial}{\partial_a}(E^*) = \tfrac{\partial}{\partial_a}(E) \odot E^*$$

$$\textbf{(5)}\ \tfrac{\partial}{\partial_a}(E \cdot F) = \begin{cases} \tfrac{\partial}{\partial_a}(E) \odot F & \text{if } \varepsilon \notin L(E) \\ \tfrac{\partial}{\partial_a}(E) \odot F \cup \tfrac{\partial}{\partial_a}(F) & \text{otherwise.} \end{cases}$$

$$\textbf{(6)}\ \tfrac{\partial}{\partial_a}(\neg E) = \ominus(\tfrac{\partial}{\partial_a}(E)), \quad \textbf{(7)}\ \tfrac{\partial}{\partial_a}(E \cap F) = \tfrac{\partial}{\partial_a}(E) \owedge \tfrac{\partial}{\partial_a}(F)$$

Proposition 1. *Let E be an extended regular expression over the alphabet Σ and a be a symbol in Σ. Then it holds: $L(\frac{\partial}{\partial_a}(E)) = a^{-1}(L(E))$.*

Proof. By induction on the structure of extended regular expressions.

According to [1], formulas **(1)** to **(3)** are satisfied.

$$\begin{aligned}
\textbf{(4)}\ L(\tfrac{\partial}{\partial_a}(E^*)) &= L(\tfrac{\partial}{\partial_a}(E) \odot E^*) \quad = L(\tfrac{\partial}{\partial_a}(E)) \cdot L(E^*) \\
&= a^{-1}(L(E)) \cdot L(E^*) = a^{-1}(L(E^*)) \\
\textbf{(5)}\ L(\tfrac{\partial}{\partial_a}(E \cdot F)) &= L(\tfrac{\partial}{\partial_a}(E) \odot F) \quad = L(\tfrac{\partial}{\partial_a}(E)) \cdot L(F) \\
&= a^{-1}(L(E)) \cdot L(F) = a^{-1}(L(E \cdot F)) \\
\textbf{(5')}\ L(\tfrac{\partial}{\partial_a}(E \cdot F)) &= L(\tfrac{\partial}{\partial_a}(E) \odot F \cup \tfrac{\partial}{\partial_a}(F)) \\
&= L(\tfrac{\partial}{\partial_a}(E)) \cdot L(F) \cup L(\tfrac{\partial}{\partial_a}(F)) \\
&= a^{-1}(L(E)) \cdot L(F) \cup a^{-1}(L(F)) = a^{-1}(L(E \cdot F)) \\
\textbf{(6)}\ L(\tfrac{\partial}{\partial_a}(\neg E)) &= L(\ominus \tfrac{\partial}{\partial_a}(E)) \quad = \neg L(\tfrac{\partial}{\partial_a}(E)) \\
&= \neg(a^{-1}(L(E))) = a^{-1}(\neg(L(E))) \\
\textbf{(7)}\ L(\tfrac{\partial}{\partial_a}(E \cap F)) &= L(\tfrac{\partial}{\partial_a}(E) \owedge \tfrac{\partial}{\partial_a}(F)) \quad = L(\tfrac{\partial}{\partial_a}(E)) \cap L(\tfrac{\partial}{\partial_a}(F)) \\
&= a^{-1}(L(E)) \cap a^{-1}(L(F)) = a^{-1}(L(E \cap F))
\end{aligned}$$

□

Every partial derivative of an extended regular expression E w.r.t. a symbol is a set $\mathbb{E}$ in which each element $\mathcal{E}$ is a derivated term of E. The partial derivative of a derivated term is computed as follows.

Definition 3. *Let E be an extended regular expression over an alphabet Σ and a be a symbol in Σ. Let $\mathcal{E}$ be a derivated term of E. Then:*

$$\tfrac{\partial}{\partial_a}(\mathcal{E}) = \owedge_{E_k \in \mathcal{E}}(\tfrac{\partial}{\partial_a}(E_k)).$$

Proposition 2. *Let E be an extended regular expression over an alphabet Σ and a be a symbol in Σ. Let $\mathcal{E}$ be a derivated term of E. Then:*

$$L(\tfrac{\partial}{\partial_a}(\mathcal{E})) = a^{-1}L(\mathcal{E}).$$

Proof. By equality of sets:

$$\begin{aligned}
L(\tfrac{\partial}{\partial_a}(\mathcal{E})) &= L(\owedge_{E_k \in \mathcal{E}} \tfrac{\partial}{\partial_a}(E_k)) \quad = \bigcap_{E_k \in \mathcal{E}} L(\tfrac{\partial}{\partial_a}(E_k)) \\
&= \bigcap_{E_k \in \mathcal{E}} a^{-1}(L(E_k)) = a^{-1}(L(\bigcap_{E_k \in \mathcal{E}} E_k)) = a^{-1}(L(\mathcal{E}))
\end{aligned}$$

□

The partial derivative of an extended regular expression E w.r.t. a word aw in Σ^+ is defined by $\frac{\partial}{\partial_{aw}}(E) = \bigcup_{\mathcal{E}' \in \frac{\partial}{\partial_a}(E)} \frac{\partial}{\partial_w}(\mathcal{E}')$.

Proposition 3. *Let E be an extended regular expression over an alphabet Σ and aw be a word in Σ^+. Then it holds: $(aw)^{-1}(L(E)) = L(\frac{\partial}{\partial_{aw}}(E))$.*

Proof. By induction on the length of w:

$$L(\tfrac{\partial}{\partial_u}(E)) = L(\tfrac{\partial}{\partial_{aw'}}(E)) = L(\tfrac{\partial}{\partial_{w'}}(\tfrac{\partial}{\partial_a}(E))).$$

By the recurrence hypothesis: $L(\frac{\partial}{\partial_{w'}}(\frac{\partial}{\partial_a}(E))) = w'^{-1}(L(\frac{\partial}{\partial_a}(E)))$.

According to Proposition 1, $L(\frac{\partial}{\partial_a}(E)) = a^{-1}(L(E))$.

Consequently, $L(\frac{\partial}{\partial_u}(E)) = w'^{-1}(a^{-1}(L(E))) = u^{-1}(L(E))$. □

4 The Derivated Term Automaton

Let E be an extended regular expression, $\mathcal{E}$ be a derivated term of E, and $\mathbb{E}$ be a partial derivative of E such that $\mathcal{E} \in \mathbb{E}$. Every expression E_k in $\mathcal{E}$ is called a *derivated expression* of E. This relation is noted $E_k \circledcirc \mathbb{E}$. According to Definition 3, for all symbol $a \in \Sigma$ and for all derivated expression $E_k \circledcirc \frac{\partial}{\partial_a}(\mathcal{E})$, there exists a derivated expression F_k in $\mathcal{E}$ such that $E_k \circledcirc \frac{\partial}{\partial_a}(F_k)$. Consequently, in order to compute the set of derivated terms of E, it is sufficient to compute the partial derivatives of derivated expressions and then to combine them with the operator $\circledcap$.

The set $\mathcal{D}'_E = \{\mathcal{E} \mid \exists w \in \Sigma^*, \mathcal{E} \in \frac{\partial}{\partial_w}(E)\}$ is called the *set of derivated terms* of E. The *derivated term automaton of* E, $A = (\Sigma, Q, q_0, F, \delta)$, is defined similarly as for a simple expression: $Q = \mathcal{D}'_E$, $q_0 = \{E\}$, $F = \{\mathcal{E}' \in \mathcal{D}'_E \mid \varepsilon \in L(\mathcal{E}')\}$, and $\delta = \{(\mathcal{F}, a, \mathcal{G}) \in Q \times \Sigma \times Q \mid \mathcal{G} \in \frac{\partial}{\partial_a}(\mathcal{F})\}$.

Example 5. Let us consider the expression $E = F \cap G$ in Example 4. The derivated terms of E are computed as follows:

$$\begin{aligned}
\tfrac{\partial}{\partial_a}(E) &= \tfrac{\partial}{\partial_a}(F) \circledcap \tfrac{\partial}{\partial_a}(G) & \tfrac{\partial}{\partial_b}(E) &= \{\{F, G\}\} \\
&= \{\{F\}, \{\Sigma^n\}\} \circledcap \{\{G\}, \{\Sigma^n\}\} & & \\
&= \{\{F, G\}, \{F, \Sigma^n\}, \{G, \Sigma^n\}, \{\Sigma^n\}\} & & \\
\tfrac{\partial}{\partial_a}(F) &= \{\{F\}, \{\Sigma^n\}\} & \tfrac{\partial}{\partial_b}(F) &= \{\{F\}\} & \tfrac{\partial}{\partial_a}(\{\Sigma^k\}) &= \{\{\Sigma^{k-1}\}\} \\
\tfrac{\partial}{\partial_a}(G) &= \{\{G\}, \{\Sigma^n\}\} & \tfrac{\partial}{\partial_b}(G) &= \{\{G\}\} & \tfrac{\partial}{\partial_b}(\{\Sigma^k\}) &= \{\{\Sigma^{k-1}\}\}
\end{aligned}$$

The set of derivated terms is: $\{F \cap G\} \cup \{F, \Sigma^n, \ldots, \varepsilon\} \times \{G, \Sigma^n, \ldots, \varepsilon\}$.

There are k derivated terms, where:

$$k = 1 + (n+2) + (n+2) + (n+1) + \ldots + 2 = \tfrac{(n+2)(n+4)}{2}.$$

The derivated term automaton of E is represented in Figure 3.

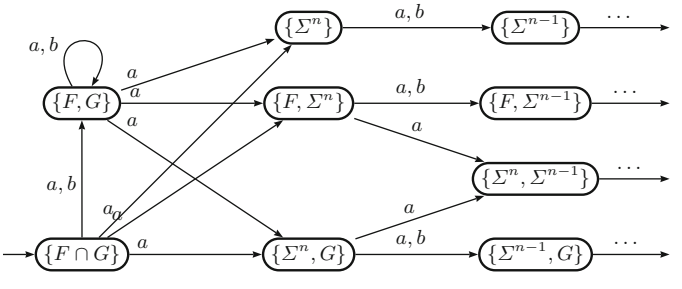


Fig. 3. The derivated term automaton of $E = F \cap G$

Proposition 4. *Let E be an extended regular expression and A be its derivated term automaton. Then it holds: $L(A) = L(E)$.*

Proof. By construction, $w \in L(A) \Leftrightarrow w \in L(\frac{\partial}{\partial_w}(E)) \Leftrightarrow w \in L(E)$. □

The derivated term automaton of an extended regular expression E may have an exponential number of states with respect to the width of E. It is the case for the expression $F = \neg E_n$ where $E_n = (a+b)^* a (a+b)^n$. It can be checked however that in this case, the derivated term automaton is not minimal. This situation is consistent with the following proposition that addresses the complexity of the size of the automata that recognize the language $L(\neg E)$, where E is a simple regular expression.

Proposition 5. ***(1)*** *For every simple regular expression E over an alphabet Σ, it is possible to compute an automaton D such that:*

$$L(D) = L(\neg E) \text{ and } O(|D|) = O(2^{|E|+1}).$$

(2) *There exists a family of expressions $(r_n)_{n\in\mathbb{N}}$ such that every automaton recognizing $L(\neg r_n)$ has an exponential size with respect to $|r_n| + 1$.*

Proof. Proof of this proposition is based on the following known results:

(a) For every simple regular expression E, a non-deterministic automaton G can be computed such that $L(G) = L(E)$ and $|G| = |E| + 1$ [9,12] or $|G| \leq |E| + 1$ [1]. **(b)** For every non-deterministic automaton A, there exists a deterministic and complete automaton D such that $L(D) = L(\neg E)$ and $|D| \leq 2^{|A|}$ [15]. **(c)** For every automaton A over an alphabet Σ, a simple regular expression E such that $L(A) = L(E)$ and $O(|E|) = O(|A| \times |\Sigma| \times 4^{|A|})$ [12,6] can be computed. **(d)** For Σ an alphabet of size 4, for all $n \in \mathbb{N}$, there exists a simple regular expression r_n which size is $O(n)$ such that for all simple regular expression r such that $L(r) = \Sigma^* \setminus L(r_n)$, $|r| \geq 2^{2^n}$ [8,7].

(1) From **(a)** and **(b)**.

(2) According to **(1)**, for every simple regular expression E, one can construct an automaton D such that $L(D) = L(\neg E)$ and $|D|$ is at most exponential w.r.t. $|E| + 1$. According to **(c)**, D can be converted into an expression F such that $L(D) = L(\neg E) = L(F)$ and $|F|$ is at most exponential w.r.t. $2^{|E|+1}$. Following [8,7], let us set $E = r_n$. Then $|D|$ must be exponential w.r.t. $|E| + 1$ and $|F|$ must be exponential w.r.t. $2^{|E|+1}$ so that condition **(d)** be satisfied. Therefore any automaton recognizing $L(\neg E_n)$ has at least an exponential size w.r.t. $|E_n| + 1$. □

5 The Partial Derivative Automaton

As in the case of a simple regular expression, the language of an extended regular expression E is recognized by a deterministic automaton the states of which are the partial derivatives of E; moreover this automaton can be computed by applying the subset construction to the derivated term automaton of E.

Definition 4. *Let E be an extended regular expression over the alphabet Σ.* The partial derivative automaton *of E is the deterministic automaton $A' = (\Sigma, Q', q'_0, F', \delta')$ defined by $Q' = \{\frac{\partial}{\partial_w}(E) \mid w \in \Sigma^*\}$, $q'_0 = \{\{E\}\}$, $F' = \{G' \in Q' \mid \varepsilon \in L(G')\}$, $\delta' = \{(G', a, H') \mid \frac{\partial}{\partial_a}(G') = H'\}$.*

Proposition 6. *Let E be an extended regular expression. The partial derivative automaton A' of E is obtained by applying the subset construction to the derivated term automaton A of E.*

Proof. The proof is the same as in the simple regular expression case. Let $A = (\Sigma, Q, q_0, F, \delta)$ and $A' = (\Sigma, Q', q'_0, F', \delta')$. By definition of A', for all $w \in \Sigma^*$, the state $q'_0 \cdot w$ is the partial derivative of E w.r.t. w. Let us consider the automaton $D = (\Sigma, S, s_0, T, \cdot)$ obtained by applying the subset construction to the automaton A. For all word $w \in \Sigma^*$, the deterministic state $s_0 \cdot w$ is associated with the subset $\delta(q_0, w)$ of states of Q. By definition of A, this subset is equal to the union of the derivated terms of E w.r.t. w, that is to the partial derivative of E w.r.t. w. Hence, applying the subset construction to the automaton A yields the automaton A'.

Proposition 7. *Let E be an extended regular expression, B be the derivative automaton of E and A' be the partial derivative automaton of E. The automata B and A' generally are not comparable.*

Proof. A state of B is a derivative of E while a state of A' is a partial derivative of E. Thus it seems natural to define a morphism (for example from B to A') by associating the state $\frac{d'}{d'_u}(E)$ to the state $\frac{\partial}{\partial_u}(E)$. However this correspondence only results in a mapping from B to A' if for all $v \in \Sigma^*$ such that $\frac{d'}{d'_u}(E) = \frac{d'}{d'_v}(E)$, we also have $\frac{\partial}{\partial_u}(E) = \frac{\partial}{\partial_v}(E)$. The reasoning is similar for a morphism from A' to B. We now exhibit extended regular expressions such that there exists no morphism either from B to A' or from A' to B.

(1) Let us consider the expression $E = a(a+\varepsilon)(ba+b)^* + (ba+b)^*$ over the alphabet $\Sigma = \{a, b\}$. It holds:

$$\frac{d'}{d'_a}(E) = (a+\varepsilon)(ba+b)^* \qquad \frac{d'}{d'_b}(E) = (a+\varepsilon)(ba+b)^*$$
$$\frac{\partial}{\partial_a}(E) = \{\{(a+\varepsilon)(ba+b)^*\}\} \quad \frac{\partial}{\partial_b}(E) = \{\{a(ba+b)^*\}, \{(ba+b)^*\}\}.$$

The derivatives of E w.r.t. a and b are equal, although the partial derivatives of E w.r.t. a and b are not equal.

(2) Let us consider the expression $F = (ba^* \cap ba^*)b + aa^*b$ over the alphabet $\Sigma = \{a, b\}$. It holds:

$$\frac{d'}{d'_a}(E) = a^*b \qquad \frac{d'}{d'_b}(E) = (a^* \cap a^*)b$$
$$\frac{\partial}{\partial_a}(E) = \{\{a^*b\}\} \quad \frac{\partial}{\partial_b}(E) = \{\{a^*b\}\}.$$

The partial derivatives of E w.r.t. a and b are equal, although the derivatives of E w.r.t. a and b are not equal.

6 Conclusion

Thanks to an appropriate definition of partial derivatives of complementation and intersection operations, we have generalized the partial derivative computation to extended regular expressions. As a result, and generalizing Antimirov algorithm, we have designed an algorithm for converting an extended regular expression into a non-deterministic automaton. To our knowledge, it is the first algorithm computing such a non-deterministic automaton. Notice that the other boolean operations can be expressed through complementation and intersection operations, for example the set difference $L(E \setminus F) = L(E) \cap \neg L(F)$ or the symmetrical difference $L(E \Delta F) = (L(E) \cap \neg L(F)) \cup (L(F) \cap \neg L(E))$. Thus the partial derivatives of the other boolean operations can easily be processed using the formulae of complementation and intersection operations. For a regular expression with n positions, the number of states of the derivated term automaton is bounded by $n+1$ if the expression is a simple one, whereas it has a worst case exponential complexity (and this bound is tight) if the expression is an extended one. It is an open question whether there exists an extension of the notion of position in an extended regular expression that would generalize the relation between positions and derivated terms of a simple regular expression.

References

1. Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. Theoretical Computer Science 155, 291–319 (1996)
2. Berry, G., Sethi, R.: From regular expressions to deterministic automata. Theoretical Computer Science 48(1), 117–126 (1986)
3. Brzozowski, J.A.: Derivatives of regular expressions. Journal of the Association for Computing Machinery 11(4), 481–494 (1964)
4. Champarnaud, J.M., Ponty, J.L., Ziadi, D.: From regular expressions to finite automata. International Journal of Computational Mathematics 72, 415–431 (1999)
5. Champarnaud, J.M., Ziadi, D.: Canonical derivatives, partial derivatives, and finite automaton constructions. Theoretical Computer Science 239(1), 137–163 (2002)
6. Ellul, K., Krawetz, B., Shallit, J., Wang, M.: Regular expressions: New results and open problems. Journal of Automata, Languages and Combinatorics 10(4), 407–437 (2005)
7. Gelade, W.: Succinctness of regular expressions with interleaving, intersection and counting. Theoretical Computer Science 411(31-33), 2987–2998 (2010)
8. Gelade, W., Neven, F.: Succinctness of the complement and intersection of regular expressions. In: Albers, S., Weil, P. (eds.) STACS. Dagstuhl Seminar Proceedings, vol. 08001, pp. 325–336 (2008)
9. Glushkov, V.M.: The abstract theory of automata. Russian Mathematical Surveys 16, 1–53 (1961)
10. Kleene, S.: Representation of events in nerve nets and finite automata. Automata Studies Annual Mathematical Studies 34, 3–41 (1956)
11. Leiss, E.: Constructing a finite automaton for a given regular expression. SIGACT News 12, 81–87 (1980)

12. McNaughton, R.F., Yamada, H.: Regular expressions and state graphs for automata. IEEE Transactions on Electronic Computers 9, 39–57 (1960)
13. Myhill, J.: Finite automata and the representation of events. Wright Air Development Command Technical Report 57-624, 112–137 (1957)
14. Nerode, A.: Linear automata transformation. In: Proceedings of AMS, vol. 9, pp. 541–544 (1958)
15. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development 3(2), 115–125 (1959)

Automatic Learning of Subclasses of Pattern Languages

John Case[1], Sanjay Jain[2,⋆], Trong Dao Le[2], Yuh Shin Ong[2], Pavel Semukhin[3,⋆⋆], and Frank Stephan[2,4,⋆⋆⋆]

[1] Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716-2586, USA
case@cis.udel.edu
[2] Department of Computer Science, National University of Singapore, Singapore 117417, Republic of Singapore
sanjay@comp.nus.edu.sg, daole@nus.edu.sg, yuhshin@gmail.com
[3] Department of Computer Science, University of Regina, Canada
pavel@semukhin.name
[4] Department of Mathematics, National University of Singapore, Singapore 119076, Republic of Singapore
fstephan@comp.nus.edu.sg

Abstract. Automatic classes are classes of languages for which a finite automaton can decide membership question for the languages in the class, in a uniform way, given an index for the language. For alphabet size of at least 4, every automatic class of erasing pattern languages is contained, for some constant n, in the class of all languages generated by patterns which contain (1) every variable only once and (2) at most n symbols after the first occurrence of a variable. It is shown that such a class is automatically learnable using a learner with long-term memory bounded by the length of the first example seen. The study is extended to show the learnability of related classes such as the class of unions of two pattern languages of the above type.

1 Introduction

The present work carries on investigations of learnability properties in connection with automatic structures. The underlying model of learnability is inductive inference [2,9,18]. Additionally, (1) the target class of languages for learning is an automatic family [10,12,13], that is, membership problem for the class to be learnt can be recognised by a finite automaton in a uniform way, and (2) the learner itself is automatic [11]. These learners are given by a function, where in each step, the learner outputs a hypothesis and updates its long term memory based on its previous memory and a current input. This function is required to

⋆ Supported in part by NUS grants C252-000-087-001 and R252-000-420-112.
⋆⋆ Supported by NUS grant R146-000-114-112.
⋆⋆⋆ Supported in part by NUS grants R146-000-114-112 and R252-000-420-112.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 192–203, 2011.

be regular, that is it must be recognised by a finite automaton. Such learners may be considered to be more realistic than learners which have access to all past data. A further motivation for studying learners which are automatic is that in some situations (such as space exploration by robots), it may be more reasonable to have finite automata as a model rather than Turing machines. Another motivation for the work goes back to the programme of Khoussainov and Nerode [13] to find which results from recursion theory can be transferred to automata theory and automatic structures.

Learners with explicit bounds on the long term memory have already been studied previously (in the general setting of algorithmic learners), see [8,14]. In the current paper, we consider learners for which the update function of the learner is automatic. We will mainly be concentrating on learning subclasses of pattern languages [1] and related classes which are automatic.

Section 2 below gives the preliminaries related to the model (for both learning and automatic classes) used in this paper. Section 3 deals with the learnability properties of certain concrete classes, namely various interesting automatic classes of pattern languages. The basic class $\mathcal{P}_n$ consists of all regular pattern languages[1] which are generated by regular patterns in which variables occur only within the last n symbols of the pattern. Each class $\mathcal{P}_n$ forms an automatic family [12] and Theorem 3 shows that each class $\mathcal{P}_n$ is learnable by an automatic learner where the long term memory is bounded by the size of the hypothesis. Further results in this section investigate the learnability of related classes such as the class of all (disjoint) unions of two members of $\mathcal{P}_n$. Section 4 deals with learnability of character pattern languages, where the variables are allowed to be replaced only by one character.

2 The Model

The *convolution* of two strings $x, y \in \Sigma^*$ (denoted $conv(x, y)$) is the string $(x(0), y(0)), (x(1), y(1)), \ldots, (x(n-1), y(n-1))$, where each pair is a symbol from $(\Sigma \cup \{\diamond\})^2$. The special symbol $\diamond$ is appended to the shorter string in order to make both strings to be of the same length $n = \max\{|x|, |y|\}$. Similarly, one can define *conv* on multiple arguments. A relation R or a function f is called *automatic* if the sets $\{\text{conv}(x_1, x_2, \ldots, x_n) : R(x_1, x_2, \ldots, x_n)\}$ and $\{\text{conv}(x_1, x_2, \ldots, x_m, y) : f(x_1, x_2, \ldots, x_m) = y\}$, respectively, are regular. Some examples of automatic predicates from the prior literature include predicates to compare the length of strings, the lexicographic order and the length-lexicographic order. Here x is *length-lexicographically less than* y iff

[1] Angluin's *pattern languages* [2] are those generated by all the positive length substitution instances in a pattern, such as, for example, `01xy200zx1` — where the variables (for substitutions) are x, y, z and the constants/terminals are $0, 1, 2$. In the present work, variables are also permitted to be substituted by empty strings. Shinohara [23] introduced *regular pattern languages* as those languages which are generated by a pattern where each variable occurring, occurs only once. These language classes have been well-studied and found various applications.

either $|x| < |y|$ or $|x| = |y|$ and x is lexicographically before y, where $|x|$ denotes the length of string x.

A family of languages $\{L_\alpha : \alpha \in I\}$, where each $L_\alpha \subseteq D$, is said to be automatic iff D and I are regular sets (over some finite alphabet Σ and Γ respectively) and the set $\{\text{conv}(\alpha, x) : \alpha \in I \wedge x \in D \wedge x \in L_\alpha\}$ is regular. The sets D and I above are respectively called the domain and index domain for the automatic family. It can be shown that the functions and relations which are first-order definable using an automatic family and other automatic parameters, is again automatic [10,13]. Automatic structures are structures given by finitely many automatic relations and functions — where these structures can also be considered in a more general sense, when they are just isomorphic to a collection of finitely many automatic predicates and functions with corresponding regular domains as defined above.

Properties such as decidability of first order theory make automatic structures a useful tool not only in learning theory but also in other areas such as model checking and Boolean algebras [6,13,21,22]. Moreover, though the class of all regular languages is learnable using queries [4], it is not learnable under the usual inductive inference criteria from positive data [2,9]. Therefore, it is interesting to investigate which subclasses of regular languages are learnable from positive data and which are not. For example, Angluin [3] considered learnability of the class of k-reversible languages. These studies were later extended [7]. In this context, it is useful to consider which automatic families are learnable and which not.[2]

The present work considers learning in the setting of automatic structures. The learning task (also called target class) is a class of languages, $\mathcal{L} = \{L_\alpha : \alpha \in I\}$ over a domain $D \subseteq \Sigma^*$, where I is the index domain. The learner uses a hypothesis space $\mathcal{H} = \{H_\beta : \beta \in J\}$ to express its conjectures (here J is the index domain for the hypothesis space). For this paper both the target class as well as the hypothesis space are automatic families. A *text* T is a mapping from $\{0, 1, 2, \ldots\}$ to $D \cup \{\#\}$. Here # denotes pauses in the presentation of data. The content of a text T, denoted $content(T)$, is $range(T) - \{\#\}$. A text T is for a language L iff $content(T) = L$. We let σ range over initial segments of texts.

A *learner* learning a target language $L \in \mathcal{L}$ from a text T for L, starts (at time 0) with an initial memory (say mem_0) and an initial hypothesis (say hyp_0). At time $t+1$, the learner gets the input $T(t)$, outputs hypothesis hyp_{t+1} and has new memory mem_{t+1}. Here mem_{t+1} and hyp_{t+1} are computed based on just $T(t)$ and mem_t (note that the learner does not have access to t, unless it remembers it as part of its memory). Here we allow hyp_t to be ?, to denote that the learner does not change its previous hypothesis (this is useful for some memory limited models of learner). The memory of the learner is often also referred to as long term memory of the learner.

[2] As noted by Jain, Luo and Stephan [11], even the class of 0-reversible languages is not automatic; however, as mentioned in the abstract and as will be seen below, some very nice classes of regular languages are automatic classes and learnable automatically, that is, by learners which are given using finite automata.

A learner is called *iterative* – see [24] – iff at every stage of the learning algorithm, the memory of the learner is identical to the current hypothesis of the learner.

A learner M is called *automatic learner* iff the mapping from $(mem_t, T(t)) \to (mem_{t+1}, hyp_{t+1})$ is automatic.

We say that a learner M *learns* a language L from a text T for L iff, on input text T, the sequence of hypothesis $hyp_0, hyp_1, \ldots$ of the learner M converges syntactically to a conjecture $\beta \in J$ such that $H_\beta = L$. Here, we say that the sequence $hyp_0, hyp_1, \ldots$ of hypotheses converges to β iff there exists a t such that (1) $hyp_t = \beta$ and (2) for all $t' \geq t$, $hyp_{t'} \in \{\beta, ?\}$. The learner M learns $\mathcal{L}$ iff it learns every member $L \in \mathcal{L}$ from each text T for L. A class $\mathcal{L}$ is said to be *learnable* iff some learner learns $\mathcal{L}$.

Note that the hypothesis space must contain the family $\mathcal{L}$ to be learnt. When we do not restrict the memory size or computational power of the learner, the above learning model is equivalent to Gold's model of inductive inference [9] (called explanatory learning). Based on a result of Angluin [2] characterising algorithmic learnability of *general* indexed classes, Proposition 1 below characterises the general *algorithmic* learnability of automatic families.[3] Note that the version of Angluin's condition for automatic classes, as used in Proposition 1, can be checked explicitly for automatic families. Hence it is decidable whether an automatic family is learnable by an *algorithmic* learner or not. In what follows, for simplicity, the tell-tale condition will be referred to as Angluin's.

Proposition 1 (Based on Angluin [2]). *An automatic family $\{L_\alpha : \alpha \in I\}$ is learnable by a recursive learner iff, for every $\alpha \in I$, there is a bound b_α such that, for all $\beta \in I$, the implication*

$$[\{x \in L_\alpha : |x| \leq b_\alpha\} \subseteq L_\beta \subseteq L_\alpha] \Rightarrow L_\beta = L_\alpha.$$

holds. One calls the set $\{x \in L_\alpha : |x| \leq b_\alpha\}$ a tell-tale set for L_α. This condition is called Angluin's tell-tale condition. *Note that one can take $b_\alpha = |\alpha| + c$ for a suitable constant c independent of α (see [12]).*

Angluin's condition solves the question of algorithmic learnability of automatic classes. Therefore, for learning automatic families, it is more interesting to consider automatic learners which have a superior run-time behaviour than usual learners as hypothesis and updated memory of automatic learners can be computed in time *linear in the length of the previous memory and current datum*; this is explained in the following remark.

Remark 2. Any automatic function f can be computed in linear time. To see this, construct a directed acyclic graph (branching program), with root being the starting state of the automaton recognising f. The acyclic graph has, at each depth, nodes representing each state of the automaton. The depth of the acyclic graph is bounded by $|x| + c$ (note that length of $f(x)$ is bounded by $|x| + c$, for

[3] Note that herein the focus will, nonetheless, remain primarily on the *automatic* learnability of automatic classes and not on their general algorithmic learnability.

some constant c). The acyclic graph has transitions from all nodes at depth d to all nodes at depth $d+1$, for each possible symbol in the alphabet used by the automaton for recognising $\mathrm{conv}(x, f(x))$. In a first pass over the acyclic graph from the root node to the end, one marks all reachable nodes when the input to f is x (and $f(x)$ is allowed all possible values); this can be done in linear time. In the second pass one identifies the unique reachable accepting node which is on level $\ell \geq |x|$. In the third pass, one goes from this accepting node backwards through the graph of reachable nodes and notes down the members from $f(x)\diamond^{\ell-|f(x)|}$ in reverse order. As the computation is unique, it does not matter which choice one takes in the nodes as long as one remains within the subgraph consisting of the reachable nodes. This allows one to compute the output of the function in linear time.

Automatic learners cannot memorise all data they observe; hence the learner can no longer access the full past history of the data seen so far. Thus, in general, the requirement of a learner to be automatic is a real restriction and learners cannot be made automatic by just applying Pitt's delaying technique [19].

Long term memory limitations were first introduced by Freivalds, Kinber and Smith [8]. The variations of long term memory in the context of automatic learners, was considered by Jain, Luo and Stephan [11]. The size of the memory of a learner may be explicitly bounded in length. The length-restrictions we consider are:

(1) memory bounded by the size of the longest datum observed so far plus a constant, that is $|mem_t| \leq max(\{|T(s)| : s < t\}) + c$, for some constant c;
(2) memory bounded by hypothesis size plus a constant, that is $|mem_t| \leq |hyp_t| + c$, for some constant c.

For the ease of notation, the "plus a constant" is omitted in the notations below. Note that the learner is not constrained regarding which alphabet it uses for its memory. Therefore, the learner might, for example, store the convolution of up to k examples (in case the memory does not exceed the allowed bound). Note that, in the case that memory is unbounded or the bound allows storage of the hypothesis, then the learner can memorise the most recent hypothesis output, and, thus, abstain from outputting ?.

For many learning paradigms of automatic learning, one can choose the hypothesis space $\mathcal{H}$ to be same as $\mathcal{L}$. However, when the amount of the memory allowed to the learner depends on the size of the hypothesis or when the long term memory of the learner has to be the most recent hypothesis, as in the case of iterative learning, this requirement may be a restriction. The main reason for hypothesis space not to be critical in many cases is that one can automatically convert the indices from one automatic family to another for the languages which are common to both automatic families. Only in the case of iterative learning and bounds given by the size of the hypothesis, it is often important to have the ability to store some additional information into the hypothesis — which is impossible in the case of a one-one hypothesis space. For example, Theorem 3 requires a special class preserving hypothesis space, if one considers learning these

classes using memory bounded by hypothesis size or for iterative learning. Here a hypothesis space $\{H_\beta : \beta \in J\}$ is called class preserving (class comprising) [16] for the target class $\{L_\alpha : \alpha \in I\}$, if $\{L_\alpha : \alpha \in I\} = \{H_\beta : \beta \in J\}$ (respectively, $\{L_\alpha : \alpha \in I\} \subseteq \{H_\beta : \beta \in J\}$).

Note that, in contrast, hypothesis spaces do matter for learning general indexed families by recursive learners [16,17].

In the case that the hypothesis space does not matter, often, for the ease of notation, the languages are given in place of the indices as conjectures of the learner.

3 Classes of Pattern Languages

Learning theorists have studied the learnability of the class of pattern languages extensively [1,15,20,23] and they are often used to judge the power of learning models. Although the full generality of pattern languages cannot be brought over into an automatic setting, there are still rich automatic classes of pattern languages which deserve to be investigated [12].

Fix an alphabet set Σ. The notion of pattern languages is based on the notion of a pattern [1] which is any string over $(\Sigma \cup V)^*$, where V is an infinite set of variables which is disjoint from Σ. The language associated with a pattern π, denoted by $Lang(\pi)$, is the set of strings which can be obtained by replacing each variable in the pattern by a string over Σ. There are two cases: In the case of an erasing pattern language one permits that the string chosen can be empty; in the case of a non-erasing pattern language the string must always contain at least one symbol. In the present work, a "pattern language" is by default an "erasing pattern language".

Shinohara [23] considered the class of languages which are generated by regular patterns, that is, patterns in which the variables do not repeat. Let $\mathcal{P}_n$ denote the class of pattern languages which can be generated by a regular pattern where variables, if any, only appear within the last n symbols of the pattern. For example, the pattern $010232012012x12y1z$ generates a language in $\mathcal{P}_6$. Jain, Ong, Pu and Stephan [12] showed that every class $\mathcal{P}_n$ can be given as an automatic family. Furthermore, every automatic class of languages generated by regular patterns is a subclass of some $\mathcal{P}_n$.

Theorem 3. *Every class $\mathcal{P}_n$ has an automatic learner. This learner can be made iterative, if one allows a suitable class preserving hypothesis space. Furthermore, there is also an automatic learner with the memory limited by the size of the first datum seen.*

Proof. Let S be the set of all $L \in \mathcal{P}_n$ which are generated by regular patterns consisting of at most n symbols. Note that S is finite. The memory of the learner is of the form $\text{conv}(x, \alpha)$, where $|\alpha| = |x| + 1$ and every symbol g in α codes a subset of S. The conjecture of the learner at any stage is the language associated with the memory defined as follows. Let A be the set of all languages L such that, for some prefix y of x and language H in the set coded by $\alpha(|y|)$, $L = y \cdot H$.

The language associated with memory $\mathrm{conv}(x, \alpha)$ is the length-lexicographically first language (based on the length-lexicographic ordering of (y, H) for the languages $y \cdot H$ in A) in A such that no proper subset of this language is in A.

The automatic learner conjectures ? until it sees the first datum $x \neq \#$. From then on its memory is of the form $\mathrm{conv}(x, \alpha)$ and its conjecture is the language associated with the above memory, where α is appropriately chosen: For every prefix y of x, $\alpha(|y|)$ contains all $H \in S$ such that $y \cdot H$ contains all the data seen so far.

It is easy to verify that the above learner is automatic. Furthermore, the memory of the learner is bounded by the size of the first datum seen. Note that the language associated with $\mathrm{conv}(x, \alpha)$ is the minimal language in $\mathcal{P}_n$ which contains all the data seen so far. Hence, the learner converges to a correct hypothesis.

The above learner can be made iterative, by using a class preserving hypothesis space, which allows one to pad the hypothesis by the memory $\mathrm{conv}(x, \alpha)$ — note that the memory value of the learner above converges in the limit. □

The above algorithm can be used to obtain the following more involved result (Theorem 6). The result mainly implies that the class of the disjoint unions of two languages from $\mathcal{P}_n$ is learnable.

Bārzdiņš [5] called a learner *consistent* if the learner, on any input σ, outputs a hypothesis which contains every data-item occurring in σ. A learner is said to be *confident* [18] if for all input texts the sequence of hypotheses output by the learner syntactically converges to a hypothesis. Note that these constraints are required even for input texts for a language outside the class to be learnt.

Proposition 4. *Let an automatic class $\mathcal{L}$ be given. Suppose that the automatic learners $M_1, M_2, \ldots, M_n$ are consistent and confident. Then, there exists another automatic learner N which is (1) consistent, (2) confident and (3) converges on a text T for a language $L \in \mathcal{L}$ to an index for L whenever at least one of the learners $M_1, M_2, \ldots, M_n$ converges on T to an index for L.*

Note that the learner N in the above proposition would have its memory bounded by the size of the longest datum seen, if each of the individual learner $M_1, M_2, \ldots$ satisfy this constraint. Similarly, if one allows class preserving hypothesis space, then N can be made to satisfy memory bounded by hypothesis size if each of $M_1, M_2, \ldots$ satisfy this constraint.

Proposition 5. *$\{L \cup \{z\} : L \in \mathcal{P}_n, z \in \Sigma^* - L\}$ is consistently and confidently learnable by an automatic learner. Furthermore, the memory of the learner is bounded by the size of the longest input datum seen.*

Theorem 6. *For every n, the class $\mathcal{P}_n \cup \{L \cup H : L, H \in \mathcal{P}_n \wedge L \cap H = \emptyset\}$ has an automatic learner. Furthermore, this learner is consistent and confident, and has memory bounded by the size of the longest datum seen by the learner.*

Proof. Note that two non-singleton pattern languages L and H in $\mathcal{P}_n$ are disjoint iff there exists a w and different $a, b \in \Sigma$ such that either (1) $L \subseteq wa\Sigma^*$ and $H \subseteq wb\Sigma^*$ or (2) length of w is at most n and $L \subseteq \Sigma^* aw$ and $H \subseteq \Sigma^* bw$.

The automatic learner N for this theorem maintains the long term memory $\mathrm{conv}(\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, xa, xb, cy, dy)$. The learner uses 6 sublearners N_0, N_1, N_2, N_3, N_4, N_5 using parts of this memory which simulate the learner M from Theorem 3 and the learner from Proposition 5 as follows.

- The learner N_0 simulates M on all inputs and β_0 is the corresponding memory.
- The learner N_1 simulates the learner from Proposition 5 and uses memory β_1.
- Let x be the longest common prefix of all data observed so far. Furthermore, if possible, let a, b be different elements of Σ such that each observed datum either extends xa or extends xb; otherwise, if such elements of Σ do not exist, then let $a = \varepsilon$ and $b = \varepsilon$.

  N_2 and N_3 simulate M on the data extending xa and xb, respectively, maintaining the corresponding memories β_2 and β_3 — except that in the case of $a = b = \varepsilon$, the conjectures of N_2 and N_3 are Σ^*, rather than the conjecture of M. Now consider the case that, due to a new datum, the values xa and xb become updated and $a \neq b$. Then, a is chosen such that all previously observed data extend xa, while the current datum extends xb. Then N_2 takes over the old memory of N_0 (before processing the current datum) and simulates M on all data which extend xa, while N_3 simulates M on all data extending xb, among which the current datum is the first to occur. The long term memories β_2 and β_3 of N_2 and N_3 are maintained accordingly. Note that when $a \neq b$, then the above process consistently maintains that N_2 and N_3 simulate M on all data which extend xa and xb respectively.
- Let y be the longest common suffix, of length at most $n + 1$, of all data observed so far. Furthermore, if possible, let c, d be different elements of Σ such that all observed data have suffix either cy or dy (where length of cy and dy are at most $n + 1$); otherwise let $c = d = \varepsilon$.

  N_4 and N_5 simulate M on the data ending with cy and dy, respectively, maintaining the corresponding memories β_4 and β_5 — except that in the case of $c = d = \varepsilon$, the conjectures of N_4 and N_5 are Σ^*, rather than the conjecture of M.

  Now consider the case that, due to a new datum, the values cy and dy become updated and $c \neq d$. Then, c is chosen such that all previously observed data end with cy, while the current datums ends with dy. Then N_4 takes over the old memory of N_0 (before processing the current datum) and simulates M on all data which end with cy, while N_5 simulates M on all data ending with dy, among which the current datum is the first to occur. The long term memories β_4 and β_5 of N_4 and N_5 are maintained accordingly. Note that when $c \neq d$, then the above process consistently maintains that N_4 and N_5 simulate M on all data which end with cy and dy, respectively.

The new learner N is a combination of the learners N_0, N_1, N_2, N_3, N_4 and N_5. When its current memory is updated to $\mathrm{conv}(\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, xa, xb, cy, dy)$, then N's hypothesis is calculated from the conjectures $L_0, L_1, L_2, L_3, L_4, L_5$ of $N_0, N_1, N_2, N_3, N_4, N_5$ as follows: L is the first language in the list L_0, L_1, $L_2 \cup L_3$

and $L_4 \cup L_5$ which is not a proper superset of any other language in the list. It is easy to see that the learner N defined like this is automatic.

For the verification, consider first the case that the language L to be learnt is from $\mathcal{P}_n$. Then N_0 learns L and the other learners converge to a superset of L. If the language to be learnt is of the form $L \cup \{z\}$ with $L \in \mathcal{P}_n$ and $z \notin L$, then N_1 converges to the correct hypothesis and all other learners converge to hypotheses for some supersets of $L \cup \{z\}$.

Otherwise, consider disjoint sets $L, H \in \mathcal{P}_n$ such that $|L|, |H| \geq 2$ and the target language is $L \cup H$. Furthermore, let x', a', b', y', c', d' be the final values of the variables x, a, b, y, c, d, respectively. Note that a, b, c, d can change their value only by either becoming the empty string or by x or y becoming shorter. This shows that there are at most 2 times as many updates of these variables as the length of the first string observed. Note that either $L \subseteq x'a'\Sigma^* \wedge H \subseteq x'b'\Sigma^* \wedge a' \neq b'$ or $L \subseteq \Sigma^*c'y' \wedge H \subseteq \Sigma^*d'y' \wedge c' \neq d'$ (where, L and H may be interchanged). Furthermore, if $a' \neq b'$, then N_2 and N_3 converge to L and H, respectively. Similarly, if $c' \neq d'$, then N_4 and N_5 converge to L and H, respectively.

The invariant of the learning process is that at every time, N_0 and N_1 conjecture sets containing all data seen so far, N_2 conjectures a set containing all data beginning with xa seen so far, N_3 conjectures a set containing all data beginning with xb seen so far, N_4 conjectures a set containing all data ending with cy seen so far and N_5 conjectures a set containing all data ending with dy seen so far. Furthermore, all learners converge. In the limit, one of the following occurs:

- N_0 converges to a language generating the target language;
- N_1 converges to a language generating the target language;
- N_2 converges to a language consisting of all the strings of the target language beginning with $x'a'$ and N_3 converges to a language consisting of all strings of the target language beginning with $x'b'$;
- N_4 converges to a language consisting of all the strings of the target language ending with $c'y'$ and N_5 converges to a language consisting of all strings of the target language ending with $d'y'$.

Hence, one of the languages $L_0, L_1, L_2 \cup L_3, L_4 \cup L_5$ as computed by the learners above, in the limit, will be same as the target language and be a subset of the other three languages; therefore, the learner N will converge to the correct language. □

We now consider the general case of learning unions of pattern languages from $\mathcal{P}_n$. While the above results also hold for non-erasing pattern languages, the following results of this section hold only for erasing pattern languages.

Proposition 7. *Suppose $L_0, L_1, \ldots, L_k$ are erasing pattern languages generated by regular patterns. If Σ contains at least $k+1$ characters, L_0 is infinite and the difference $L_0 - \bigcup_{i \in \{1,2,\ldots,k\}} L_i$ is not empty, then this difference is infinite.*

Let $\mathcal{G}_n$ denote the set of all erasing pattern languages which are generated by a regular pattern of length at most n which start with a variable. Note that $\mathcal{G}_n$ is finite.

Theorem 8. *Suppose* $|\Sigma| \geq 3$. *Let* $\mathcal{L} = \{L_1 \cup L_2 : L_1, L_2 \in \mathcal{P}_n, |L_1| > 1, |L_2| > 1\}$. *Then,* $\mathcal{L}$ *is learnable by an automatic learner (using class comprising hypothesis space) which (1) uses memory bounded by the size of the longest datum seen so far and (2) for all texts* T *for a language* L, *converges to an index for a language* L' *such that* $L - L'$ *is finite.*

Proof. Note that by Proposition 7 different languages in $\mathcal{L}$ are pairwise infinitely different.

For each $H_0, H_1 \in \mathcal{G}_n$, consider M_{H_0,H_1} defined as follows. Conjectures of M_{H_0,H_1} will be of the form $conv(x, s_0, s_1, c)$, where $x, s_0, s_1 \in \Sigma^*, c \in \{0, 1, 2\}$. The language associated with $conv(x, s_0, s_1, 0)$ is Σ^*. The language associated with $conv(x, s_0, s_1, 1)$ is $s_0 \cdot H_0$. The language associated with $conv(x, s_0, s_1, 2)$ is $s_0 \cdot H_0 \cup s_1 \cdot H_1$. The string x will be the length-lexicographically smallest string seen in the input.

Whenever M_{H_0,H_1} sees an input w which is length-lexicographically smaller than any previously seen input, it outputs $conv(w, s_0, s_1, c)$, where if there is a prefix s of w such that w is the length lexicographically least element of $s \cdot H_0$, then $s_0 = s$ and $c = 1$; otherwise, $s_0 = w$ and $c = 0$. In both cases, s_1 is ε.

In other cases, suppose the previous conjecture of M_{H_0,H_1} is (x, s_0, s_1, c) and the new input is w. Then, use the first case below which applies:

Case 1: $c = 0$ or $(w \in s_0 \cdot H_0)$. In this case conjecture (x, s_0, s_1, c).
Case 2: $w \notin H_1$. In this case conjecture $(x, s_0, s_1, 0)$.
Case 3: $c = 1$. In this case let s_1' be the longest prefix of w such that $w \in s_1' \cdot H_1$, and conjecture $(x, s_0, s_1', 2)$.
Case 4: $c = 2$. In this case let s_1' be longest prefix of s_1 such that $w \in s_1' \cdot H_1$, and conjecture $(x, s_0, s_1', 2)$.

Note that, on all input texts, M_{H_0,H_1} converges. Say the converged index is (x, s_0, s_1, c). Then, this is either a conjecture for Σ^*, or $s_0 \cdot H_0$ contains the length-lexicographically smallest string in the input and (x, s_0, s_1, c) represents the language which is a superset of the input language, except maybe for finitely many strings.

Furthermore, if the input language is $L = s \cdot H \cup s' \cdot H'$, for $H, H' \in \mathcal{G}_n$, (where $s \cdot H$ contains the length-lexicographically smallest string in L) then the following two statements hold:

(a) for each H_0, H_1, M_{H_0,H_1} converges to a superset of L, as this converged language is a superset of a finite variant of L, and thus by Proposition 7 a superset of L.

(b) $M_{H,H'}$ converges to an index for a subset of L (and thus by (a) above to an index for L). To see this, suppose $M_{H,H'}$ converges to index (x, s_0, s_1, c). Then x is the lexicographically least element of L, $s \cdot H$ and $s_0 \cdot H$. Thus $s = s_0$. Furthermore, if $c = 2$, we have that $s' \cdot H' \supseteq s_1 \cdot H'$ (since in Case 3 and 4 the algorithm chooses the longest possible prefix).

Now define M which outputs the convolution of the outputs of all $M_{H,H'}$, $H, H' \in \mathcal{G}_n$. This conjecture represents language associated with the conjecture of $M_{R,R'}$, where R, R' are chosen to be length-lexicographically least such that

the conjecture of $M_{R,R'}$ is not a proper subset of the conjecture of any other $M_{R'',R'''}$. It follows using (a) and (b) that M learns $\mathcal{L}$. □

A combination of the constructions from Proposition 5 and Theorem 8 can be used to show the following theorem.

Theorem 9. *Suppose* $|\Sigma| \geq 3$. *Let* $\mathcal{L} = \{L_1 \cup L_2 : L_1, L_2 \in \mathcal{P}_n\}$. *Then,* $\mathcal{L}$ *is learnable by a learner (using class comprising hypothesis space) which uses memory bounded by the size of the longest datum seen so far. Furthermore, the learner, for all texts* T *for a language* L*, converges to an index for a language* L' *such that* $L - L'$ *is finite.*

4 Character Variables

In this section, we consider the following modification of pattern languages. We consider two types of variables: character variables which can be replaced by one symbol of Σ and string variables which can be replaced by any string, including the empty string. Note that one can simulate non-erasing pattern languages (as studied by Angluin [1]) by putting one character variable followed by one string variable. The non-erasing pattern language associated with pattern $xyxz$ can be proven to be regular, by chosing the equivalent pattern $abyacz$ of character variables a, b, c and erasing string variables y, z.

When investigating the learnability properties of such pattern languages, it turned out that the character variables make it very difficult to build automatic learners. For that reason, the case where only character variables are allowed is considered here.

Let $\mathcal{O}_n$ denote the class of languages formed using patterns with variables only of the type character variable such that, between the first and last occurrence of any variable, at most n distinct variables appear in the pattern.

Theorem 10. (1) *Each* $\mathcal{O}_n$ *is learnable by an automatic learner with memory bounded by the size of the longest datum seen so far in the input.*
(2) *The class* $\{L \cup H : L \cap H = \emptyset \wedge L, H \in \mathcal{O}_0\}$ *is not automatically learnable.*

Acknowledgments. We thank the anonymous referees for useful comments.

References

1. Angluin, D.: Finding patterns common to a set of strings. Journal of Computer and System Sciences 21, 46–62 (1980)
2. Angluin, D.: Inductive inference of formal languages from positive data. Information and Control 45, 117–135 (1980)
3. Angluin, D.: Inference of reversible languages. Journal of the ACM 29, 741–765 (1982)
4. Angluin, D.: Learning regular sets from queries and counter-examples. Information and Computation 75, 87–106 (1987)

5. Bārzdiņš, J.: Inductive inference of automata, functions and programs. In: Proceedings of the 20th International Congress of Mathematicians, Vancouver, pp. 455–460 (1974) (in Russian; English translation in American Mathematical Society Translations: Series 2, 109:107-112, 1977)
6. Blumensath, A., Grädel, E.: Automatic structures. In: 15th Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 51–62. IEEE Computer Society, Los Alamitos (2000)
7. Fernau, H.: Identification of function distinguishable languages. Theoretical Computer Science 290, 1679–1711 (2003)
8. Freivalds, R., Kinber, E., Smith, C.: On the impact of forgetting on learning machines. Journal of the ACM 42, 1146–1168 (1995)
9. Gold, E.M.: Language identification in the limit. Information and Control 10, 447–474 (1967)
10. Hodgson, B.R.: Décidabilité par automate fini. Annales des sciences mathématiques du Québec 7, 39–57 (1983)
11. Jain, S., Luo, Q., Stephan, F.: Learnability of automatic classes. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 321–332. Springer, Heidelberg (2010)
12. Jain, S., Ong, Y.S., Pu, S., Stephan, F.: On automatic families. Technical Report TRB1/10, School of Computing, National University of Singapore (2010)
13. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995)
14. Kinber, E., Stephan, F.: Language learning from texts: Mind changes, limited memory and monotonicity. Information and Computation 123, 224–241 (1995)
15. Lange, S., Wiehagen, R.: Polynomial time inference of arbitrary pattern languages. New Generation Computing 8, 361–370 (1991)
16. Lange, S., Zeugmann, T.: Incremental learning from positive data. Journal of Computer and System Sciences 53, 88–103 (1996)
17. Lange, S., Zeugmann, T., Zilles, S.: Learning indexed families of recursive languages from positive data: A survey. Theoretical Computer Science 397, 194–232 (2008)
18. Osherson, D., Stob, M., Weinstein, S.: Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists. MIT Press, Cambridge (1986)
19. Pitt, L.: Inductive inference, DFAs, and computational complexity. In: Jantke, K.P. (ed.) AII 1989. LNCS (LNAI), vol. 397, pp. 18–44. Springer, Heidelberg (1989)
20. Reidenbach, D.: A non-learnable class of E-pattern languages. Theoretical Computer Science 350, 91–102 (2006)
21. Rubin, S.: Automatic Structures. PhD thesis, The University of Auckland (2004)
22. Rubin, S.: Automata presenting structures: a survey of the finite string case. The Bulletin of Symbolic Logic 14, 169–209 (2008)
23. Shinohara, T.: Polynomial time inference of extended regular pattern languages. In: Goto, E., Furukawa, K., Nakajima, R., Nakata, I., Yonezawa, A. (eds.) RIMS 1982. LNCS, vol. 147, pp. 115–127. Springer, Heidelberg (1983)
24. Wiehagen, R.: Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. Elektronische Informationsverarbeitung und Kybernetik (EIK) 12, 93–99 (1976)

Finite Orbits of Language Operations

Émilie Charlier[1], Mike Domaratzki[2], Tero Harju[3], and Jeffrey Shallit[1]

[1] University of Waterloo, Waterloo, ON N2L 3G1 Canada
{echarlier@,shallit@cs.}uwaterloo.ca
[2] University of Manitoba, Winnipeg, MB R3T 2N2 Canada
mdomarat@cs.umanitoba.ca
[3] University of Turku, FIN-20014 Turku, Finland
harju@utu.fi

Abstract. We consider a set of natural operations on languages, and prove that the orbit of any language L under the monoid generated by this set is finite and bounded, independently of L. This generalizes previous results about complement, Kleene closure, and positive closure.

1 Introduction

If t, x, y, z are (possibly empty) words with $t = xyz$, we say

- x is a *prefix* of t;
- z is a *suffix* of t; and
- y is a *factor* of t.

If $t = x_1 t_1 x_2 t_2 \cdots x_n t_n x_{n+1}$ for some $n \geq 1$ and some (possibly empty) words t_i, x_j, $1 \leq i \leq n$, $1 \leq j \leq n+1$, then $t_1 \cdots t_n$ is said to be a *subword* of t. Thus a factor is a contiguous block, while a subword can be "scattered".

Let L be a language over the finite alphabet Σ, that is, $L \subseteq \Sigma^*$. We consider the following eight natural operations applied to L:

$$\begin{aligned} k &: L \to L^* \\ e &: L \to L^+ \\ c &: L \to \overline{L} = \Sigma^* - L \\ p &: L \to \mathrm{pref}(L) \\ s &: L \to \mathrm{suff}(L) \\ f &: L \to \mathrm{fact}(L) \\ w &: L \to \mathrm{subw}(L) \\ r &: L \to L^R. \end{aligned}$$

Here

$$\begin{aligned} \mathrm{pref}(L) &= \{x \in \Sigma^* \ : \ x \text{ is a prefix of some } y \in L\}; \\ \mathrm{suff}(L) &= \{x \in \Sigma^* \ : \ x \text{ is a suffix of some } y \in L\}; \\ \mathrm{fact}(L) &= \{x \in \Sigma^* \ : \ x \text{ is a factor of some } y \in L\}; \\ \mathrm{subw}(L) &= \{x \in \Sigma^* \ : \ x \text{ is a subword of some } y \in L\}; \\ L^R &= \{x \in \Sigma^* \ : \ x^R \in L\}; \end{aligned}$$

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 204–215, 2011.

where x^R denotes the reverse of the word x.

We compose these operations as follows:

$$\text{if } x = a_1 a_2 \cdots a_n \in \{k, e, c, p, s, f, w, r\}^*, \text{ then}$$
$$x(L) = a_1(a_2(a_3(\cdots(a_n(L))\cdots))).$$

Thus, for example, $ck(L) = \overline{L^*}$. We also write $\epsilon(L) = L$.

Given two elements $x, y \in \{k, e, c, p, s, f, w, r\}^*$, we write $x \equiv y$ if $x(L) = y(L)$ for all languages L, and we write $x \subseteq y$ if $x(L) \subseteq y(L)$ for all languages L.

Given a subset $S \subseteq \{k, e, c, p, s, f, w, r\}$, we can consider the orbit of languages

$$\mathcal{O}_S(L) = \{x(L) \ : \ x \in S^*\}$$

under the monoid of operations generated by S. We are interested in the following questions: when is this monoid finite? Is the cardinality of $\mathcal{O}_S(L)$ bounded, independently of L?

These questions were previously investigated for the sets $S = \{k, c\}$ and $S = \{e, c\}$ [4,1], where the results can be viewed as the formal language analogues of Kuratowski's celebrated "14-theorem" for topological spaces [3,2]. In this paper we consider the questions for other subsets of $\{k, e, c, p, s, f, w, r\}$. Our main result is Theorem 20 below, which shows finiteness for any subset of these eight operations.

2 Operations with Infinite Orbit

We point out that the orbit of L under an arbitrary operation need not be finite. For example, consider the operation q defined by

$$q(L) = \{x \in \Sigma^* \ : \ x \text{ there exists } y \in L \text{ such that } x \text{ is a proper prefix of } y \ \}.$$

Here by "x is a proper prefix of y", we mean that x is a prefix of y with $|x| < |y|$.

Let $L = \{a^n b^n \ : \ n \geq 1\}$. Then it is easy to see that the orbit

$$\mathcal{O}_{\{q\}}(L) = \{L, q(L), q^2(L), q^3(L), \ldots\}$$

is infinite, since the shortest word in $q^i(L) \ \cap \ a^+b$ is $a^{i+1}b$.

The situation is somewhat different if L is regular:

Theorem 1. *Let q denote the proper prefix operation, and let L be a regular language accepted by a DFA of n states. Then $\mathcal{O}_{\{q\}}(L) \leq n$, and this bound is tight.*

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an n-state DFA accepting L. Note that a DFA accepting $q(L)$ is given by $M' = (Q, \Sigma, \delta, q_0, F')$ where

$$F' = \{q \in Q \ : \ \text{ there exists a path of length} \geq 1 \text{ from } q \text{ to a state of } F \ \}.$$

Reinterpreting this in terms of the underlying transition diagram, given a directed graph G on n vertices, and a distinguished set of vertices F, we are

interested in the number of different sets obtained by iterating the operation that maps F to the set of all vertices that can reach a vertex in F by a path of length ≥ 1. We claim this is at most n. To see this, note that if a vertex v is part of any directed cycle, then once v is included, further iterations will retain it. Thus the number of distinct sets is as long as the longest directed path that is not a cycle, plus 1 for the inclusion of cycle vertices.

To see that the bound is tight, consider the language $L_n = \{\epsilon, a, a^2, \ldots, a^{n-2}\}$, which is accepted by a (complete) unary DFA of n states. Then $q(L_n) = L_{n-1}$, so this shows $|\mathcal{O}_{\{q\}}(L_n)| = n$. □

It is possible for the orbit under a single operation to be infinite even if the operation is (in the terminology of the next section) expanding and inclusion-preserving. As an example, consider the operation of fractional exponentiation, defined by

$$n(L) = \{x^\alpha \ : \alpha \geq 1 \text{ rational } \} = \bigcup_{x \in L} x^+ p(\{x\}).$$

Proposition 2. *Let $L = \{ab\}$. Then the orbit $\mathcal{O}_{\{n\}}(L)$ is infinite.*

Proof. We have $aba^i \in n^i(\{ab\})$, but $aba^i \notin n^j(\{ab\})$ for $j < i$. □

3 Kuratowski Identities

Let $a : 2^{\Sigma^*} \to 2^{\Sigma^*}$ be an operation on languages. Suppose a satisfies the following three properties:

1. L is a subset of $a(L)$ (expanding);
2. If $L \subseteq M$ then $a(L) \subseteq a(M)$ (inclusion-preserving);
3. $a(a(L)) = a(L)$ (idempotent).

Then we say a is a *closure operation*. Examples of closure operations include k, e, p, s, f, and w.

Note that if a, b are closure operations, then their composition ab trivially satisfies properties 1 and 2 above, but may not satisfy property 3. For example, pk is not idempotent, as can be seen by examining its action on $L = \{ab\}$ ($aab \notin pk(L)$, but $aab \in pkpk(L)$).

Lemma 3. *Let $a \in \{k, e\}$ and $b \in \{p, s, f, w\}$. Then $aba \equiv bab \equiv ab$.*

Proof. We prove the result only for $b = p$; the other results are similar.

Since $L \subseteq a(L)$, we get $p(L) \subseteq pa(L)$, and then $ap(L) \subseteq apa(L)$. It remains to see $apa(L) \subseteq ap(L)$.

Any element of $a(L)$ is either ϵ or of the form $t = t_1 t_2 \cdots t_n$ for some $n \geq 1$, where each $t_i \in L$. Then any prefix of t looks like $t_1 t_2 \cdots t_{i-1} p_i$ for some $i \geq 1$, where p_i is a prefix of t_i, and hence in $p(L)$. But each t_i is also in $p(L)$, so this shows

$$pa(L) \subseteq ap(L). \tag{1}$$

Since a is a closure operation, $apa(L) \subseteq aap(L) = ap(L)$.

Similarly, we have $ap(L) \subseteq pap(L)$. Substituting $p(L)$ for L in (1) gives $pap(L) \subseteq app(L) = ap(L)$. □

Lemma 4. *The operations $kp, ks, kf, kw, ep, es, ef$ and ew are closure operations.*

Proof. We prove the result for kp, with the other results being similar. It suffices to prove property 3. From Lemma 3 we have $pkp(L) = kp(L)$. Applying k to both sides, and using the idempotence of k, we get $kpkp(L) = kkp(L) = kp(L)$. □

If a is a closure operation, and c denotes complement, then it is well-known (and shown, for example, in [4]) that $acacaca \equiv aca$. However, we will need the following more general observation, which seems to be new:

Theorem 5. *Let x, y be closure operations. Then $xcycxcy \equiv xcy$.*

Proof. $xcycxcy \subseteq xcy$: We have $L \subseteq y(L)$ by the expanding property. Then $cy(L) \subseteq c(L)$. By the inclusion-preserving property we have $xcy(L) \subseteq xc(L)$. Since this identity holds for all L, it holds in particular for $cxcy(L)$. Substituting, we get $xcycxcy(L) \subseteq xccxcy(L)$. But $xccxcy(L) = xcy(L)$ by the idempotence of x.

$xcy \subseteq xcycxcy$: We have $L \subseteq x(L)$ by the expanding property. Then, replacing L by $cy(L)$, we get $cy \subseteq xcy$. Applying c to both sides, we get $cxcy \subseteq ccy = y$. Applying y to both sides, and using the inclusion-preserving property and idempotence, we get $ycxcy \subseteq yy = y$. Applying c to both sides, we get $cy \subseteq cycxcy$. Finally, applying x to both sides and using the inclusion-preserving property, we get $xcy \subseteq xcycxcy$. □

Remark 6. Theorem 5 would also hold if c were replaced by any inclusion-reversing operation satisfying $cc \equiv \epsilon$.

As a corollary, we get [4,1]:

Corollary 7. *If $S = \{a, c\}$, where a is any closure operation, and L is any language, the orbit $\mathcal{O}_S(L)$ contains at most 14 distinct languages.*

Proof. The 14 languages are given by the image of L under the 14 operations

$$\epsilon, a, c, ac, ca, aca, cac, acac, caca, acaca, cacac, acacac, cacaca, cacacac.$$

□

Remark 8. Theorem 5, together with Lemma 4, thus gives 196 separate identities.

In a similar fashion, we can obtain many kinds of Kuratowski-style identities involving k, e, c, p, s, f, w and r.

Theorem 9. *Let $a \in \{k, e\}$ and $b \in \{p, s, f, w\}$. Then we have the following identities:*

4. $abcacaca \equiv abca$
5. $bcbcbcab \equiv bcab$
6. $abcbcabcab \equiv abcab$

Proof. We only prove the first; the rest are similar. From Theorem 5 we get $acacaca \equiv aca$. Hence $ab(acacaca) \equiv ab(aca)$, or equivalently, $aba(cacaca) \equiv aba(ca)$. Since $aba \equiv ab$ from Lemma 3, we get $abcacaca \equiv abca$. □

4 Additional Identities

In this section we prove some additional identities connecting the operations $\{k, e, c, p, s, f, w, r\}$.

Theorem 10. *We have*

7. $rp \equiv sr$
8. $rs \equiv pr$
9. $rf \equiv fr$
10. $rc \equiv cr$
11. $rk \equiv kr$
12. $rw \equiv wr$
13. $ps \equiv sp \equiv f$
14. $pf \equiv fp \equiv f$
15. $sf \equiv fs \equiv f$
16. $pw \equiv wp \equiv sw \equiv ws \equiv fw \equiv wf \equiv w$
17. $kw \equiv wk$
18. $rkw \equiv kw$
19. $ek \equiv ke \equiv k$
20. $fks \equiv pks$
21. $fkp \equiv skp$
22. $rkf \equiv skf \equiv pkf \equiv fkf \equiv kf$

Proof. All of these are relatively straightforward. To see (20), note that $p(L) \subseteq f(L)$ for all L, and hence $pks(L) \subseteq fks(L)$. Hence it suffices to show the reverse inclusion.

Note that every element of $ks(L)$ is either ϵ or can be written $x = s_1 s_2 \cdots s_n$ for some $n \geq 1$, where each $s_i \in s(L)$. In the latter case, any factor of x must be of the form $y = s_i'' s_{i+1} \cdots s_{j-1} s_j'$, where s_i'' is a suffix of s_i and s_j' is a prefix of s_j. Then $s_i'' s_{i+1} \cdots s_{j-1} s_j \in ks(L)$ and hence $y \in pks(L)$.

Similarly, we have $pkf \equiv pk(ps) \equiv (pkp)s \equiv (kp)s \equiv k(ps) = kf$, which proves part of (22).

Theorem 11. *We have*

23. $pcs(L) = \Sigma^*$ *or* $\emptyset$.
24. *The same result holds for* $pcf, fcs, fcf, scp, scf, fcp, wcp, wcs, wcf, pcw, scw, fcw, wcw$.

Proof. Let us prove the first statement. Either $s(L) = \Sigma^*$, or $s(L)$ omits some word v. In the former case, $cs(L) = \emptyset$, and so $pcs(L) = \emptyset$. In the latter case, we have $s(L)$ omits v, so $s(L)$ must also omit $\Sigma^* v$ (for otherwise, if $xv \in f(L)$ for some x, then $v \in s(L)$). So $\Sigma^* v \subseteq cs(L)$. Hence $pcs(L) = \Sigma^*$.

The remaining statements are proved similarly. □

The following result was proved in [1, Theorems 2 and 3].

Lemma 12. *We have ecece ≡ cece.*

Theorem 13. *Let L be any language.*

25. We have $kckck(L) = ckck(L) \cup \{\epsilon\}$.

Proof. First, suppose $\epsilon \in L$. Then $e(L) = k(L)$ and $ce(L) = ck(L)$. Since $\epsilon \notin ck(L)$, we obtain $ece(L) = eck(L) = kck(L) - \{\epsilon\}$. Then, $cece(L) = ckck(L) \cup \{\epsilon\}$. So $ecece(L) = kckck(L)$. From Lemma 12, we deduce $kckck(L) = ecece(L) = cece(L) = ckck(L) \cup \{\epsilon\}$.

Second, suppose $\epsilon \notin L$. Then $e(L) = k(L) - \{\epsilon\}$ and $ce(L) = ck(L) \cup \{\epsilon\}$. We obtain $ece(L) = kck(L)$ and $cece(L) = ckck(L)$. So $ecece(L) = eckck(L) = kckck(L) - \{\epsilon\}$. From Lemma 12, we deduce $kckck(L) = ecece(L) \cup \{\epsilon\} = cece(L) \cup \{\epsilon\} = ckck(L) \cup \{\epsilon\}$.

Lemma 14. *Let L be any language.*

(a) If $xy \in kp(L)$ then $x \in kp(L)$ and $y \in kf(L)$.
(b) If $xy \in ks(L)$ then $x \in kf(L)$ and $y \in ks(L)$.
(c) If $xy \in kf(L)$ then $x, y \in kf(L)$.
(d) If $xy \in kw(L)$, then $x, y \in kw(L)$.

Proof. We prove only (b), with the others being proved similarly. If $xy \in ks(L)$, then $x \in pks(L)$ and $y \in sks(L)$. But $s \subseteq f$, so $pks \subseteq pkf$, and $pkf = kf$ by (22). Hence $x \in kf(L)$. Similarly, $sks \equiv ks$ by Lemma 3, so $y \in ks(L)$. □

Lemma 15. *We have pcpckp ⊆ kp.*

Proof. Let $x \in pcpckp(L)$. Then there exists y such that $xy \in cpckp(L)$. So $xy \notin pckp(L)$. Then, for all z, we have $xyz \notin ckp(L)$. Hence $xyz \in kp(L)$. Thus $x \in pkp(L) = kp(L)$.

Theorem 16. *Let $b \in \{p, s, f, w\}$. Then*

26. $kcb(L) = cb(L) \cup \{\epsilon\}$
27. $kckb(L) = ckb(L) \cup \{\epsilon\}$
28. $kbcbckb(L) = bcbckb(L) \cup \{\epsilon\}$.

Proof. We prove only three of these identities; the others can be proved similarly.

$kcp(L) = cp(L) \cup \{\epsilon\}$: Assume $x \in kcp(L)$. Either $x = \epsilon$ or we can write $x = x_1 x_2 \cdots x_n$ for some $n \geq 1$, where each $x_i \in cp(L)$. Then each $x_i \notin p(L)$.

In particular $x_1 \notin p(L)$. Then $x_1x_2\cdots x_n \notin p(L)$, because if it were, then $x_1 \in p(L)$, a contradiction. Hence $x \in cp(L)$.

$kckp(L) = ckp(L) \cup \{\epsilon\}$: Assume $x \in kckp(L)$. Either $x = \epsilon$ or we can write $x = x_1x_2\cdots x_n$ for some $n \geq 1$, where each $x_i \in ckp(L)$. Then each $x_i \notin kp(L)$. In particular $x_1 \notin kp(L)$. Hence $x_1(x_2\cdots x_n) \notin kp(L)$, because if it were, then $x_1 \in kp(L)$ by Lemma 14, a contradiction. Hence $x \notin kp(L)$, so $x \in ckp(L)$, as desired.

$kpcpckp(L) = pcpckp(L) \cup \{\epsilon\}$: Assume $x \in kpcpckp(L)$. Either $x = \epsilon$ or we can write $x = x_1\cdots x_n$, where each $x_i \in pcpckp(L)$. In particular, there exists y such that $x_ny \in cpckp(L)$; that is, $x_ny \notin pckp(L)$. Assume $x \notin pcpckp(L)$. Then $xy \notin cpckp(L)$, so $xy \in pckp(L)$. Then there exists z such that $xyz \in ckp(L)$; that is, $xyz \notin kp(L)$. But from Lemma 15, we know that every x_i is in $kp(L)$. Further, since $x_ny \notin pckp(L)$, we have $x_nyz \notin ckp(L)$; that is, $x_nyz \in kp(L)$. This shows that $xyz = x_1\cdots x_{n-1}(x_nyz)$ belongs to $kp(L)$, a contradiction. □

Theorem 17. *We have*

29. $sckp(L) = \Sigma^*$ *or* $\emptyset$.
30. *The same result holds for*

$$fckp, pcks, fcks, pckf, sckf, fckf, wckp, wcks, wckf, wckw, pckw, sckw, fckw.$$

Proof. To prove (29), note that either $kp(L) = \Sigma^*$, or $kp(L)$ omits some word v. In the former case, $ckp(L) = \emptyset$, and so $sckp(L) = \emptyset$. In the latter case, we have $kp(L)$ omits v, so $kp(L)$ must also omit $v\Sigma^*$ (for otherwise, if $vx \in kp(L)$ for some x, then $v \in kp(L)$ by Lemma 14, a contradiction). Then $v\Sigma^* \in ckp(L)$ and hence $sckp(L) = \Sigma^*$.

The other results can be proved similarly. □

Lemma 18. *Let L be any language.*

(a) If $xy \in skp(L)$, then $x, y \in skp(L)$.
(b) If $xy \in pks(L)$, then $x, y \in pks(L)$.

Proof. We prove only (a), with (b) being proved similarly.

If $xy \in skp(L)$, then $x \in pskp(L)$ and $y \in sskp(L)$. But $pskp \equiv (ps)kp \equiv fkp \equiv skp$ by (21). So $x \in skp(L)$. Also, $sskp = skp$, so $y \in skp(L)$. □

Theorem 19. *We have*

31. $scskp(L) = \Sigma^*$ *or* $\emptyset$.
32. *The same result holds for $pcpks$.*

Proof. We prove only the first result; the second can be proved analogously. Either $skp(L) = \Sigma^*$, or it omits some word v. In the first case we have $cskp(L) = \emptyset$ and hence $scskp(L) = \emptyset$. In the second case, $skp(L)$ must omit $v\Sigma^*$ (for if $vx \in skp(L)$ for any x, then by Lemma 18 we have $v \in skp(L)$, a contradiction). Hence $scskp(L) = \Sigma^*$. □

5 Results

Our main result is the following:

Theorem 20. *Let $S = \{k, e, c, p, f, s, w, r\}$. Then for every language L, the set $\mathcal{O}_S(L)$ contains at most 5676 distinct languages.*

Proof. Our proof was carried out mechanically. We used breadth-first search to examine the set $S^* = \{k, e, c, p, f, s, w, r\}^*$ by increasing length of the words; within each length we used lexicographic order with $k < e < c < p < f < s < w < r$. The nodes remaining to be examined are stored in a queue Q.

As each new word x representing a series of language operations is examined, we test it to see if any factor is of the form given in identities (23)–(24) or (30)–(32). If it is, then the corresponding language must be either Σ^*, $\emptyset$, $\{\epsilon\}$, or Σ^+; furthermore, each descendant language will be of this form. In this case the word x is discarded.

Otherwise, we use the remaining identities above to try to reduce x to an equivalent word that we have previously encountered. If we succeed, then x is discarded. Otherwise $x(L)$ is potentially a new language, so we append all the words Sx to the end of the queue. Some simplifications are possible. For example, using our identities we can assume x contains only a single r and this appears at the end; this cuts down on the search space.

We treat the identities (25)–(27) somewhat differently. We keep track of whether a language contains ϵ or not. For example, when appropriate, we can replace $akcb$ with acb for $a, b \in \{p, s, f, w\}$.

If the process terminates, then $\mathcal{O}_S(L)$ is of finite cardinality.

We wrote our program in APL. For $S = \{k, c, p, f, s, w, r\}$, the process terminated with 5672 nodes that could not be simplified using our identities. We did not count $\emptyset, \{\epsilon\}, \Sigma^+$, and Σ^*. The total is thus 5676.

The longest word examined was $ckcpcpckpckpckpcpcpckckcr$, of length 25, and the same word with p replaced by s.

Our program generates a complete description of the words and how they simplify, which can be viewed at `www.cs.uwaterloo.ca/~shallit/papers.html`.

□

Remark 21. If we use *two* arbitrary closure operations a and b with no relation between them, then the monoid generated by $\{a, b\}$ could potentially be infinite, since any two finite prefixes of $ababab\cdots$ are distinct.

Here is an example. Let p denote prefix, as above, and define the exponentiation operation

$$t(L) = \{x^i \ : \ x \in L \text{ and } i \text{ is an integer} \geq 1\}. \tag{2}$$

Then it is easy to see that t is a closure operation, and hence the orbits $\mathcal{O}_{\{p\}}(L)$ and $\mathcal{O}_{\{t\}}(L)$ are finite, for all L. However, for $L = \{ab\}$, the orbit $\mathcal{O}_{\{p,t\}}(L)$ is infinite, as $aba^i \in (pt)^i(L)$, but $aba^i \notin (pt)^j(L)$ for all $j < i$.

Thus our proof of Theorem 20 crucially depends on the properties of the operations $\{k, e, c, p, s, f, w, r\}$.

We now give some results for some interesting subsets of S.

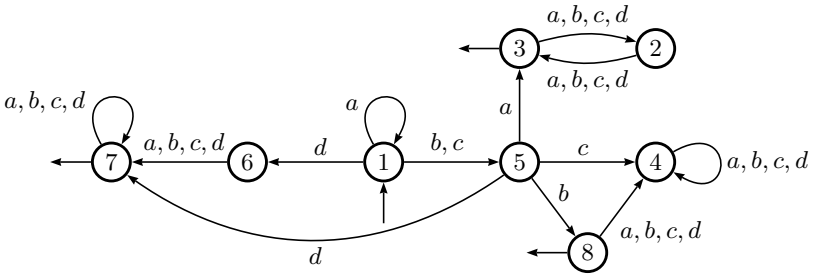


Fig. 1. DFA accepting a language L with orbit size 14 under operations p and c

Table 1. Final states for composed operations

language	final states	language	final states
L	3,7,8	$pcpc(L)$	1,5,6,7
$c(L)$	1,2,4,5,6	$cpcp(L)$	2,3,6,7
$p(L)$	1,2,3,5,6,7,8	$cpcpc(L)$	2,3,4,8
$pc(L)$	1,2,3,4,5,6,8	$pcpcp(L)$	1,2,3,5,6,7
$cp(L)$	4	$pcpcpc(L)$	1,2,3,4,5,8
$cpc(L)$	7	$cpcpcp(L)$	4, 8
$pcp(L)$	1,4,5,8	$cpcpcpc(L)$	6, 7

5.1 Prefix and Complement

In this case at most 14 distinct languages can be generated. The bound of 14 can be achieved, e.g., by the regular language over $\Sigma = \{a, b, c, d\}$ given by the regular expression $a^*((b + c)(a(\Sigma\Sigma)^* + b + d\Sigma^*) + d\Sigma^+)$ and accepted by the DFA in Figure 1.

Table 1 gives the appropriate set of final states under the operations.

5.2 Prefix, Kleene Star, Complement

The same process, described above for the operations $\{k, e, c, p, s, f, w, r\}$, can be carried out for other subsets, such as $\{k, c, p\}$. For this our breadth-first search gives 1066 languages. The longest word examined was $ckcpcpckpckpckpcpcpckckc$.

5.3 Factor, Kleene Star, Complement

Similarly, we can examine $\{k, c, f\}$. Here breadth-first search gives 78 languages, so our bound is $78 + 4 = 82$. We can improve this bound by considering new kinds of arguments.

Lemma 22. *Let L be any language. There are at most 4 languages distinct from $\Sigma^*, \emptyset, \Sigma^+$, and $\{\epsilon\}$ in $\mathcal{O}_{\{k,f,kc,fc\}}(f(L))$. These languages are among $f(L)$, $kf(L)$, $kckf(L)$, and $kcf(L)$.*

Proof (Sketch). First observe that the set of languages $\{\Sigma^*, \emptyset, \Sigma^+, \{\epsilon\}\}$ is closed under any operation of the set $\{k, c, f\}$. We make a case study. We consider

successively the languages generated by $\{k, f, kc, fc\}$ from $fcf(L), kf(L)$, and $kcf(L)$. We make use of Identities (22), (24), (26), (27), and (30).

Let $\mathrm{alph}(L)$ denote the minimal alphabet of a language L, that is, the minimal set of letters that occur in words of L.

Lemma 23. *Let L be any language. We have $kf(L) = k(\mathrm{alph}(L))$.*

Proof. The minimal alphabets of L and $f(L)$ coincide. Thus $f(L) \subseteq k(\mathrm{alph}(L))$, so $kf(L) \subseteq k(\mathrm{alph}(L))$. Further, $\mathrm{alph}(L) \subseteq f(L)$. So $k(\mathrm{alph}(L)) \subseteq kf(L)$ as well.

Lemma 24. *Let L be any language. There are at most 2 languages distinct from $\Sigma^*, \emptyset, \Sigma^+$, and $\{\epsilon\}$ in $\mathcal{O}_{\{k,f,kc,fc\}}(fk(L)) - \mathcal{O}_{\{k,f,kc,fc\}}(f(L))$. These languages are among $fk(L)$ and $kcfk(L)$.*

Proof. Apply Lemma 22 to $k(L)$ and use $kfk \equiv kf$. To see the latter identity, use Lemma 23 and observe that $\mathrm{alph}(k(L)) = \mathrm{alph}(L)$.

Lemma 25. *For any language L, we have either $f(L) = \Sigma^*$ or $fc(L) = \Sigma^*$.*

Proof. Assume $f(L) \neq \Sigma^*$. Then there exists a word in $cf(L)$, say w. Hence $\Sigma^* w \Sigma^* \cap f(L) = \emptyset$. Since $L \subseteq f(L)$, we also have $\Sigma^* w \Sigma^* \cap L = \emptyset$, that is $\Sigma^* w \Sigma^* \subseteq c(L)$. This implies $fc(L) = \Sigma^*$.

Theorem 26. *50 is a tight upper bound for the size of the orbit of $\{k, c, f\}$.*

Proof (Sketch). From Lemmas 22 and 24, and Identity (25), starting with an arbitrary language L, the languages in $\mathcal{O}_{\{k,c,f\}}(L)$ that may differ from $\Sigma^*, \emptyset, \Sigma^+$, and $\{\epsilon\}$ are among the images of L and $c(L)$ under the 16 operations

$$f, kf, kckf, kcf, fk, kcfk, fck, kfck, kckfck, kcfck, \\ fkck, kcfkck, fckck, kfckck, kckfckck, kcfckck. \tag{3}$$

the complements of these images, together with the 14 languages in $\mathcal{O}_{\{k,c\}}(L)$.

By using Lemma 25, we show that there are at most 32 pairwise distinct languages among the $64 = 16 \cdot 4$ languages given by the images of L and $c(L)$ under the 16 operations (3) and their complements.

Adding the 14 languages in $\mathcal{O}_{\{k,c\}}(L)$, and $\Sigma^*, \emptyset, \Sigma^+$, and $\{\epsilon\}$, we obtain that $50 = 32 + 14 + 4$ is an upper bound for the size of the orbit of $\{k, c, f\}$.

The bound is tight because the language L given by two copies (over disjoint alphabets) of the language accepted by the DFA of Figure 2 over the alphabet $\{a, b, c, d, e, f, g, h, i\}$ (i is a letter that does not occur in any word of L, i.e., $i \notin \mathrm{alph}(L)$) has 50 pairwise distinct elements in $\mathcal{O}_{\{k,c,f\}}(L)$.

5.4 Kleene Star, Prefix, Suffix, Factor

Here there are at most 13 distinct languages, given by the action of

$$\{\epsilon, k, p, s, f, kp, ks, kf, pk, sk, fk, pks, skp\}.$$

The bound of 13 is achieved, for example, by $L = \{abc\}$.

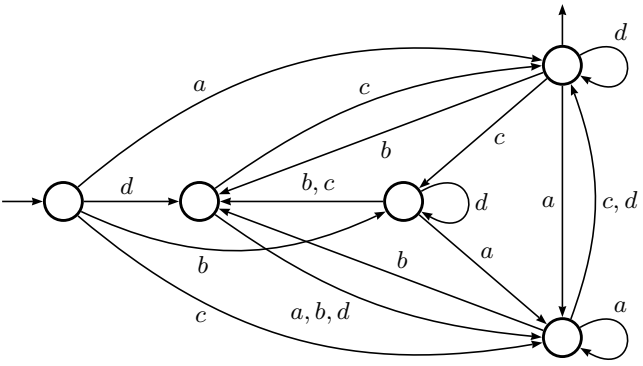


Fig. 2. The DFA made of two copies of this DFA accept a language L with orbit size 50 under operations k, c, and f

Table 2. Upper bounds on the size of the orbit

r	**2**	w	**2**	f	**2**
s	**2**	p	**2**	c	**2**
k	**2**	w, r	**4**	f, r	**4**
f, w	**3**	s, w	**3**	s, f	**3**
p, w	**3**	p, f	**3**	c, r	**4**
c, w	**6**∗	c, f	**6**∗	c, s	**14**
c, p	**14**	k, r	**4**	k, w	**4**
k, f	**5**	k, s	**5**	k, p	**5**
k, c	**14**	f, w, r	**6**	s, f, w	**4**
p, f, w	**4**	p, s, f	**4**	c, w, r	**10**∗
c, f, r	**10**∗	c, f, w	**8**∗	c, s, w	**16**∗
c, s, f	**16**∗	c, p, w	**16**∗	c, p, f	**16**∗
k, w, r	**7**	k, f, r	**9**	k, f, w	**6**
k, s, w	**7**	k, s, f	**9**	k, p, w	**7**
k, p, f	**9**	k, c, r	**28**	k, c, w	**38**∗
k, c, f	**50**∗	k, c, s	1070	k, c, p	1070
p, s, f, r	**8**	p, s, f, w	**5**	c, f, w, r	**12**∗
c, s, f, w	**16**∗	c, p, f, w	**16**∗	c, p, s, f	**16**∗
k, f, w, r	**11**	k, s, f, w	**10**	k, p, f, w	**10**
k, p, s, f	**13**	k, c, w, r	72∗	k, c, f, r	**84**∗
k, c, f, w	66∗	k, c, s, w	1114	k, c, s, f	1450
k, c, p, w	1114	k, c, p, f	1450	p, s, f, w, r	**10**
c, p, s, f, r	**30**∗	c, p, s, f, w	**16**∗	k, p, s, f, r	**25**
k, p, s, f, w	**14**	k, c, f, w, r	120∗	k, c, s, f, w	1474
k, c, p, f, w	1474	k, c, p, s, f	2818	c, p, s, f, w, r	**30**∗
k, p, s, f, w, r	**27**	k, c, p, s, f, r	5628	k, c, p, s, f, w	2842
k, c, p, s, f, w, r	5676				

5.5 Summary of Results

Table 2 gives our upper bounds on the number of distinct languages generated by the set of operations. An entry in **bold** indicates that the bound is known to be tight. Some entries, such as p, r, are omitted, since they are the same as others (in this case, p, s, f, r).

Most bounds were obtained directly from our program, and others by additional reasoning. An asterisk denotes those bounds for which some additional reasoning was required to reduce the upper bound found by our program to the bound shown in Table 2.

6 Further Work

We plan to continue to refine our estimates in Table 2, and pursue the status of other sets of operations. For example, if t is the exponentiation operation defined in (2), then, using the identities $kt = tk = k$, and the inclusion $t \subseteq k$, we get the additional Kuratowski-style identities $kctckck \equiv kck$, $kckctck \equiv kck$, $kctctck \equiv kck$, $tctctck \equiv tck$, and $kctctct \equiv kct$. This allows us to prove that $\mathcal{O}_{\{k,c,t\}}(L)$ is finite and of cardinality at most 126.

Acknowledgments

We thank John Brzozowski for his comments.

References

1. Brzozowski, J., Grant, E., Shallit, J.: Closures in formal languages and Kuratowski's theorem. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 125–144. Springer, Heidelberg (2009)
2. Gardner, B.J., Jackson, M.: The Kuratowski closure-complement theorem. New Zealand J. Math. 38, 9–44 (2008)
3. Kuratowski, C.: Sur l'opération $\bar{A}$ de l'analysis situs. Fund. Math. 3, 182–199 (1922)
4. Peleg, D.: A generalized closure and complement phenomenon. Discrete Math. 50(2-3), 285–293 (1984)

Finitary Languages*,**

Krishnendu Chatterjee[1] and Nathanaël Fijalkow[1,2]

[1] IST (Institute of Science and Technology), Austria
krishnendu.chatterjee@ist.ac.at
[2] ÉNS Cachan (École Normale Supérieure de Cachan), France
nathanael.fijalkow@gmail.com

Abstract. The class of ω-regular languages provides a robust specification language in verification. Every ω-regular condition can be decomposed into a safety part and a liveness part. The liveness part ensures that something good happens "eventually". Finitary liveness was proposed by Alur and Henzinger as a stronger formulation of liveness [2]. It requires that there exists an unknown, fixed bound b such that something good happens within b transitions. In this work we consider automata with finitary acceptance conditions defined by finitary Büchi, parity and Streett languages. We give their topological complexity of acceptance conditions, and present a regular-expression characterization of the languages they express. We provide a classification of finitary and classical automata with respect to the expressive power, and give optimal algorithms for classical decisions questions on finitary automata. We (a) show that the finitary languages are Σ_2^0-complete; (b) present a complete picture of the expressive power of various classes of automata with finitary and infinitary acceptance conditions; (c) show that the languages defined by finitary parity automata exactly characterize the star-free fragment of ωB-regular languages [4]; and (d) show that emptiness is NLOGSPACE-complete and universality as well as language inclusion are PSPACE-complete for finitary automata.

1 Introduction

Classical ω-regular languages: strengths and weakness. The widely studied class of ω-regular languages provides a robust language for specification for solving control and verification problems (see, *e.g*, [13,14]). Every ω-regular specification can be decomposed into a safety part and a liveness part [1]. The safety part ensures that the component will not do anything "bad" (such as violate an invariant) within any finite number of transitions. The liveness part ensures that the component will do something "good" (such as proceed, or respond, or terminate) in the long-run. Liveness can be violated only in the limit, by infinite sequences of transitions, as no bound is stipulated on when the "good" thing must happen. This infinitary, classical formulation of liveness has both strengths and

* The research was supported by Austrian NFN ARiSE.
** Fuller version with proofs available at [7].

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 216–226, 2011.

weaknesses. A main strength is robustness, in particular, independence from the chosen granularity of transitions. Another main strength is simplicity, allowing liveness to serve as an abstraction for complicated safety conditions. For example, a component may always respond in a number of transitions that depends, in some complicated manner, on the exact size of the stimulus. Yet for correctness, we may be interested only that the component will respond "eventually". However, these strengths also point to a weakness of the classical definition of liveness: it can be satisfied by components that in practice are quite unsatisfactory because no bound can be put on their response time.

Stronger notion of liveness. For the weakness of the infinitary formulation of liveness, alternative and stronger formulations of liveness have been proposed. One of these is *finitary* liveness [2]: finitary liveness does not insist on a response within a known bound b (*i.e*, every stimulus is followed by a response within b transitions), but on response within some unknown bound (*i.e*, there exists b such that every stimulus is followed by a response within b transitions). Note that in the finitary case, the bound b may be arbitrarily large, but the response time must not grow forever from one stimulus to the next. In this way, finitary liveness still maintains the robustness (independence of step granularity) and simplicity (abstraction of complicated safety) of traditional liveness, while removing unsatisfactory implementations.

Finitary parity and Streett conditions. The classical infinitary notion of fairness is given by the Streett condition: it consists of a set of d pairs of requests and corresponding responses (grants) and requires that every request that appears infinitely often must be responded infinitely often. Its finitary counterpart, the finitary Streett condition requires that there is a bound b such that in the limit every request is responded within b steps. The classical infinitary parity condition consists of a priority function and requires that the minimum priority visited infinitely often is even. Its finitary counterpart, the finitary parity condition requires that there is a bound b such that in the limit after every odd priority a lower even priority is visited within b steps.

Results on classical automata. There are several robust results on the languages expressible by automata with infinitary Büchi, parity and Streett conditions, as follows: (a) *Topological complexity:* it is known that Büchi languages are Π_2^0-complete, whereas parity and Streett languages lie in the boolean closure of Σ_2^0 and Π_2^0 [12]; (b) *Automata expressive power:* non-deterministic automata with Büchi conditions have the same expressive power as deterministic and non-deterministic parity and Streett automata [9,15]; and (c) *Regular expression characterization:* the class of languages expressed by deterministic parity is exactly defined by ω-regular expressions (see the handbook [16] for details).

Our results. For finitary languages, topological, automata-theoretic, regular-expression and decision problems studies were all missing. In this work we present results in the four directions, as follows:

1. *Topological complexity.* We show that finitary Büchi, parity and Streett conditions are Σ_2^0-complete.

2. *Automata expressive power.* We show that finitary automata are incomparable in expressive power with classical automata. As in the infinitary setting, we show that non-deterministic automata with finitary Büchi, parity and Streett conditions have the same expressive power, as well as deterministic parity and Streett automata, which are strictly more expressive than deterministic finitary Büchi automata. However, in contrast to the infinitary case, for finitary parity condition, non-deterministic automata are strictly more expressive than the deterministic counterpart. As a by-product we derive boolean closure properties for finitary automata.
3. *Regular expression characterization.* We consider the characterization of finitary automata through an extension of ω-regular languages defined as ωB-regular languages by [4]. We show that non-deterministic finitary Büchi automata express exactly the star-free fragment of ωB-regular languages.
4. *Decision problems.* We show that emptiness is NLOGSPACE-complete and universality as well as language inclusion are PSPACE-complete for finitary automata.

Related works. The notion of finitary liveness was introduced in [2], and games with finitary objectives was studied in [8]. A generalization of ω-regular languages as ωB-regular languages was introduced in [4] and variants were studied in [5] (also see [3] for a survey); a topological characterization has been given in [11]. Our work along with topological and automata-theoretic studies of finitary languages, explores the relation between finitary languages and ωB-regular expressions, rather than identifying a subclass of ωB-regular expressions. We identify the exact subclass of ωB-regular expressions that corresponds to non-deterministic finitary parity automata.

2 Definitions

2.1 Topological Complexity of Languages

Let Σ be a finite set, called the alphabet. A word w is a sequence of letters, which can be either finite or infinite, it will be described as a sequence $w_0 w_1 \dots$ of letters. A language is a set of words: $L \subseteq \Sigma^*$ is a language over finite words and $L \subseteq \Sigma^\omega$ over infinite words.

Cantor topology and Borel hierarchy. Cantor topology on Σ^ω is given by *open* sets: a language is open if it can be described as $W \cdot \Sigma^\omega$ where $W \subseteq \Sigma^*$. Let Σ^0_1 denote the open sets and Π^0_1 denote the closed sets (a language is closed if its complement is open): they form the first level of the Borel hierarchy. Inductively, we define: Σ^0_{i+1} is obtained as countable union of Π^0_i sets; and Π^0_{i+1} is obtained as countable intersection of Σ^0_i sets. The higher a language is in the Borel hierarchy, the higher its topological complexity.

Since the above classes are closed under continuous preimage, we can define the notion of Wadge reduction [17]: L reduces to L', denoted by $L \preceq L'$, if there exists a continuous function $f : \Sigma^\omega \to \Sigma^\omega$ such $L = f^-(L')$, where $f^-(L')$ is the

preimage of L' by f. A language is hard with respect to a class if all languages of this class reduce to it. If it additionally belongs to this class, then it is complete.

Classical liveness conditions. We now consider three classes of languages that are widespread in verification and specification. They define liveness properties, *i.e*, intuitively say that something good will happen "eventually". For an infinite word w, let $\mathrm{Inf}(w) \subseteq \Sigma$ denote the set of letters that appear infinitely often in w. The class of Büchi languages is defined as follows, given $F \subseteq \Sigma$:

$$\text{Büchi}(F) = \{w \mid \mathrm{Inf}(w) \cap F \neq \emptyset\}$$

i.e, the Büchi condition requires that some letter in F appears infinitely often. The class of parity languages is defined as follows, given $p : \Sigma \to \mathbb{N}$ a priority function that maps letters to integers (representing priorities):

$$\mathrm{Parity}(p) = \{w \mid \min(p(\mathrm{Inf}(w))) \text{ is even}\}$$

i.e, the parity condition requires that the lowest priority which appears infinitely often is even. The class of Streett languages is defined as follows, given $(R, G) = (R_i, G_i)_{1 \leq i \leq d}$, where $R_i, G_i \subseteq \Sigma$ are request-grant pairs:

$$\mathrm{Streett}(R, G) = \{w \mid \forall i, 1 \leq i \leq d, \mathrm{Inf}(w) \cap R_i \neq \emptyset \Rightarrow \mathrm{Inf}(w) \cap G_i \neq \emptyset\}$$

i.e, the Streett condition requires that for all requests R_i, if it appears infinitely often, then the corresponding grant G_i also appears infinitely often.

The following theorem presents the topological complexity of the classical languages:

Theorem 1 (Topological complexity of classical languages [12])
- *For all $\emptyset \subset F \subset \Sigma$, the language* Büchi$(F)$ *is Π_2^0-complete.*
- *The parity and Streett languages lie in the boolean closure of Σ_2^0 and Π_2^0.*

2.2 Finitary Languages

The finitary parity and Streett languages have been defined in [8]. We recall their definitions, and also specialize them to finitary Büchi languages. Let $(R, G) = (R_i, G_i)_{1 \leq i \leq d}$, where $R_i, G_i \subseteq \Sigma$, the definition for $\mathrm{FinStreett}(R, G)$ uses distance sequence as follows:

$$\mathrm{dist}_k^j(w, (R, G)) = \begin{cases} 0 & w_k \notin R_j \\ \inf\{k' - k \mid k' \geq k, w_{k'} \in G_j\} & w_k \in R_j \end{cases}$$

i.e, given a position k where R_j is requested, $\mathrm{dist}_k^j(w, (R, G))$ is the time steps (number of transitions) between the request R_j and the corresponding grant G_j. Note that $\inf(\emptyset) = \infty$. Then $\mathrm{dist}_k(w, (R, G)) = \max\{\mathrm{dist}_k^j(w, p) \mid 1 \leq j \leq d\}$ and:

$$\mathrm{FinStreett}(R, G) = \{w \mid \limsup_k \mathrm{dist}_k(w, (R, G)) < \infty\}$$

i.e, the finitary Streett condition requires the supremum limit of the distance sequence to be bounded.

Since parity languages can be considered as a particular case of Streett languages, where $G_1 \subseteq R_1 \subseteq G_2 \subseteq R_2 \ldots$, the latter allows to define FinParity(p). The same applies to finitary Büchi languages, which is a particular case of finitary parity languages where the letters from the set F have priority 0 and others have priority 1. We get the following definitions. Let $p : \Sigma \to \mathbb{N}$ a priority function, we define:

$$\mathrm{dist}_k(w,p) = \inf\{k' - k \mid k' \geq k, p(w_{k'}) \text{ is even and } p(w_{k'}) \leq p(w_k)\}$$

i.e, given a position k where $p(w_k)$ is odd, $\mathrm{dist}_k(w,p)$ is the time steps between the odd priority $p(w_k)$ and a lower even priority. Then $\mathrm{FinParity}(p) = \{w \mid \limsup_k \mathrm{dist}_k(w,p) < \infty\}$. We define similarly the finitary Büchi language: given $F \subseteq \Sigma$, let:

$$\mathrm{next}_k(w,F) = \inf\{k' - k \mid k' \geq k, w_{k'} \in F\}$$

i.e, $\mathrm{next}_k(w,F)$ is the time steps before visiting a letter in F. Then

$$\mathrm{FinBüchi}(F) = \{w \mid \limsup_k \mathrm{next}_k(w,F) < \infty\}.$$

2.3 Automata, ω-Regular and Finitary Languages

Definition 1. *An automaton is a tuple $\mathcal{A} = (Q, \Sigma, Q_0, \delta, Acc)$, where Q is a finite set of states, Σ is the finite input alphabet, $Q_0 \subseteq Q$ is the set of initial states, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation and $Acc \subseteq Q^\omega$ is the acceptance condition.*

An automaton is deterministic if it has a single initial state and for every state and letter there is at most one transition. The transition relation of deterministic automata are described by functions $\delta : Q \times \Sigma \to Q$. An automaton is complete if for every state and letter there is a transition.

Acceptance conditions. We will consider various acceptance conditions for automata obtained from the last section by considering Q as the alphabet. Automata with finitary acceptance conditions are referred as finitary automata; classical automata are those equipped with infinitary acceptance conditions.

Notation 1. *We use a standard notation to denote the set of languages recognized by some class of automata. The first letter is either N or D, where N stands for "non-deterministic" and D stands for "deterministic". The last letter refers to the acceptance condition: B stands for "Büchi", P stands for "parity" and S stands for "Streett". The acceptance condition may be prefixed by F for "finitary". For example, NP denotes non-deterministic parity automata, and DFS denotes deterministic finitary Streett automata. We have the following combination:*

$$\left\{\begin{matrix} N \\ D \end{matrix}\right\} \cdot \left\{\begin{matrix} F \\ \varepsilon \end{matrix}\right\} \cdot \left\{\begin{matrix} B \\ P \\ S \end{matrix}\right\}$$

We denote by $\mathbb{L}_\omega$ the set of ω-regular languages ([6,15,9,10]):

$$\mathbb{L}_\omega = NB = DP = NP = DS = NS.$$

3 Topological Complexity

In this section we define a finitary operator $\mathsf{UniCloOmg}$ that allows us to relate finitary languages to their infinitary counterparts; we then give their topological complexity.

Union-closed-omega-regular operator on languages. Given a language $L \subseteq \Sigma^\omega$, the language $\mathsf{UniCloOmg}(L) \subseteq \Sigma^\omega$ is the *union* of the languages M that are subsets of L, *ω-regular* and *closed*, *i.e*, $\mathsf{UniCloOmg}(L) = \bigcup\{M \mid M \subseteq L, M \in \Pi_1, M \in \mathbb{L}_\omega\}$.

Proposition 1. *For all languages $L \subseteq \Sigma^\omega$ we have $\mathsf{UniCloOmg}(L) \in \Sigma_2^0$.*

The following lemma shows that $\mathrm{FinStreett}(R, G)$ is obtained by applying the $\mathsf{UniCloOmg}$ operator to $\mathrm{Streett}(R, G)$.

Lemma 1. *For all $(R, G) = (R_i, G_i)_{1 \le i \le d}$, where $R_i, G_i \subseteq \Sigma$, we have*

$$\mathsf{UniCloOmg}(\mathrm{Streett}(R, G)) = \mathrm{FinStreett}(R, G).$$

Corollary 1. *The following assertions hold:*
- *For all $p : \Sigma \to \mathbb{N}$, we have $\mathsf{UniCloOmg}(\mathrm{Parity}(p)) = \mathrm{FinParity}(p)$;*
- *For all $F \subseteq \Sigma$, we have $\mathsf{UniCloOmg}(\mathrm{Büchi}(F)) = \mathrm{FinBüchi}(F)$.*

Theorem 2 (Topological characterization of finitary languages). *The finitary Büchi, finitary parity and finitary Streett languages are Σ_2^0-complete.*

Proof. We show that if $\emptyset \subset F \subset \Sigma$, then $\mathrm{FinBüchi}(F)$ is Σ_2^0-complete. It follows from Corollary 1 that $\mathrm{FinBüchi}(F) \in \Sigma_2^0$. We now show that $\mathrm{FinBüchi}(F)$ is Σ_2^0-hard. By Theorem 1 we have that $\mathrm{Büchi}(\Sigma \setminus F)$ is Π_2^0-complete, hence $\Sigma^\omega \setminus \mathrm{Büchi}(\Sigma \setminus F)$ is Σ_2^0-complete. We present a topological reduction to show that $\Sigma^\omega \setminus \mathrm{Büchi}(\Sigma \setminus F) \preceq \mathrm{FinBüchi}(F)$. Let $b : \Sigma^\omega \to \Sigma^\omega$ be the stuttering function defined as follows:

$$\begin{array}{rcl} w & = & w_0 \quad w_1 \quad \dots \quad\quad w_n \quad\quad \dots \\ b(w) & = & w_0 \underbrace{w_1 w_1}_{2} \dots \underbrace{w_n w_n \dots w_n}_{2^n} \dots \end{array}$$

The function b is continuous. We can easily check that the following holds:

$$\mathrm{Inf}(w) \subseteq F \;\text{ iff }\; \exists B \in \mathbb{N}, \exists n \in \mathbb{N}, \forall k \ge n, \mathrm{next}_k(b(w), F) \le B.$$

Hence we get $\Sigma^\omega \setminus \mathrm{Büchi}(\Sigma \setminus F) \preceq \mathrm{FinBüchi}(F)$, so $\mathrm{FinBüchi}(F)$ is Σ_2^0-complete. From this we can deduce the two other claims. □

4 Expressive Power of Finitary Automata

In this section we consider the finitary automata, and compare their expressive power to classical automata. We then address the question of determinization. Deterministic finitary automata enjoy nice properties that allow to describe languages they recognize using the UniCloOmg operator. As a by-product we get boolean closure properties of finitary automata.

4.1 Comparison with Classical Automata

Finitary conditions allow to express bounds requirements:

Example 1 (DFB $\not\subseteq \mathbb{L}_\omega$). Consider the finitary Büchi automaton shown in Fig. 1, the state labeled 0 being its only final state. Its language is $L_B = \{(b^{j_0}a^{f(0)}) \cdot (b^{j_1}a^{f(1)}) \cdot (b^{j_2}a^{f(2)}) \dots \mid f : \mathbb{N} \to \mathbb{N}, f \text{ bounded}, \forall i \in \mathbb{N}, j_i \in \mathbb{N}\}$. Indeed, 0-labeled state is visited while reading the letter b, and the 1-labeled state is visited while reading the letter a. An infinite word is accepted iff the 0-labeled state is visited infinitely often and there is a bound between two consecutive visits of the 0-labeled state. We can easily see that L_B is not ω-regular, using proof ideas from [4]: its complement would be ω-regular, so it would contain ultimately periodic words, which is not the case.

However, finitary automata cannot distinguish between "many b's" and "only b's":

Example 2 (DB $\not\subseteq$ NFB). Consider the language of infinitely many a's, *i.e*, $L_I = \{w \mid w \text{ has an infinite number of } a\}$. The language L_I is recognized by a simple deterministic Büchi automaton. However, we can show that there is no finitary Büchi automata that recognizes L_I. Intuitively, such an automaton would, while reading the infinite word $w = ab\ ab^2\ ab^3\ ab^4 \dots ab^n \dots \in L_I$, have to distinguish between all b's, otherwise it would accept a word with only b's at the end.

4.2 Deterministic Finitary Automata

Given a deterministic complete automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, Acc)$, we define its finitary restriction by $\mathsf{UniCloOmg}(\mathcal{A}) = (Q, \Sigma, q_0, \delta, \mathsf{UniCloOmg}(Acc))$.

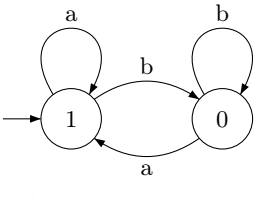


Fig. 1. A finitary Büchi automaton $\mathcal{A}$

Treating the automaton as a transducer, we consider the following function: $C_{\mathcal{A}} : \Sigma^\omega \to Q^\omega$ which maps an infinite word w to the unique run ρ of $\mathcal{A}$ on w (there is a unique run since $\mathcal{A}$ is deterministic and complete). Then:

$$\mathcal{L}(\mathcal{A}) = \{w \mid C_{\mathcal{A}}(w) \in Acc\} = C^-_{\mathcal{A}}(Acc).$$

The property $C^-_{\mathcal{A}}(\mathsf{UniCloOmg}(Acc)) = \mathsf{UniCloOmg}(C^-_{\mathcal{A}}(Acc))$ follows from the following lemma.

Lemma 2. *For all $\mathcal{A} = (Q, \Sigma, q_0, \delta, Acc)$ deterministic complete automaton, we have:*

1. *for all $A \subseteq Q^\omega$, A is closed $\Rightarrow C^-_{\mathcal{A}}(A)$ closed ($C_{\mathcal{A}}$ is continuous).*
2. *for all $L \subseteq \Sigma^\omega$, L is closed $\Rightarrow C_{\mathcal{A}}(L)$ closed ($C_{\mathcal{A}}$ is closed).*
3. *for all $A \subseteq Q^\omega$, A is ω-regular $\Rightarrow C^-_{\mathcal{A}}(A)$ ω-regular.*
4. *for all $L \subseteq \Sigma^\omega$, L is ω-regular $\Rightarrow C_{\mathcal{A}}(L)$ ω-regular.*

Theorem 3. *For any deterministic complete automaton $\mathcal{A}$ recognizing a language L, the finitary restriction of this automaton $\mathsf{UniCloOmg}(\mathcal{A})$ recognizes $\mathsf{UniCloOmg}(L)$.*

Theorem 3 allows to extend all known results on deterministic classes to finitary deterministic classes: as a corollary, we have $DFB \subset DFP$ and $DFP = DFS$.

We now show that non-deterministic finitary parity automata are more expressive than deterministic finitary parity automata. However, for every language $L \in \mathbb{L}_\omega$ there exists $\mathcal{A} \in DP$ such that $\mathcal{A}$ recognizes L, and by Theorem 3 the deterministic finitary parity automaton $\mathsf{UniCloOmg}(\mathcal{A})$ recognizes $\mathsf{UniCloOmg}(L)$. Observe that Theorem 3 does not hold for non-deterministic automata, since we have $DP = NP$ but $DFP \subset NFP$.

Example 3 ($DFP \subset NFP$). As for Example 1 we consider the languages $L_1 = \{(a^{j_0}b^{f(0)}) \cdot (a^{j_1}b^{f(1)}) \cdot (a^{j_2}b^{f(2)}) \ldots \mid f : \mathbb{N} \to \mathbb{N}, f \text{ bounded}, \forall i \in \mathbb{N}, j_i \in \mathbb{N}\}$ and $L_2 = \{(a^{f(0)}b^{j_0}) \cdot (a^{f(1)}b^{j_1}) \cdot (a^{f(2)}b^{j_2}) \ldots \mid f : \mathbb{N} \to \mathbb{N}, f \text{ bounded}, \forall i \in \mathbb{N}, j_i \in \mathbb{N}\}$. It follows from Example 1 that both L_1 and L_2 belong to DFP, hence to NFP. A finitary parity automaton, relying on non-determinism, is easily built to recognize $L = L_1 \cup L_2$, hence $L \in NFP$. We can show that we cannot bypass this non-determinism, as by reading a word we have to decide well in advance which sequence will be bounded: a's or b's, *i.e*, $L \notin DFP$. To prove it, we interleave words of the form $(a^* \cdot b^*)^* \cdot a^\omega$ and $(a^* \cdot b^*)^* \cdot b^\omega$, and use a pumping argument to reach a contradiction.

4.3 Non-deterministic Finitary Automata

We can show that non-deterministic finitary Streett automata can be reduced to non-deterministic finitary Büchi automata, and this would complete the picture of expressive power comparison.

Our results are summarized in Corollary 2 and shown in Fig 2.

Corollary 2. *We have (a) $DFB \not\subseteq \mathbb{L}_\omega$; (b) $DFB \subset DFP = DFS \subset NFB = NFP = NFS$; (c) $DB \not\subseteq NFB$; (d) $\mathbb{L}_\omega \not\subseteq NFB$.*

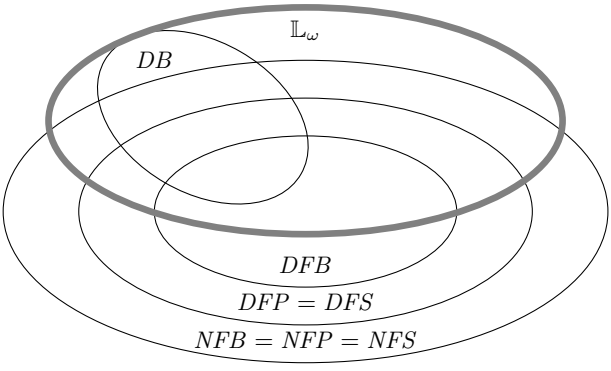


Fig. 2. Expressive power classification

4.4 Closure Properties

Theorem 4 (Closure properties). *The following closure properties hold:*

1. *DFP is closed under intersection.*
2. *DFP is not closed under union.*
3. *NFP is closed under union and intersection.*
4. *DFP and NFP are not closed under complementation.*

5 Regular Expression Characterization

In this section we address the question of giving a syntactical representation of finitary languages, using a special class of regular expressions.

The class of ωB-regular expressions was introduced in the work of [4] as an extension of ω-regular expressions, as an attempt to express bounds in regular languages. To define ωB-regular expressions, we need regular expressions and ω-regular expressions.

Regular expressions define regular languages over finite words, and have the following grammar:

$$L := \emptyset \mid \varepsilon \mid \sigma \mid L \cdot L \mid L^* \mid L + L; \quad \sigma \in \Sigma$$

In the above grammar, $\cdot$ stands for concatenation, $*$ for Kleene star and $+$ for union. Then ω-regular languages are finite unions of $L \cdot L'^{\omega}$, where L and L' are regular languages of finite words. The class of ωB-regular languages, as defined in [4], is described by finite union of $L \cdot M^{\omega}$, where L is a regular language over finite words and M is a B-regular language over infinite sequences of finite words. The grammar for B-regular languages is as follows:

$$M := \emptyset \mid \varepsilon \mid \sigma \mid M \cdot M \mid M^* \mid M^B \mid M + M; \quad \sigma \in \Sigma$$

The semantics of regular languages over infinite sequences of finite words will assign to a B-regular expression M, a language in $(\Sigma^*)^{\omega}$. The infinite sequence $\langle u_0, u_1, \ldots \rangle$ will be denoted by $\boldsymbol{u}$. The semantics is defined by structural induction as follows.

- $\emptyset$ is the empty language,
- ε is the language containing the single sequence $(\varepsilon, \varepsilon, \dots)$,
- a is the language containing the single sequence $(a, a, \dots)$,
- $M_1 \cdot M_2$ is the language $\{\langle u_0 \cdot v_0, u_1 \cdot v_1, \dots\rangle \mid \boldsymbol{u} \in M_1, \boldsymbol{v} \in M_2\}$,
- M^* is the language $\{\langle u_0 \dots u_{f(1)-1}, u_{f(1)} \dots u_{f(2)-1}, \dots\rangle \mid \boldsymbol{u} \in M, f : \mathbb{N} \to \mathbb{N}\}$,
- M^B is defined like M^* but we additionally require the values $f(i+1) - f(i)$ to be bounded uniformly in i,
- $M_1 + M_2$ is $\{\boldsymbol{w} \mid \boldsymbol{u} \in M_1, \boldsymbol{v} \in M_2, \forall i, w_i \in \{u_i, v_i\}\}$.

Finally, the ω-operator on sequences with nonempty words on infinitely many coordinates is: $\langle u_0, u_1, \dots\rangle^\omega = u_0 u_1 \dots$. This operation is naturally extended to languages of sequences by taking the ω power of every sequence in the language. The class of ωB-regular languages is more expressive than *NFB*, and this is due to the $*$-operator. We will consider the following fragment of ωB-regular languages where we do not use the $*$-operator for B-regular expressions (however, the $*$-operator is allowed for L, regular languages over finite words). We call this fragment the star-free fragment of ωB-regular languages.

Theorem 5. *NFB has exactly the same expressive power as star-free ωB-regular expressions.*

To prove that any language in *NFB* can be described by a star-free ωB-regular expression, we use the same lines as for ω-regular languages, except that a special attention is needed on size of final loops. The converse implication is more involved. We define acceptance conditions for automata reading infinite sequence of finite words, and proceed by induction on star-free B-regular expressions to build a finitary Büchi automaton that recognizes M^B. Then, we lift up automata reading infinite sequences of finite words to automata reading infinite words. This transformation is possible due to the *key*, yet simple observation that for all star-free B-regular expressions M and for all $\boldsymbol{v} \in M$ we have $(|v_n|)_n$ is bounded.

6 Decision Problems

In this section we consider the complexity of the decision problems for finitary languages. We present the results for finitary Büchi automata for simplicity, but the arguments for finitary parity and Streett automata are similar.

Theorem 6 (Decision problems). *The following assertions hold:*

1. (Emptiness). *Given a finitary Büchi automaton $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A}) = \emptyset$ is* NLOGSPACE*-complete and can be decided in linear time.*
2. (Universality). *Given a finitary Büchi automaton $\mathcal{A}$ whether $\mathcal{L}(\mathcal{A}) = \Sigma^\omega$ is* PSPACE*-complete.*
3. (Language inclusion). *Given two finitary Büchi automata $\mathcal{A}$ and $\mathcal{B}$, whether $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ is* PSPACE*-complete.*

We can show that a finitary Büchi automaton is empty if and only if it is empty regarded as a Büchi automaton. The PSPACE-hardness for universality and language inclusion follows from the special case of automata over finite words. For the PSPACE membership, we design a PSPACE algorithm for language inclusion (and universality follows as a special case), by performing a synchronous product of $\mathcal{A}$ and a subset construction of $\mathcal{B}$.

References

1. Alpern, B., Schneider, F.B.: Defining Liveness. Information Processing Letters 21(4), 181–185 (1985)
2. Alur, R., Henzinger, T.A.: Finitary Fairness. ACM Transactions on Programming Languages and Systems 20(6), 1171–1194 (1998)
3. Bojanczyk, M.: Beyond omega-regular languages. In: International Symposium on Theoretical Aspects of Computer Science, STACS, pp. 11–16 (2010)
4. Bojanczyk, M., Colcombet, T.: Bounds in ω-Regularity. In: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science, LICS 2006, pp. 285–296. IEEE Computer Society, Los Alamitos (2006)
5. Bojanczyk, M., Torunczyk, S.: Deterministic Automata and Extensions of Weak MSO. In: International Conference on the Foundations of Software Technology and Theoretical Computer Science, FSTTCS, pp. 73–84 (2009)
6. Büchi, J.R.: On a decision method in restricted second-order arithmetic. In: Proceedings of the 1st International Congress of Logic, Methodology, and Philosophy of Science, CLMPS 1960, pp. 1–11. Stanford University Press, Stanford (1962)
7. Chatterjee, K., Fijalkow, N.: Finitary languages. CoRR abs/1101.1727 (2011)
8. Chatterjee, K., Henzinger, T.A., Horn, F.: Finitary Winning in ω-regular Games. ACM Transactions on Computational Logic 11(1) (2009)
9. Choueka, Y.: Theories of automata on ω-tapes: A simplified approach. Journal of Computer and System Sciences 8, 117–141 (1974)
10. Gurevich, Y., Harrington, L.: Trees, Automata, and Games. In: Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC 1982, pp. 60–65. ACM Press, New York (1982)
11. Hummel, S., Skrzypczak, M., Toruńczyk, S.: On the Topological Complexity of MSO+U and Related Automata Models. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 429–440. Springer, Heidelberg (2010)
12. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, Heidelberg (1992)
13. Pnueli, A., Rosner, R.: On the Synthesis of a Reactive Module. In: Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages, POPL 1989, pp. 179–190 (1989)
14. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete-event processes. SIAM Journal on Control and Optimization 25(1), 206–230 (1987)
15. Safra, S.: Exponential Determinization for ω-Automata with Strong-Fairness Acceptance Condition. In: Annual ACM Symposium on Theory of Computing, STOC. ACM Press, New York (1992)
16. Thomas, W.: Languages, Automata, and Logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Beyond Words, vol. 3, pp. 389–455. Springer, Heidelberg (1997)
17. Wadge, W.W.: Reducibility and Determinateness of Baire Spaces. PhD thesis, UC Berkeley (1984)

The Complexity of Request-Response Games

Krishnendu Chatterjee[1], Thomas A. Henzinger[1], and Florian Horn[1,2]

[1] IST (Institute of Science and Technology), Austria
{krish.chat,tah}@ist.ac.at
[2] LIAFA, CNRS & Université Paris 7, France
horn@liafa.jussieu.fr

Abstract. We consider two-player graph games whose objectives are request-response condition, *i.e* conjunctions of conditions of the form "if a state with property Rq is visited, then later a state with property Rp is visited". The winner of such games can be decided in EXPTIME and the problem is known to be NP-hard. In this paper, we close this gap by showing that this problem is, in fact, EXPTIME-complete. We show that the problem becomes PSPACE-complete if we only consider games played on DAGs, and NP-complete or PTIME-complete if there is only one player (depending on whether he wants to enforce or spoil the request-response condition).

We also present near-optimal bounds on the memory needed to design winning strategies for each player, in each case.

1 Introduction

Games. Games played on graphs are suitable models for multi-component systems: vertices represent states; edges represent transitions; players represent components; and objectives represent specifications. The specification of a component is typically given as an ω-regular condition [6], and the resulting ω-regular games have been used for solving control and verification problems (see, e.g., [1,7,8]).

Fairness specifications. The class of fairness objectives is one of the most important specifications in verification and synthesis. The two classical notions of fairness are as follows: (a) strong fairness (or Streett) objectives, and (b) request-response (or assume-guarantee) objectives. The fairness objectives consist of a set of k pairs of requests and corresponding responses. The Streett objective requires that every request that appears infinitely often must be granted infinitely often. The request-response objective requires that every request that appears is granted after it appears. The class of Streett objectives is a canonical and widely used form of fairness specification [10,6]. The class of request-response (assume-guarantee) specifications was studied in [11], and it was shown that a wide range of practical specifications (such as an elevator controller) can be specified as request-response specifications.

Previous results. Games with Streett objectives have been widely studied and optimal bounds on computational complexity and memory required by winning

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 227–237, 2011.

strategies have been established. The winner problem in games with Streett objectives with k request-response pairs is coNP-complete [5]. The memory bound for winning strategies is as follows: there is an optimal (matching lower and upper) bound of $k!$ for the size of memory for the player with the Streett objective and the opposing player has a memoryless winning strategy (a strategy that is independent of the history and depends only on the current vertex) [4]. In contrast, for games with request-response objectives there are gaps both in the computational complexity bounds, and memory bounds required for winning strategies. Games with request-response objectives can be solved in EXPTIME [11] and are NP-hard [3]. The winning strategies for the player with request-response objective require a memory of size at least $\lfloor k/3 \rfloor \cdot 2^{\lfloor k/3 \rfloor}$ and memory of size $k \cdot 2^{k+1}$ suffices for winning strategies for both players [11].

Our results. We present tight computational complexity bounds for request-response games, and present near optimal bounds on memory required by winning strategies. Our results are as follows:

1. We first show that games with request-response objectives are EXPTIME-complete (improving the NP-hardness lower bound). In the study of turn-based deterministic games with classical objectives such as Rabin, Streett, Muller the complexities are NP-complete, coNP-complete, PSPACE-complete, respectively [10]. For turn-based games, several EXPTIME-completeness results are known for more general class of games such as pushdown games [12] or imperfect-information games [9]. We show that for perfect-information finite-state turn-based deterministic games, a natural variant of Streett objectives lead to EXPTIME-completeness. The EXPTIME-hardness results for pushdown or imperfect-information games are either due to the infinite store (stack) or the imperfect-information, whereas our proof is different and shows how to exploit the simple extension of Streett objectives to request-response objectives to mimic runs of alternating polynomial space Turing machines.
2. For the special class of DAG-games we show that request-response objectives are PSPACE-complete. We also study the complexity of one player game graphs: if there is only one player with request-response objectives, then the problem is NP-complete; and if there is only the opposing player, then the problem can be solved in polynomial time.
3. We improve the lower bounds for memory required for winning strategies in games with request-response objectives: we show that the protagonist player (whose goal is to enforce the request-response objective) requires $2^k - 1$ and the antagonist (opposing) player requires 2^k memory states, (improving the lower bound of $\lfloor k/3 \rfloor \cdot 2^{\lfloor k/3 \rfloor}$ for the protagonist player, and no bound was known for the opposing player). With a very simple argument we show the construction of [11] can be used to obtain an upper bound $k \cdot 2^k$ for the protagonist and 2^k for the antagonist. Thus our lower bound of $2^k - 1$ almost matches the upper bound of $k \cdot 2^k$ for the protagonist, and our bound of 2^k for the opposing player is a tight bound. Thus, we present almost optimal bounds on memory required by winning strategies. For DAG-games

with request-response objectives, we prove an optimal (matching upper and lower) bound of memory size $\binom{k}{\lfloor k/2 \rfloor}$ for winning strategies of both players.

2 Definitions

Arenas and plays. A (finite) game *arena* $\mathcal{A}$ is a tuple $((\mathcal{V}, \mathcal{E}), \mathcal{V}_\circ, \mathcal{V}_\Box)$, where $(\mathcal{V}, \mathcal{E})$ is a finite graph and $\mathcal{V}_\circ, \mathcal{V}_\Box$ is a partition of $\mathcal{V}$. The vertices in $\mathcal{V}_\circ$ are called *Eve's vertices* and those in $\mathcal{V}_\Box$ are called *Adam's vertices*. For a vertex u in $\mathcal{V}$, we denote by $\mathcal{E}(u)$ the set of successors of u: $\mathcal{E}(u) = \{v \in \mathcal{V} \mid (u, v) \in \mathcal{E}\}$. We assume that every vertex has at least one successor. A play ρ on an arena $\mathcal{A}$ is a (possibly infinite) sequence $\rho = \rho_0, \rho_1, \ldots$ of vertices respecting the edge relation: for all $i \geq 0$ we have $(\rho_i, \rho_{i+1}) \in \mathcal{E}$.

Strategies. Intuitively, a strategy is a recipe to extend plays. Formally, a *strategy* σ for Eve is a function $\sigma : \mathcal{V}^* \cdot \mathcal{V}_\circ \to \mathcal{V}$ such that for all finite plays (or histories) x ending in a vertex v of Eve, $\sigma(x)$ is a successor of v. Strategies for Adam are defined analogously (and are usually denoted τ).

An equivalent definition of strategies uses the notion of *memory*. A *strategy with memory* σ for Eve is a tuple $(\sigma^{\mathtt{M}}, \sigma^{\mathtt{i}}, \sigma^{\mathtt{n}}, \sigma^{\mathtt{u}})$ where $\sigma^{\mathtt{M}}$ is the set of *memory states*, $\sigma^{\mathtt{i}} \in \sigma^{\mathtt{M}}$ is the *initial memory state*, $\sigma^{\mathtt{n}} : \mathcal{V}_\circ \times \sigma^{\mathtt{M}} \to \mathcal{V}$ is the *next-move function*, and $\sigma^{\mathtt{u}} : \mathcal{V} \times \sigma^{\mathtt{M}} \to \sigma^{\mathtt{M}}$ is the *memory update function*. Notice that any strategy can be represented as a strategy with memory $\mathcal{V}^*$. A strategy σ has finite memory if $\sigma^{\mathtt{M}}$ is finite (in this case, $|\sigma^{\mathtt{M}}|$ is the *size of* σ); it is memoryless if $\sigma^{\mathtt{M}}$ is a singleton. Notice that a memoryless strategy for Eve is independent of the history of the play and depends only on the current vertex, and hence can be described as a function from $\mathcal{V}_\circ$ to $\mathcal{V}$ respecting the edge relation. The notation for strategies with memory and memoryless strategies for Adam is analogous.

A play ρ is *consistent with* σ if for all $i \geq 0$ such that ρ_i belongs to Eve we have ρ_{i+1} is $\sigma(\rho_0, \rho_1, \ldots, \rho_i)$. Given an initial vertex $v \in \mathcal{V}$, a strategy σ for Eve and a strategy τ for Adam, we denote by $\rho(v, \sigma, \tau)$ the unique infinite play starting in v and consistent with σ and τ.

Request-response objectives. A *winning condition (objective)* Φ for an arena $\mathcal{A}$ is a subset of the plays on the arena. In this paper, we consider the request-response objectives introduced by Wallmeier, Hütten, and Thomas in [11]. It refers to vertex properties $\mathsf{Rq}_1, \ldots, \mathsf{Rq}_k$ which capture k different "requests", and other vertex properties $\mathsf{Rp}_1, \ldots, \mathsf{Rp}_k$ which represent the corresponding "responses" (each $\mathsf{Rq}_i, \mathsf{Rp}_i \subseteq \mathcal{V}$). The associated request-response condition requires that *for each i, whenever a vertex in Rq_i is visited, then later a vertex in Rp_i is visited.* In linear time temporal logic (LTL) the condition is more often formalized as $\wedge_{i=1}^{k} \mathrm{G}(\mathsf{Rq}_i \to \mathrm{XF}(\mathsf{Rp}_i))$, where G, X, and F denote *globally*, *next*, and *eventually*, respectively. The Streett objective in LTL is described as $\wedge_{i=1}^{k} (\mathrm{G\ F}(\mathsf{Rq}_i) \to \mathrm{G\ F}(\mathsf{Rp}_i))$.

A strategy σ is *winning for Eve from a vertex v* in a game $\mathcal{G} = (\mathcal{A}, \Phi)$ if, for any strategy τ for Adam, the play $\rho(v, \sigma, \tau)$ belongs to Φ. A strategy τ is winning for Adam from a vertex v if for all strategies σ, the play $\rho(v, \sigma, \tau)$ belongs to $\neg\Phi = \Pi \setminus \Phi$. The *winning region of Eve in* $\mathcal{G}$, denoted $W_E(\Phi)$, is the

set of vertices from which Eve has a winning strategy, and the winning region for Adam, denoted $W_A(\neg\Phi)$, is defined similarly.

Theorem 1 (Determinacy [11]). *For all request-response games, for all vertices v, either Eve or Adam has a winning strategy from v.*

3 Complexity of Request-Response Games

In this section, we consider the computational complexity of request-response games in general. Our main result is an EXPTIME lower bound in complexity, matching the EXPTIME membership from [11]. We also study the complexity of the winning strategies in terms of memory, and provide near optimal bounds for both players.

3.1 Request-Response Games Are EXPTIME-Complete

In [11], the authors show that request-response games can be solved in EXPTIME, but they do not provide any lower bound in complexity. In this subsection, we show that the problem is in fact EXPTIME-hard, through a reduction from the membership problem for alternating polynomial space Turing machines.

An *alternating Turing machine* (ATM) is a tuple $(Q, q_{\mathsf{in}}, Q_\vee, Q_\wedge, \mathcal{I}, \delta, q_{\mathsf{acc}})$ where:

- Q is a finite set of control states, partitioned into existential ($Q_\vee$) and universal ($Q_\wedge$) states;
- $q_{\mathsf{in}} \in Q$ is the initial state;
- $\mathcal{I} = \{0, 1\}$ is the tape alphabet;
- $\delta \subseteq Q \times \mathcal{I} \times Q \times \mathcal{I} \times \{-1, 1\}$ is the transition relation;
- $q_{\mathsf{acc}} \in Q$ is the accepting state.

For a given polynomial p, the question of whether an ATM M accepts a word w in space at most $p(|w|)$ is EXPTIME-complete [2]. We reduce this problem to the winner problem of request-response games. The idea is that the players build a run of the machine: Eve controls the existential states and Adam the universal ones; if the run reaches an accepting state, Eve wins; if it goes on forever, Adam does. A winning strategy for Eve in the game translates as an accepting run tree of the machine.

We use $p(|w|)$ copies of the control graph of the machine in order to store the current location of the head. However, the arena does not store the *content* of the tape. Instead, at each step, Eve announces the current symbol and Adam either accepts it or challenges it. If he does the latter, the play stops: Eve wins if she has been truthful; Adam wins if she cheated. This interaction is described in Figure 1.

We use request-response pairs to force Eve to announce the correct symbol at each step. There is a pair ℓ^s for each location ℓ and each symbol s. An extra pair \$ guarantees that the correct simulation of an infinite run is winning for Adam.

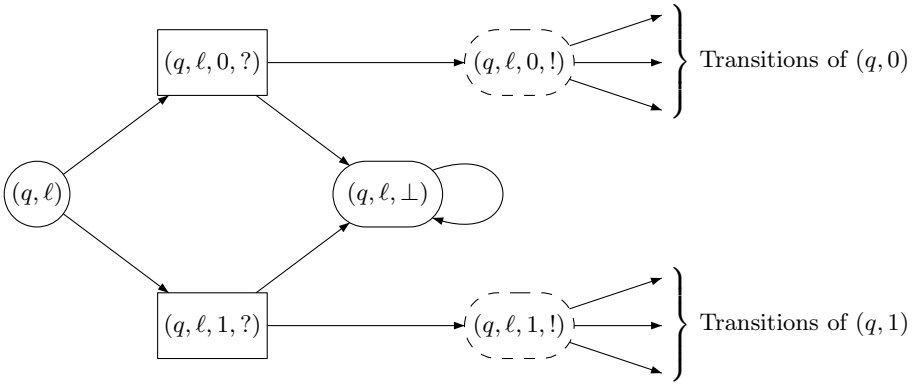


Fig. 1. The consistency gadget at a state (q, ℓ)

The idea is that whenever a symbol s is written on the location ℓ, a request of type ℓ^s is generated. The play begins in the vertex start, which is a request of type i^{w_i} for each $i \in \{0, \ldots, |w| - 1\}$, as well as a request of type \$. Then the token goes to $(q_{\mathsf{in}}, 0)$ for the first step.

At the beginning of a step where the machine is in the control state q and its head is at the ℓ^{th} cell, the token is in the vertex (q, ℓ), which belongs to Eve. Her first task is to announce the contents of the cell. She does so by granting either ℓ^0 or ℓ^1 (by going to $(q, \ell, 0, ?)$ or $(q, \ell, 1, ?)$, respectively). At this point, Adam can challenge her choice by sending the token to $(q, \ell, \bot)$, which is a sink where all the pairs except for ℓ^0 and ℓ^1 are granted. Thus, if Eve has announced the correct symbol, she wins; otherwise, either ℓ^0 or ℓ^1 is left pending and she loses. If Adam chooses to accept Eve's claim, the token goes to the vertex $(q, \ell, i, !)$, which belongs to Eve if q is an existential state and to Adam if q is a universal state. There, the controlling player chooses a transition of the form $t = (q, i, r, j, \pm)$ by going to the vertex (t, ℓ), which generates a request of type ℓ^j. The token goes then to the vertex $(r, \ell \pm 1)$ for the next step, unless $r = q_{\mathsf{acc}}$, in which case the token goes to the sink vertex stop, which grants all the requests.

Let us show that Eve has a winning strategy in $\mathcal{G}$ if, and only if, M accepts w. We call *honest* a strategy for Eve which always calls the correct symbol in vertices of the form (q, ℓ), and *trusting* a strategy for Adam which never challenges the choices of Eve. It is clear that any winning strategy of Eve has to be honest, and that an honest strategy of Eve is winning if and only if it is winning against any trusting strategy of Adam.

There is a natural bijection between (i) plays consistent with an honest strategy for Eve and a trusting one for Adam and (ii) runs of M on w. It can be extended to a bijection between the honest strategies of Eve and the run trees of M on w, which matches winning strategies and accepting run trees. Thus Eve has a winning strategy if and only if M accepts w. It follows that the problem of deciding the winner in request-response games is `EXPTIME`-hard. As it is known to belong to `EXPTIME` [11], Theorem 2 follows:

Theorem 2. *The problem of deciding the winner in a request-response game is* EXPTIME*-complete.*

3.2 Strategy Complexity

We consider now the complexity, in terms of memory, of the winning strategies for both players. In [11], the reduction to Büchi games yields strategies with memory $k \cdot 2^k$. It is possible to improve these bounds a little with two simple observations:

- In the original reduction, the arena keeps track of the pending requests (2^k memory) and of an "active" pair which must be satisfied next (k memory); Eve does not need to discriminate between the vertices where the active pair is not pending, so she only needs $\sum_{i=0}^{k} i \cdot \binom{k}{i} = k \cdot 2^{k-1}$.
- By replacing the Büchi condition with a generalized Büchi conditions (with k target sets), one can get rid of the "active pair" tracker; Adam still has memoryless winning strategies in the reduced game, so he only needs 2^k memory states in the original request-response game.

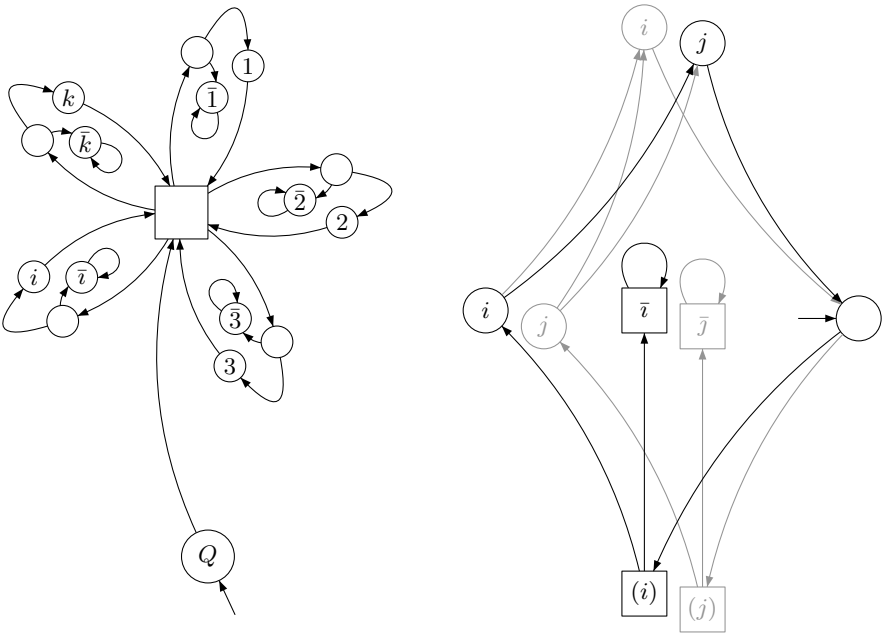
The authors presented only a lower bound for Eve, who was shown to need $2^{\lfloor k/3 \rfloor}$ memory states. In this section, we improve and complete this picture with a better lower bound for Eve ($2^k - 1$), and a tight bound for Adam (2^k). The games realizing the lower bounds are presented in Figure 2.

In the game of Figure 2(a), the vertex labelled Q is a request of each type; for each i, a vertex labelled i is a response of type i, and a vertex labelled $\bar{i}$ is a response of every type but i. Intuitively, a play is divided in steps in which Adam first chooses a pair and Eve then grants either this pair (and the play continues) or all the others (and the play stops). It is clear that Eve can win with the following strategy: the first time Adam chooses the i^{th} petal, she grants the pair i; the second time, she grants all the other pairs[1]. We show that Adam can defeat any strategy with less than $2^k - 1$ memory states.

Let $\sigma = (\sigma^{\mathtt{M}}, \sigma^{\mathtt{i}}, \sigma^{\mathtt{n}}, \sigma^{\mathtt{u}})$ be a strategy for Eve with less than $2^k - 1$ memory states. For each memory state m, we define the stopping set $\chi(m)$ of m as the set of petals where Eve would stop the play if Adam chose them (notice that m is the memory of Eve in the "heart" vertex: it might change after Adam has made his choice, but her behaviour is still determined by m and the petal that Adam chose). As there are less than $2^k - 1$ memory states, there is a strict subset X of $\{1, \ldots, k\}$ which is not the stopping set of any memory state. Now, Adam can win against σ by choosing, at each step, a petal in the symmetric difference of X and $\chi(m)$, where m is Eve's current memory under σ. Such a play can either go on forever if Adam keeps to petals in X, or stop the first time he gets out. In either case, there is at least one request outside of X which is never granted.

In the game of Figure 2(b), the arena has $4k + 1$ vertices: there is one copy of the bottom, middle, left, and top vertices for each request-response pair.

[1] This strategy uses 2^k memory states, but it is clear that there is no actual need for a specific memory state to remember that every petal has been visited: in this case, the play is already won, no matter what Eve does later on.



(a) Eve: $2^k - 1$ (b) Adam: 2^k

Fig. 2. Lower bounds in memory

For each pair, say the i^{th}, the vertices labelled i) (left and top) are both requests and responses (recall that requests have to be granted in the strict future); the vertex labelled $\bar{\imath}$ (middle) is a response of every type but i. The right and bottom vertices are neither requests nor responses of any type (the label (i) of the bottom vertex only serves as a reminder that there are k different copies of this vertex). From the initial vertex (on the right), Eve can go to any of the bottom vertices; likewise, from *each* left vertex, Eve can go to any of the top vertices. By contrast, in a bottom vertex, say (i), Adam has to go to either the left vertex i or the middle vertex $\bar{\imath}$. A step of this game can be described by the three following actions: Eve chooses a pair, say i; Adam either grants it and requests it again (and the play continues), or grants every other pair (and the game stops); Eve then chooses a (possibly different) pair, say j, grants it, and requests it again. Adam can win by stopping the game the second time a pair is requested (thus with 2^k memory states), and we show that he cannot win with less.

Let $\tau = (\tau^{\mathtt{M}}, \tau^{\mathtt{i}}, \tau^{\mathtt{n}}, \tau^{\mathtt{u}})$ be a strategy for Adam with less than 2^k memory states. For a memory state m in $\tau^{\mathtt{M}}$, we define the stopping set $\chi(m)$ of m as the sets of bottom vertices where Adam would stop the play if Eve chose them (once again, m is the memory of Adam in the right vertex: it might change after Eve's choice, but Adam's behaviour is determined by m and this choice). As there are less than 2^k memory states, there is a subset X of $\{1, \dots, k\}$ which is not the stopping set of any memory state. Now, Eve can win against τ by choosing, at

each step, a pair in the symmetric difference of X and $\chi(m)$, where m is Adam's current memory under τ in the right vertex and cycling through the pairs in X in the left one. Such a play can either go on forever if Eve keeps to petals in X, or stop the first time she gets out. In both cases, only requests in X are enabled. Furthermore, in the first case, each such request is granted infinitely often; in the second case, each request in X is granted when the play stops, and never enabled again.

Theorem 3. *In any request-response game with k request-response pairs, wherever Eve has a winning strategy, she has a winning strategy with memory $k \cdot 2^{k-1}$; wherever Adam has a winning strategy, he has a winning strategy with memory 2^k. Furthermore, there is a request-response game with k request-response pairs in which Eve can only win with at least $2^k - 1$ memory states, and one where Adam can only win with at least 2^k memory states.*

4 Restrictions

In this section, we consider two special types of request-response games, where the winner problem is easier to solve.

4.1 DAG Arenas

The first one is the case where the arenas have the form of a directed acyclic graph (no cycles apart from loops on vertices with no other successors). By contrast to the usual study of "long-term" behaviours, these games focus on "short-term" objectives. We show that request-response games played on DAG-arenas are PSPACE-complete, and provide tight bounds for the memory required of each player.

As with most games played on DAG arenas, it is possible to solve the winner problem in polynomial space, by enumerating the plays in lexicographic order. We show PSPACE-hardness through a reduction from the truth problem of quantified boolean formulae. From a QBF in conjunctive normal form with k variables, we derive a request-response game with $3 \cdot k + 1$ vertices as follows: there is a vertex for each variable *and* one for each literal; the "variable" vertex leads to the two corresponding "literal vertices, and belongs to Eve if the variable is existential or to Adam if is is universal; there is a request-response pair for each clause, which is requested at the beginning of the play and solved at each literal present in the clause. For a QBF of the form $\exists x_1, \forall x_2, \ldots, \exists x_k$, the resulting game is described in Figure 3 (the vertex $\mathcal{C}$ is a request of each type).

Theorem 4 follows:

Theorem 4. *The problem of deciding the winner in request-response games played on DAG arenas is* PSPACE*-complete.*

The restriction to DAG arenas also affects the complexity of strategies. Theorem 5 provides tight bounds for both players:

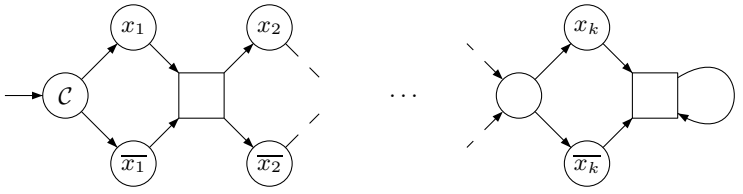


Fig. 3. Reduction from QBF to request-response games on DAG arenas

Theorem 5. *In any request-response game with k request-response pairs played on a DAG-arena, wherever a player has a winning strategy, he has a winning strategy with memory $\binom{k}{\lfloor k/2 \rfloor}$. Furthermore, for each player there is a request-response game with k request-response pairs in they can only win with at least $\binom{k}{\lfloor k/2 \rfloor}$ memory states.*

Proof. In order to devise a winning strategy for either player, it is enough to remember the current set of unanswered requests: as all the plays are finite, they are winning if, and only if, they end with all their requests answered. Furthermore, there is no need to keep two separate memory states for two sets A and B of pairs such that $A \subset B$: if Eve can win in both cases, she can do so in both cases by playing as if the set of unanswered requests was B; symmetrically, Adam can win by playing in both cases as if the set of unanswered requests was A. As there are at most $\binom{k}{\lfloor k/2 \rfloor}$ incomparable subsets of $\{1, \ldots, k\}$, both Eve and Adam can win with memory $\binom{k}{\lfloor k/2 \rfloor}$ in any request-response game with k request-response pairs.

A family of arenas where this much memory is necessary can be described as follows:

- *Eve.* Adam can choose $\lfloor k/2 \rfloor$ requests. Then Eve can choose $\lfloor k/2 \rfloor$ responses. It is clear that she must choose the exact the same subset that Adam chose. As there are $\binom{k}{\lfloor k/2 \rfloor}$ possibilities, she needs memory $\binom{k}{\lfloor k/2 \rfloor}$.
- *Adam.* All pairs are initially requested. Eve can choose $\lfloor k/2 \rfloor$ responses, then Adam chooses $\lfloor k/2 \rfloor$ requests. Finally, Eve can choose $k-1$ responses. Again, Adam needs to match the subset that Eve chose, so he needs $\binom{k}{\lfloor k/2 \rfloor}$ memory states.

Theorem 5 follows. □

4.2 One-Player Arenas

One-player games correspond to the synthesis of controllable systems, with no interaction from the environment. Game problems are usually much simpler in this case. For example, if the player tries to ensure a request-response specification, the winner problem becomes NP-complete:

Theorem 6. *The problem of deciding whether Eve has a winning strategy in a one-player request-response game is* NP*-complete.*

Proof. We can reduce SAT to one-player games using the same reduction that we used in the former section: if all the quantifiers are existential, all the vertices in the resulting game belong to Eve. Thus the problem is NP-hard.

In order to describe a NP procedure to solve this problem, first observe that a play always consists of a finite path w followed by infinite occurrences of all the vertices in a strongly connected component C. It is winning for Eve if every request unresolved in w or present in C is matched by a corresponding response in C. The crux of the proof is the fact that we can always choose w of size at most $(k+1) \cdot |\mathcal{V}|$ by removing from it all the cycles which do not contain the *last* occurrence of a response. We can thus guess non-deterministically both w and C, and the NP-membership follows. □

If the player is trying to spoil, rather than ensure, a request-response objective, the winner problem can be decided in polynomial time:

Theorem 7. *The problem of deciding whether Adam has a winning strategy in a one-player request-response game is* PTIME*-complete.*

Proof. The PTIME hardness comes from the trivial reduction from alternating reachability. In order to describe a PTIME procedure, observe that in order to win, Adam needs only to reach a request from which he can avoid the corresponding response. As safety and reachability winning regions can be computed in polynomial time, so can be the winning region of Adam in a one-player game where he controls all the vertices. □

References

1. Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. JACM 49, 672–713 (2002)
2. Chandra, A.K., Kozen, D., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981)
3. Chatterjee, K., Henzinger, T., Horn, F.: Finitary winning in ω-regular games. ACM ToCL 11(1) (2009)
4. Dziembowski, S., Jurdziński, M., Walukiewicz, I.: How much memory is needed to win infinite games? In: Logic In Computer Science, pp. 99–110. IEEE Computer Society, Los Alamitos (1997)
5. Emerson, E., Jutla, C.: The complexity of tree automata and logics of programs. In: Foundations of Computer Science, pp. 328–337. IEEE Computer Society, Los Alamitos (1988)
6. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, Heidelberg (1992)
7. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL, pp. 179–190. ACM, New York (1989)
8. Ramadge, P., Wonham, W.: Supervisory control of a class of discrete-event processes. SIAM Journal of Control and Optimization 25, 206–230 (1987)

9. Reif, J.H.: The complexity of two-player games of incomplete information. Journal of Computer and System Sciences 29(2), 274–301 (1984)
10. Thomas, W.: Languages, automata, and logic. Handbook of Formal Languages 3, 389–455 (1997)
11. Wallmeier, N., Hütten, P., Thomas, W.: Symbolic synthesis of finite-state controllers for request-response specifications. In: Ibarra, O.H., Dang, Z. (eds.) CIAA 2003. LNCS, vol. 2759, pp. 11–22. Springer, Heidelberg (2003)
12. Walukiewicz, I.: Pushdown processes: Games and model-checking. Inf. Comput. 164(2), 234–263 (2001)

Improved Alignment Based Algorithm for Multilingual Text Compression*

Ehud S. Conley[1,2] and Shmuel Tomi Klein[2]

[1] Jerusalem College of Technology, Jerusalem, Israel
[2] Bar-Ilan University, Ramat Gan, Israel
{conley@jct,tomi@cs.biu}.ac.il

Abstract. Multilingual text compression exploits the existence of the same text in several languages to compress the second and subsequent copies by reference to the first. This is done based on bilingual text alignment, a mapping of words and phrases in one text to their semantic equivalents in the translation. A new *multilingual* text compression scheme is suggested, which improves over an immediate generalization of bilingual algorithms. The idea is to store the necessary markup data within the source language text; the incurred compression loss due to this overhead is smaller than the savings in the compressed target language texts, for a large enough number of the latter. Experimental results are presented for a parallel corpus in six languages extracted from the EUR-Lex website of the European Union. These results show the superiority of the new algorithm as a function of the number languages.

1 Introduction

In countries like Canada, Belgium and Switzerland, where speakers of two or more languages live side-by-side, all official texts have to be published in multilingual form. The current legislation of the ever expanding European Union obliges the translation of all official texts into the languages of all member states. As a result, there is a growing corpus of important texts, large parts of which are highly redundant, since they do not have any information content of their own, and are just transformed copies of some other parts of the text collection.

We wish to exploit this redundancy to improve compression efficiency in such situations, and introduce the notion of *Multilingual Text Compression*: one is given two or more texts, which are supposed to be translations of each other and are referred to as *parallel* texts. One of the texts will be stored on its own (or compressed by means of pointers referencing only the text itself), the other texts can be compressed by referring to the translation, using appropriate dictionaries.

The basis for enabling multilingual text compression is first the ability to match the corresponding parts of related texts by identifying semantic correspondences across the various sub-texts, a task generally referred to as *alignment*. As the methods for detailed alignment are quite sensitive to noise, they

* This work has been done while the first author was a PhD student at Bar Ilan University.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 238–249, 2011.

usually use a rough alignment of the text as an auxiliary input. They might also use an existing multilingual glossary, but they always generate their own probabilistic glossary, which corresponds to the processed text (see [3,6,9,2]).

The compression of parallel texts was first treated almost two decades ago in [12], but without using text alignment tools. The idea of using alignment was raised in [5], and a detailed algorithm was presented in [4]. Different, though basically similar algorithms have recently been suggested in [11,1]. However, all these algorithms relate only to bilingual parallel texts; therefore, should three or more parallel texts be compressed, the algorithms would be applied to each source-target pair of texts independently.

The current work proposes a new compression scheme for parallel texts, referred to as *multilingual*. Similar to our previous *bilingual* algorithm [4], the new method also exploits alignment to increase space efficiency. However, the current algorithm stores the information regarding the aligned source-text fragments within the source text rather than within each of the target texts. On one hand, significant savings are made for each target text, but on the other hand, a large overhead is paid for the source text. Nevertheless, major parts of the additional information are shared by many of the bilingual alignments and the incurred overhead converges to a constant rate for a large-enough number of target texts, which makes it worthy paying in such cases, as our empirical results clearly confirm.

The next section presents the details of the algorithm, and experimental results are reported in Section 3.

2 Multilingual Compression Algorithm

Reminiscent of the bilingual algorithm, the multilingual algorithm assumes the following resources:

1. S, T^1, T^2, …, T^k: The single source and k target texts, respectively, where $\forall h \in [1,k]$, T^h is a translation of S.
2. A_{S,T^1}, A_{S,T^2}, …, A_{S,T^k}: Word- and phrase-level alignments of the text pairs (S,T^1), (S,T^2), …, (S,T^k), correspondingly.
   Let $s_{i,l}$ denote the word sequence of length l within S beginning at the ith word. Similarly, let $t^h_{j,m}$ denote the word sequence of length m within T^h beginning at the jth word. A_{S,T^h} consists of a set of connections of the form $\langle i,l,j,m\rangle$, each of which indicating the fact that $s_{i,l}$ and $t^h_{j,m}$ have been determined as matching phrases. We assume that for any pair (j,m) there is at most one connection of the form $\langle i,l,j,m\rangle$ within A_{S,T^h}. From here and below, s_i and t^h_j stand for $s_{i,1}$ (the ith word of S) and $t^h_{j,1}$ (the jth word of T^h), correspondingly.
3. S^{lem}, $(T^1)^{lem}$, $(T^2)^{lem}$, …, $(T^k)^{lem}$: Lemmatized forms of S, T^1, T^2, …, T^k, respectively.
   Let $(s_{i,l})^{lem}$ and $(t^h_{j,m})^{lem}$ denote the lemma sequences corresponding to $s_{i,l}$ and $t^h_{j,m}$, respectively. That is, the concatenations of the lemmata of s_i, s_{i+1}, …, s_{i+l-1} and t^h_j, t^h_{j+1}, …, t^h_{j+m-1}, correspondingly.

4. L_S: A lemmata dictionary corresponding to S. The entries of this dictionary are the words appearing in S. Each entry stores a list of all possible lemmata of the keyword, sorted in descending order of frequency.

   Let $L_S(s)$ denote the lemma list for the word s. For instance, if S is an English text, then $L_S(\texttt{working}) = (\texttt{work}, \texttt{working})$.
5. V_{T^1}, V_{T^2}, ..., V_{T^k}: Variant dictionaries corresponding to the target texts. For each $h \in [1, k]$, the entries of V_{T^h} are the lemmata of all words appearing in T^h. Each entry stores a list of all possible morphological variants of the key lemma, sorted in descending order of frequency.

   Let $V_{T^h}(t)$ denote the variant list for lemma t. For example, if T^3 is a French text, then $V_{T^3}(\texttt{normal}) = (\texttt{normal}, \texttt{normale}, \texttt{normaux}, \texttt{normales})$.
6. $G_{S,T^1}, G_{S,T^2}, \ldots, G_{S,T^k}$: Bilingual glossaries corresponding to the text pairs (S, T^1), (S, T^2), ..., (S, T^k), respectively. The entries of these glossaries are source-language lemma sequences. Each entry includes a list of possible translations of the key sequence into target-language sequences, sorted in descending order of frequency. The translations also appear in lemmatized form.

   Let $G_{S,T^h}(s)$ denote the translation list of the source-language sequence s into the language of T^h. For instance, if S and T^3 are English and French texts, correspondingly, then $G_{S,T^3}(\texttt{mineral water}) = (\texttt{eau mineral})$. Note that the word `eau` (water) in French is feminine, which requires a feminine-form adjective, namely `minerale`, whereas the adjective `mineral`—the corresponding lemma—is the masculine singular form.

The new algorithm encodes all aligned source sequences appearing in any of the k alignments within the source text. This is done using some special codewords, each indicating a different sequence length, inserted just before the first words of the aimed sequences within the text run. When more than one sequence begins at the same word, the respective annotations are inserted one after another preceding the start word. The order by which such successive marks are introduced can be determined arbitrarily. However, the compression procedure for the target texts relates to the order of these annotations when computing offset values for the aligned target sequences. Hence, this order must be decided before any target text is compressed.

The fact that the entries of the bilingual glossaries are lemmata sequences means that the aligned source fragments must be stored in a way that enables retrieving the lemma of each aligned word. Keeping only the lemmatized version of the source text (S^{lem}) is unacceptable because the original source text, being an integral part of the multilingual corpus, must be restorable like the k target texts. Due to space-efficiency consideration, we have chosen to store the inflected form of each source word along with its lemma index, if more than one lemma exists.

The output of the annotation procedure can be compressed using any encoding, for instance, Huffman coding with two Huffman trees: H_1^S will store the words and aligned-sequence marks, whereas H_2^S will hold the lemma indices.

Following the annotation of the source text, each aligned source sequence can be referred to using its ordinal number. At that point, each target text T^h may be compressed independently using the algorithm given below.

Beginning at the first position $j = 1$ within T^h, use A_{S,T^h} to find the longest sequence $t^h_{j,m}$ having a corresponding sequence $s_{i,l}$ in S. The redundant connections can be removed in advance, thereby avoiding redundant source sequences and their direct and indirect overheads. If a corresponding sequence is found, replace $t^h_{j,m}$ with the concatenation of a pointer to $s_{i,l}$ with some indices, which are necessary for the restoration of the missing target words. The substituting reference consists of the following details:

1. $ord\,(i,l) - expected$: Offset of $s_{i,l}$ from the expected sequence. This value is actually a pointer to $s_{i,l}$. $\mathrm{ord}\,(i,l)$ denotes the ordinal number of $s_{i,l}$ among the aligned sequences annotated within S. Initially, $expected = 0$; after using a connection $\langle i,l,j,m\rangle$, *expected* is assigned the ordinal number of the next sequence still not used which follows $s_{i,l}$. As an example, if $\mathrm{ord}\,(i,l) = 4$, and the indices of the aligned source sequences used so far are 1, 2, 5 and 8, then the new value of *expected* should be 6: the sequence indexed 5 is skipped since we assume in this example that it has already been used. 4 should now join the set of used sequences.
2. Index of $(t^h_{j,m})^{lem}$ within $G_{S,T^h}(s^{lem}_{i,l})$. In the case of a single translation, this index is omitted (same as emitting ϵ).
3. Indices of $t^h_j \ldots t^h_{j+m-1}$ within $V_{T^h}((t^h_j)^{lem}) \ldots V_{T^h}((t^h_{j+m-1})^{lem})$, correspondingly. Again, ϵ is used in the case of singletons.

The above reference is output preceded by a special codeword meaning "reference". The next iteration will work for $j = j + m$.

If no m is found such that $\langle i,l,j,m\rangle \in A_{S,T^h}$, t^h_j is written to the output stream and j is incremented by 1. The process continues while $j \leq |T^h|$.

The way the expected source sequence (*expected*) is determined is derived from the nature of the alignment. Given a pair of aligned source and target sequences, it is probable that the source counterpart of the next target sequence be the next source sequence. Of course, differences in word and phrase ordering as well as the existence of additional source sequences, aligned with sequences in other target texts, often yield some small deviations, which are overcome using the offset values. Another quite realistic assumption is that a source sequence already referred to once during the process will be rarely used again. The search for the next unreferenced sequence is always very local, meaning that it is performed in time $O(1)$.

For a single alignment, the expected source sequence for target number n could be simply n itself. The unification of source sequences from k alignments into one common list does not permit such an assumption. In this case, relating to the last known connection as an anchor point is a more conceivable heuristics.

The output for each target text T^h can be encoded using three Huffman codes: $H_1^{T^h}$ will contain the unaligned words as well as the special `#REF#` escape sequence, $H_2^{T^h}$ will hold the offset values, and $H_3^{T^h}$ shall store the translation

n	S (English)	T^1 (French)	A_{S,T^1}	T^2 (German)	A_{S,T^2}
1	For	Aux		Im	
2	the	fins	⟨3,1,2,1⟩	Sinne	
3	purpose	de		dieses	
4	of	la		Gemeinsamen	⟨6,1,4,1⟩
5	this	présente		Standpunkts	⟨7,1,5,1⟩
6	common	position	⟨6,2,6,2⟩	bedeutet	
7	position	commune		der	
8	,	,		Ausdruck	
9	the	on		:	
10	following	entend			
11	definitions	par			
12	shall	:			
13	apply				
14	:				

Fig. 1. Example of paragraphs with corresponding alignments

and variant indices. Indeed, if the distributions of the offsets and indices for all k texts are similar, then it might be worthy using two common codes, H_2^T and H_3^T, for all offsets and indices, respectively. As opposed to the indices, the offset values also include negative integers. Therefore, their distributions are significantly different, which makes it preferable to use two distinct codes.

One of the important properties of both our bilingual and multilingual schemes is that sections are compressed independently of other sections. The use of Huffman coding rather than Bzip, LZ or any other encoding with inter-section dependencies for representing both source and target texts enables storing them in blocks of any size without any decrease in space efficiency. Nevertheless, adaptive encodings are also applicable in our case, but the compression savings might be hurt in case of partition into small blocks.

The decompression algorithm is straightforward. Note that it needs only the dictionary files, as all relevant information included in the other files is encoded within the compressed text itself.

Figures 1 and 2 give an example of the algorithm's input and output, correspondingly. The index n in the first columns denotes the ordinal token number. The second column of Figure 1 lists the tokens of an English paragraph, taken from the experimental multilingual corpus (see Section 3). The third and fifth columns are the French and German parallels of that paragraph, respectively. The fourth and sixth columns show the connections suggested by the English-French and English-German alignments. As an example, the connection $\langle 6, 2, 6, 2\rangle$ in A_{S,T^1} indicates the fact that the English sequence `common position`, beginning at token number 6 and consisting of 2 tokens, is aligned with the French sequence `position commune`, also beginning at token number 6 (but of the French paragraph) and consisting of 2 tokens. Note that the English-German alignment (A_{S,T^2}) also relates to the same two English tokens; nevertheless, it aligns each of them separately with its German counterpart. This difference merely originates from the way in which the alignment algorithm computes the probabilities

n	Annotated S (English)	Compressed T^1 (French)	Compressed T^2 (German)
1	For	Aux	Im
2	the #AL1#	#REF#, 0, ϵ, 0	Sinne
3	purpose,ϵ	de	dieses
4	of	la	#REF#, 1, 0, 0
5	this #AL1# #AL2#	présente	#REF#, 1, ϵ, 1
6	common,ϵ #AL1#	#REF#, 1, ϵ, 0, 0	bedeutet
7	position,ϵ		der
8	,	,	Ausdruck
9	the	on	:
10	following	entend	
11	definitions	par	
12	shall	:	
13	apply		
14	:		

Fig. 2. Compression of French and German texts using their English parallel

of candidate connections, which also takes into account some offset and length probabilities.

The second column of Figure 2 displays the English text, playing the role of the source text, with annotation of the aligned sequences as well as their lemmatization indices. In this specific case, ϵ's are output as lemmatization indices, because all 3 aligned tokens have only one possible lemma. The codeword `#AL1#` indicates that the next token is the beginning of an aligned sequence of length 1. Likewise, `#AL2#` marks the beginning of an aligned sequence of length 2. Notice that token number 6 is the beginning of two distinct sequences. The former, of length 1, comes from A_{S,T^2}, whereas the latter, of length 2, originates in A_{S,T^1}. At the same time, token No. 7 is also a 1-token sequence, also coming from A_{S,T^2}.

Finally, the third and fourth columns present the compressed forms of the French and German paragraphs, respectively. The two aligned sequences in each target text have been replaced with suitable references. For instance, the French sequence `position commune` has been substituted with the reference $1, \epsilon, 0, 0$. The offset 1 results from the fact that the previous replacement relates to source sequence number 0, namely, `purpose`. Thus, *expected* for the current connection is 1, which refers to the sequence `common`, rather than `common position`, enumerated as number 2. The offset, therefore, is ord $(6, 2)$ $-$ *expected* $= 2 - 1 = 1$.

The sequence `common position` has a single translation in G_{S,T^1}, that is, `position commune`[1]. As a result, no translation index is written to the output

[1] Note that the French lemmatizer has given the feminine form `commune` as the lemma of `commune`. That is because it referred to the feminine noun `commune` (community) rather than to the feminine form of the adjective `commun` (common). However, this has no significance for our algorithm.

stream (expressed in the figure by ϵ). When the decoder reaches the discussed reference, it will first identify the implied English sequence using the offset value. Then, after lemmatizing its words using L_S, it will look up the sequence in G_{S,T^1}. At that point, it will find out that the proposed translation is unique, and therefore will not read an index at that stage. The next step will be seeking the word `position` in V_{T^1}, finding there several options such as `position`, `positions`, `POSITION` etc.. Now the decoder must read the next index, i.e. 0, in order to choose the right variant. The same process will be performed for the lemma `commune`, which also has a few possible variants.

The next section exhibits our empirical results on a real-life corpus and discusses their consequences.

3 Results and Analysis

In order to assess our algorithms, we extracted a 6-language corpus from the *EUR-Lex* website [7], which holds the European Union's legislative publications of the last few years in all EU members' languages (currently 23). Our subset was formed of all texts published between January 1st and May 31st 2005 in the following languages: German (de = Deutsch), English (en), Spanish (es = Español), French (fr), Italian (it) and Portuguese (pt).

Table 1 displays the size of each part of the corpus in MBs and in million words. Notice that the Portuguese text is significantly smaller than the others. That is because a very large document, constituting around 11.5% of each of the other texts, had no Portuguese version. It should also be noted that a few other small pieces are missing in most of the texts since they have not been translated or due to technical problems. This situation simulates a real-life corpus, where not all contents exist in all languages. This fact makes the current results even more relevant.

The lemmatization of the texts was done using the *Tree Tagger* [13], a language-independent part-of-speech tagger and lemmatizer, currently adapted to several European languages. As the paragraphs of each document in the various languages were not precisely aligned with each other, we applied a simple adjustment of the *DKvec* algorithm [8] to all 30 possible pairs of texts in order to obtain a better paragraph-level alignment. Finally, the bilingual glossaries and detailed alignments were automatically generated by an extension of the *word_ align* algorithm [6] to multi-word sequences. The average length of an aligned target sequence for all 30 alignments was around 1.7 words per sequence. The rate of aligned target words was within the range of 28–40%, depending on the available level of monolingual pre-processing, the relative nature of the languages of

Table 1. Full sizes of the parallel texts

Unit	de	en	es	fr	it	pt
MB	27.37	25.98	28.03	28.56	27.55	24.28
MW	4.46	4.79	5.10	5.20	4.93	4.41

Table 2. Averages for monolingual methods

	Method				
	BZip2		GZip		HWord
Excluding	1file	split	1file	split	
de	17.5	24.5	22.6	27.3	23.1
en	17.5	24.6	22.7	27.5	22.7
es	17.6	24.7	22.8	27.6	22.9
fr	17.7	24.7	22.8	27.6	22.8
it	17.6	24.7	22.8	27.6	22.8
pt	17.4	24.3	22.6	27.3	22.8

Table 3. Results for bilingual algorithm

	Target												
	de		en		es		fr		it		pt		Avg
Src	MB	%	MB	%	MB	%	MB	%	MB	%	MB	%	(%)
de			4.92	18.9	5.27	18.8	5.41	18.9	5.26	19.1	4.63	19.1	19.0
en	4.70	17.2			4.90	17.5	5.00	17.5	4.92	17.8	4.34	17.9	17.6
es	4.85	17.7	4.71	18.1			4.95	17.3	4.84	17.6	4.22	17.4	17.6
fr	4.77	17.4	4.60	17.7	4.75	16.9			4.75	17.2	4.21	17.3	17.3
it	4.77	17.4	4.67	18.0	4.79	17.1	4.88	17.1			4.23	17.4	17.4
pt	4.93	18.0	4.88	18.8	4.97	17.7	5.14	18.0	5.01	18.2			18.1
Avg		17.6		18.3		17.6		17.8		18.0		17.8	17.8

the aligned texts and some other text-specific factors (average rates presented in Table 7).

As established above, the basic idea of alignment-based compression is to exploit parallelism to achieve better results compared with general-purpose, monolingual methods. Therefore, it is necessary to apply several such methods on the test corpus in order to examine the extent of improvement obtained by the various multilingual schemes.

We define the *compression rate* as the fraction, given in percent, of the size of the compressed file divided by the original size. Table 2 details the average compression rates yielded by 3 principal methods—Bzip2, Gzip and HuffWord—for all 6 possible combinations of 5 languages. The first column, titled "Excluding", indicates the identity of the text *not* taken into account. This is needed for a fair comparison with the performances of our algorithms on each combination of target languages. BZip2 and GZip were first applied to the target texts, considering each as a single file (1file), then each file was split into its 3076 documents, and each fragment was compressed on its own (split). The increase in the file sizes can be immediately recognized. Note that HWord gives identical results for both settings.

Table 3 presents the results achieved by the bilingual algorithm for each source-target pair of texts along with the average compression rates for each source and target texts and the aggregate average.

Table 4. Source annotation overheads

	Text	Text + 1 alignment						Text + 5 alignments					
	Size	Size	Al.		Lem.		Tot.	Size	Al.		Lem.		Tot.
S.	MB	MB	MB	%	KB	%	%	MB	MB	%	KB	%	%
de	5.91	6.43	0.52	8.81	2.83	0.05	8.86	6.77	0.85	14.46	4.53	0.07	14.54
en	6.09	6.71	0.62	10.27	8.95	0.14	10.41	7.19	1.10	18.09	14.46	0.23	18.32
es	6.35	7.00	0.65	10.25	24.29	0.37	10.62	7.52	1.17	18.38	40.47	0.62	19.00
fr	6.56	7.22	0.67	10.15	35.52	0.53	10.68	7.79	1.23	18.82	57.92	0.86	19.68
it	6.39	7.04	0.65	10.19	30.12	0.46	10.66	7.56	1.18	18.46	48.10	0.74	19.20
pt	5.64	6.23	0.59	10.38	23.14	0.40	10.78	6.70	1.06	18.77	35.29	0.61	19.38

Table 5. Additional overhead as function of k (English text as source)

k	Text + k alignments	Diff (MB)	Add ovrhd (%)	ratio
0	6.09			
1	6.67	0.58	9.55	
2	6.92	0.26	4.21	0.44
3	7.06	0.13	2.17	0.52
4	7.15	0.09	1.51	0.70
5	7.20	0.05	0.87	0.57

It should be noted that the numbers for the bilingual and multilingual algorithms do not include the sizes of the auxiliary files, since in the scenario of a large multilingual Information Retrieval system, dictionaries and glossaries are needed anyway and are not stored exclusively as an aid for compression. However, even if those sizes are to be considered, it should be kept in mind that, according to Heaps' Law [10], the size of a dictionary for a text of size n is expected to be αn^{β}, where $0.4 \leq \beta \leq 0.6$. The total size of the auxiliary dictionaries for the current evaluation corpus, compressed using Bzip2 (rather than a dictionary-oriented compression scheme), is about 9% of the decompressed text. Should a 20GB corpus be compressed, the corresponding dictionaries would comprise less than 1% of the original text. Obviously, specific dictionary compression can further decrease that rate.

As stated in Section 2, the multilingual algorithm uses an annotated source text. Obviously, this annotation is not for free. Table 4 displays the overheads paid for the markup of aligned source sequences ("Al.") as well as for lemmatization indices ("Lem."). As already remarked, we used one Huffman code for text words and alignment marks (for sequence lengths 1–7), and another Huffman code for the lemma indices. Therefore, the overhead of the alignment data is calculated by subtracting the size of the HuffWord file encoding the pure text from that of the file encoding the annotated text, whereas the lemmatization overhead is simply the cost of storing the series of lemma indices using a distinct Huffman code. For the sake of briefness, we present only the average overheads in the case of a single target text. This is acceptable because the differences from

Table 6. Results for multilingual algorithm (with $k = 5$)

Src.	Oh. (MB)	Target						Inc. Oh. (%)	Exc. Oh. (%)
		de	en	es	fr	it	pt		
de	0.86		4.58	4.92	5.05	4.91	4.32	18.3	17.7
en	1.11	4.38		4.50	4.60	4.53	3.99	17.0	16.2
es	1.21	4.51	4.30		4.48	4.39	3.83	17.0	16.1
fr	1.29	4.42	4.19	4.28		4.29	3.79	16.7	15.7
it	1.23	4.43	4.27	4.34	4.42		3.83	16.8	15.9
pt	1.09	4.62	4.51	4.56	4.71	4.60		17.5	16.7
Avg.								17.2	16.4

Table 7. Result summary

Src	Alignment Coverage	Bilingual	Multilingual			BZip2		GZip		HWord
			$k = 1$	$k = 5$	$k \to \infty$	1file	split	1file	split	
de	29.0	19.0	19.4	18.3	17.7	17.5	24.5	22.6	27.3	23.1
en	35.9	17.6	17.7	17.0	16.2	17.5	24.6	22.7	27.5	22.7
es	36.6	17.6	17.6	17.0	16.1	17.6	24.7	22.8	27.6	22.9
fr	38.0	17.3	17.2	16.7	15.7	17.7	24.7	22.8	27.6	22.8
it	37.3	17.4	17.4	16.8	15.9	17.6	24.7	22.8	27.6	22.8
pt	32.8	18.1	18.2	17.5	16.7	17.4	24.3	22.6	27.3	22.8
Avg	34.9	17.8	17.9	17.2	16.4	17.5	24.6	22.7	27.5	22.8

the average values are very small. In addition, this impreciseness has no effect on the average compression rates.

A glance taken at the overhead table reveals that the ratio between the annotation cost for five targets ($k = 5$) and for a single target ($k = 1$) is less than 2. That is, the additional overhead for another four targets is smaller than the cost paid for the first target. Table 5 details the growing sizes of the text+alignment HuffWord files for the English text as source, when each time marks for an additional alignment are incorporated into the text. The ratio between the additional overheads for the kth and $(k-1)$th alignments is around 0.6, which leads to the thought that if some more targets were added, the additional overhead would quickly converge towards 0. Consequently, the annotation overhead in the case of large enough k's may be deemed a constant independent of k, which permits ignoring it and taking into account only the sizes of the compressed targets .

Table 6 exhibits the performances of the multilingual algorithm for $k = 5$.

The compression rates were computed by accumulating the overhead and the sizes of the five compressed targets and then dividing the result by the sum of the sizes of the five decompressed targets. The results for $k = 1$ were slightly worse than those achieved by the bilingual algorithm (averages given in Table 7). This had been quite expected in light of the high overhead paid for a stand-alone alignment.

The rightmost column of Table 6 shows the compression rate when overheads are excluded. For $k = 5$, the average overhead constitutes ca. 0.8% of the entire decompressed corpus. As Table 5 hints, overhead is unlikely to grow significantly

even for a sixth target, so we may already deem it maximal. Hence, for $k = 23$, the current number of languages in the European Union, the overhead's relative part may drop below 0.2%.

Finally, Table 7 puts side-by-side the average alignment coverage rates and the compression rates achieved by the monolingual, bilingual and multilingual algorithms. For the monolingual methods, the term "Source" refers to the text *not* taken into average account (equivalent to "Excluding" in Table 2). The column labeled "$k \to \infty$" includes the same data presented in the "Exc. Oh." column of Table 6. Indeed, the compression rate which could be obtained for a large number of targets with compressibility similar to that of the current test texts converges to the rate computed excluding the source annotation overhead. That is because for large k's, this overhead converges towards a relatively small constant and may therefore be neglected.

A close observation into the results would recognize a tight correlation between alignment coverage and compression rates. This is, of course, expected, because each aligned target sequence is replaced with a reference, which is shorter, in average, than the corresponding HuffWord encoding. Hence, the higher the coverage rate, the better the compression. It may be assumed that if the German text could be stemmed before being aligned, we would get even better results. In general, if the alignment algorithm yielded denser outputs, the results could improve significantly.

BZIP2 does not perform as well for small blocks as it does for large blocks, as opposed to Huffman coding, which is indifferent to block size. Therefore, in cases where extraction of relatively small pieces is desired, even the bilingual scheme with Huffman coding would be preferable over regular BZIP2, not to mention the other 2 monolingual methods. Results also show that it is unworthy using the multilingual scheme rather than the bilingual one for small k. However, for large enough k, the multilingual algorithm is advantageous even if each target is to be stored in a bulk, rather than split into pieces. In that situation, of course, the encoding method of the compression procedure's output may be changed from HWORD to BZIP2, which is expected to yield a further improved compression.

4 Conclusion

The current work suggests the first specific methods for compressing multilingual parallel texts, based on text alignment. The algorithm has been tested on a real-life multilingual corpus and achieved significant improvements over general-purpose methods.

A prominent advantage of the compression scheme is its static nature, which enables applying it to blocks of any size without changing compression efficiency. This property is particularly important for Information Retrieval systems, where users are frequently interested in relatively small pieces of texts. Compressing each small piece per se permits transferring and deciphering the desired piece only, thereby saving a lot of expensive communication and processing time.

Acknowledgment

The first author would like to express his deep gratitude for Drs. Mordecai & Monique Katz and the Mozes S. Schupf Fellowship Program for their support for his work on this research.

References

1. Adiego, J., Brisaboa, N.R., Martínez-Prieto, M.A., Sánchez-Martínez, F.: A two-level structure for compressing aligned bitexts. In: Karlgren, J., Tarhio, J., Hyyrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 114–121. Springer, Heidelberg (2009)
2. Ahrenberg, L., Andersson, M., Merkel, M.: A knowledge-lite approach to word alignment. In: Véronis, J. (ed.) Parallel Text Processing, pp. 97–116. Kluwer Academic Publishers, Dordrecht (2000)
3. Brown, P.F., Della Pietra, S., Della Pietra, V.J., Mercer, R.L.: The mathematics of statistical machine translation: parameter estimation. Computational Linguistics 19(2), 263–311 (1993)
4. Conley, E.S., Klein, S.T.: Using alignment for multilingual text compression. Int. J. Found. Comput. Sci. 19(1), 89–101 (2008)
5. Conley, E.S., Klein, S.T.: Compression of multilingual aligned texts. In: DCC, p. 442. IEEE Computer Society, Los Alamitos (2006)
6. Dagan, I., Church, K.W., Gale, W.A.: Robust bilingual word alignment for machine-aided translation. In: Proc. of the Workshop on Very Large Corpora: Academic and Industrial Perspectives, Columbus, Ohio, pp. 1–8 (1993)
7. EUR-Lex, http://eur-lex.europa.eu/
8. Fung, P., McKeown, K.: Aligning noisy parallel corpora across language groups: Word pair feature matching by dynamic time warping. In: Proceedings of the First Conference of the Association for Machine Translation in the Americas, pp. 81–88 (1994)
9. Gaussier, É., Hull, D., Aït-Mokhtar, S.: Term alignment in use: Machine-aided human translation. In: Véronis, J. (ed.) Parallel Text Processing, pp. 253–274. Kluwer Academic Publishers, Dordrecht (2000)
10. Heaps, J.: Information Retrieval: Computational and Theoretical Aspects. Academic Press, Inc., New York (1978)
11. Martínez-Prieto, M.A., Adiego, J., Sánchez-Martínez, F., de la Fuente, P., Carrasco, R.C.: On the use of word alignments to enhance bitext compression. In: Storer, J.A., Marcellin, M.W. (eds.) DCC, p. 459. IEEE Computer Society, Los Alamitos (2009)
12. Nevill, C., Bell, T.: Compression of parallel texts. Information Processing & Management 28, 781–793 (1992)
13. Schmid, H.: TreeTagger – a language-independent part-of-speech tagger. Web address, http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

Singular Artin Monoids of Finite Coxeter Type Are Automatic*

Ruth Corran[1], Michael Hoffmann[2], Dietrich Kuske[3], and Richard M. Thomas[2]

[1] American University of Paris, Paris, France
[2] Dept of Computer Science, University of Leicester, Leicester, UK
[3] Technische Universität Ilmenau, Ilmenau, Germany

Abstract. We consider the positive singular and the singular Artin monoids of finite type. These have been the subject of a great deal of recent research and the main purpose of this paper is to prove that these monoids are automatic. In order to do this we establish a new criterion for proving monoids automatic that may be of independent interest.

1 Introduction

The concept of an automatic group was introduced in [4,14] in order to describe a large class of naturally occurring groups with an easily solvable word problem. This was then extended to automatic monoids; see [8,18,20] for example. This notion (see Definition 1) uses the concept of a *transducer*, a finite automaton with two input tapes and two one-way heads. If these two heads are required to move synchronously, we speak of a *synchronous transducer*. As a transducer has two input tapes, its language can be described as a binary relation on the set of strings. Such a relation is said to be (*synchronously*) *rational* if it is the language of a (synchronous) transducer.

When constructing a synchronous transducer, it can be simpler to first build an asynchronous machine which then is transformed into a synchronous one. When considering automatic monoids and groups, one technique for this transformation has mainly been used: if the positions of the two asynchronously moving heads differ uniformly by at most k, then an equivalent synchronous automaton exists; this technique requires one to show that along any successful computation, *at any given time*, the difference does not exceed k (see [17]). Using a result of Frougny and Sakarovitch [15], this can be relaxed to showing that, in any successful run, the difference is at most k in the *final* configuration (see Proposition 2 below).

* This work was begun while the first author was the recipient of a grant from the EPSRC (UK) and continued while she was a Marie Curie Postdoctoral Researcher (EU); the support of these two funding bodies is gratefully acknowledged. The results of this paper were obtained when the third author was affiliated with the University of Leicester and, later, the Universität Leipzig. The second and fourth authors would like to thank Chen-Hui Chiu and Hilary Craig for all their help and encouragement.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 250–261, 2011.

In this paper we apply this new approach to singular Artin monoids of finite type (Theorems 1 and 2). The Artin groups of finite type all share the remarkable properties of the braid group B_n and B_n may be studied very fruitfully in its guise as such a group. The braid group has applications in diverse areas such as combinatorial group theory, representation theory, trace monoids and low-dimensional topology; it has even been the focus of interest as a prospect for a public-key authentication system based on a non-Abelian group [23]. Shortly after B_n was introduced by Artin, a presentation by generators and relations with remarkable properties was obtained which led to the definition of the more general Artin monoids; this motivates the currently developing theory of Garside structures.

The singular braid monoid SB_n was introduced in [3,6] to study Vassiliev knot invariants; the idea is that the strings of a braid can have intersections. In the same way that the braid group may be considered as an example of an Artin group, the definition of singular braid monoid may be extended to arbitrary Artin type and a *singular Artin monoid* is obtained [10]. Roughly speaking, a singular Artin monoid is an Artin monoid with some additional "singular" generators. We combine part of the normal form from [9] for Artin monoids with the singular parts of the braids to give automatic structures for singular Artin monoids.

An interesting connection with the theory of trace monoids [11] is that the submonoid consisting of the purely singular braids of a singular Artin monoid defined by a graph Γ is isomorphic to the trace monoid on a graph which is the complement of Γ. Thus the singular Artin monoid is some sort of blend of an Artin group with a trace monoid, and the normal form for the automatic structure we obtain in this paper is a blend between that for the (bi)automatic structure for Artin groups of finite type and the Foata normal form for traces.

The proofs are quite lengthy and combinatorial in nature and we do not give the full details in this extended abstract; instead we provide a sequence of lemmas and previously known facts that we have used to prove these theorems and we hope that these give the reader a feeling as to the overall structure of the proofs of these results.

2 Automaticity via Rational Relations

Let $(M, \cdot, 1)$ be a monoid. An *M-automaton* is a structure $\mathcal{A} = (Z, \delta, \iota, F)$ where Z is a finite set of states, $\delta \subseteq Z \times M \times Z$ is a finite transition relation with $(z, 1, z') \in \delta$ if and only if $z = z'$, $\iota \in Z$ is an initial state, and $F \subseteq Z$ is a set of accepting states; thus an M-automaton is a finite graph whose edges are labeled by elements of the monoid M. A run is a finite sequence $(z_i, m_i, z_{i+1})_{1 \leqslant i \leqslant n}$ of transitions; its label is the element $m_1 \cdot m_2 \cdots m_n$ of the monoid M. A run is successful if $z_1 = \iota$ and $z_{n+1} \in F$. The set $L(\mathcal{A})$ accepted by $\mathcal{A}$ is the set of labels of successful runs. A subset X of M is said to be *rational* if there is an M-automaton $\mathcal{A}$ with $X = L(\mathcal{A})$, i.e. if it is the behaviour of some M-automaton.

Any finite set $X \subseteq M$ is rational. If $X, Y \subseteq M$ are rational, then so are the following sets (see [13] for example):

- $X \cup Y$ and $X \cdot Y = \{x \cdot y : x \in X, y \in Y\}$,
- $\langle X \rangle = \{x_1 \cdot x_2 \cdots x_n : x_i \in X\}$ (some authors use X^* in place of $\langle X \rangle$).

Conversely, any rational set can be constructed from finite sets using the operations $\cup$, $\cdot$, and $\langle\,\rangle$.

If $M = \Gamma^*$ is a finitely generated free monoid then the rational subsets of M are also known as *regular languages*; in this case $M \setminus X$ and $X \cap Y$ are rational whenever X and Y are rational (but this does not hold for general monoids M).

If $M = \Gamma^* \times \Delta^*$ is the direct product of two finitely generated free monoids then M-automata are also known as *transducers* and rational subsets of M as *rational relations* or *transductions* (see [5] for an extensive treatment of rational relations). To emphasize that the behaviour of a transducer $\mathcal{A}$ is a relation we will write $R(\mathcal{A})$ instead of $L(\mathcal{A})$. For example, if $\Gamma = \Delta = \{a\}$, then the relation

$$\{(a^n, a^{2n}) : n \in \mathbb{N}\} = \langle\{(a, aa)\}\rangle$$

can be realized by an automaton with just one state and a loop labeled (a, aa).

Another way to consider relations accepted by a finite state device is to first transform the relation into a language and then check whether this language is regular. Let $\perp \notin \Gamma$ be a symbol and $\Gamma(2, \perp) = (\Gamma \cup \{\perp\})^2 \setminus \{(\perp, \perp)\}$. We define the *convolution* $\otimes : \Gamma^* \times \Gamma^* \to \Gamma(2, \perp)^*$ by:

$$\varepsilon \otimes \varepsilon = \varepsilon \qquad a \otimes \varepsilon = (a, \perp) \qquad \varepsilon \otimes b = (\perp, b) \qquad av \otimes bw = (a, b)(v \otimes w)$$

for $a, b \in \Gamma$ and $v, w \in \Gamma^*$. If $R \subseteq \Gamma^* \times \Gamma^*$ let $R^{\otimes} = \{v \otimes w : (v, w) \in R\}$ denote the convolution of R. Note that $R^{\otimes}$ is a language over the alphabet $\Gamma(2, \perp)$. Define a homomorphism $\eta : \Gamma(2, \perp)^* \to \Gamma^* \times \Gamma^*$ by:

$$\eta(a, b) = (a, b), \qquad \eta(a, \perp) = (a, \varepsilon), \qquad \eta(\perp, b) = (\varepsilon, b)$$

for $a, b \in \Gamma$. Since $R = \eta(R^{\otimes})$ and rational sets are closed under homomorphic images [13], we have that, if $R^{\otimes}$ is rational, then R is rational. The converse is not true however: for example, if $R = \langle(a, aa)\rangle$, then R is rational but $R^{\otimes} = \{(a, a)^n(\perp, a)^n : n \in \mathbb{N}\}$ is not. Given the following result the reason for this failure is that the length difference of u and v is unbounded for $(u, v) \in R$:

Proposition 1 (Corollary 2.5 of [15]). *If $\mathcal{A} = (Z, \delta, \iota, F)$ is a transducer and $k \in \mathbb{N}$ is such that $||u| - |v|| \leqslant k$ for $(u, v) \in R(\mathcal{A})$ then $R(\mathcal{A})^{\otimes}$ is regular.*

Given this result, we say that a rational relation R is *difference bounded* if there is a constant k such that $||u| - |v|| \leqslant k$ for all $(u, v) \in R$.

Let M be a monoid, Γ a finite set, $\theta : \Gamma^* \to M$ an epimorphism, and $L \subseteq \Gamma^*$. Then we define:

$$L(\varepsilon) = \{(u, v) \in L^2 : \theta(u) = \theta(v)\}; \qquad L_= = L(\varepsilon)^{\otimes};$$
$$L(a) = \{(u, v) \in L^2 : \theta(ua) = \theta(v)\}; \qquad L_a = L(a)^{\otimes}$$

for $a \in \Gamma$. We have the following definition of an automatic structure for M:

Definition 1. *An* automatic structure *for M is a triple (Γ, θ, L) where:*

1. *Γ is a finite set, $\theta : \Gamma^* \to M$ is an epimorphism and $L \subseteq \Gamma^*$ is regular;*
2. *$\theta(L) = M$; and*
3. *$L_=$ and L_a are regular for every $a \in \Gamma$.*

A monoid is said to be automatic *if it has an automatic structure.*

It is not hard to see that the regularity of L follows from the remaining conditions; in the automaton for the language L_a, we replace labels of the form (a, b) by a and those of the form $(\perp, b)$ by ε and drop the remaining transitions. We can assume [8] that θ maps L injectively onto M. In this case, the regularity of $L_=$ follows immediately from the regularity of L: replace transition labels of the form a by (a, a). The resulting automaton, considered over $\Gamma(2, \perp)$, accepts $\{u \otimes u \ : \ u \in L\}$ which equals $L_=$ by the injectivity of $\theta{\restriction_L}$. Hence the main task in showing the automaticity of a monoid is the construction of synchronous transducers whose language is $L(a)$. Since these transducers describe the multiplication by generators, they are called *multiplier automata*.

Given Proposition 1 one can derive an alternative characterization of automatic monoids as follows:

Proposition 2. *Let M be a monoid.*

1. *If there exists a finite set Γ, an epimorphism $\theta : \Gamma^* \to M$ and a regular language $L \subseteq \Gamma^*$ such that $L(a)$ is a rational relation and difference bounded for any $a \in \Gamma$, then (Γ, θ, L) is an automatic structure for M.*
2. *If M has an automatic structure (Γ, θ, L) with $\theta{\restriction_L}$ injective and if the set $\{x \in M \ : \ x\theta(a) = y\}$ is finite for any $y \in M$ and $a \in \Gamma$ then $L(a)$ is difference bounded for any $a \in \Gamma$.*

Note that the finiteness assumption is necessary in the second part. For example, if the automatic monoid M contains a zero element, then the relations L_a cannot be difference bounded. If the monoid M has a length function, then the finiteness assumption is always satisfied; this will be the case in our considerations.

We finish this section by introducing some notational conventions. The set of natural numbers $\{1, 2, \ldots, n\}$ is denoted by $[n]$. For a monoid $(M, \cdot, 1)$, we write $x \preceq y$ if and only if x is a left divisor of y, i.e. if there exists $z \in M$ with $x \cdot z = y$. Since the monoids under consideration will be cancellative and will have no nontrivial decomposition of the unit element, $\preceq$ will be a partial order.

3 The Results

A *Coxeter graph* is a partially edge-labelled (undirected) graph with vertex set $[n]$ with labels coming from $\{3, 4, \ldots\} \cup \{\infty\}$. For a fixed Coxeter graph Γ, we define m_{ij} to be the label of the edge between vertices i and j if it exists, and 2 otherwise; in particular, $m_{ij} = m_{ji}$ holds. The Coxeter graphs of *finite type* finite unions of graps from Fig. 1 (see below for an explanation of this terminology).

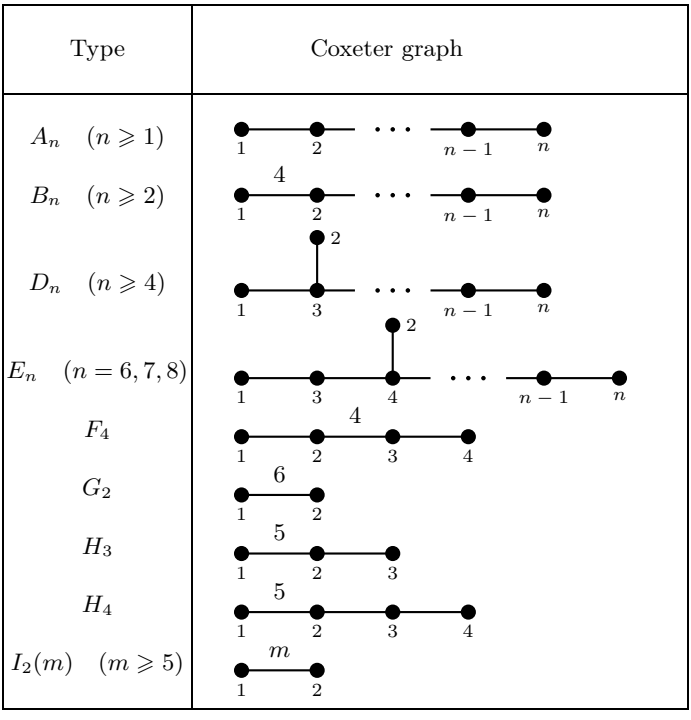


Fig. 1. Connected Coxeter graphs of finite type. Unlabelled edges have value 3.

For a nonempty word w and a natural number n, let $\langle w \rangle^n$ be the prefix of length n of the word w^n.

The *positive singular Artin monoid of type* Γ, denoted by M_Γ, will be defined via a presentation. The generators are $\Sigma \cup T$ where

$$\Sigma = \{\sigma_1, \ldots, \sigma_n\} \quad \text{and} \quad T = \{\tau_1, \ldots, \tau_n\},$$

that is, one σ- and one τ-type generator for each vertex of Γ. The relations are constructed via the edge information. For $m_{ij} \neq \infty$, we have the relations:

$$\langle \sigma_i \sigma_j \rangle^{m_{ij}} = \langle \sigma_j \sigma_i \rangle^{m_{ij}} \tag{R_1}$$

$$\tau_i \langle \sigma_j \sigma_i \rangle^{m_{ij}-1} = \langle \sigma_j \sigma_i \rangle^{m_{ij}-1} \tau_{j(m_{ij} \bmod 2)+i((m_{ij}+1) \bmod 2)} \tag{R_2}$$

$$\tau_i \tau_j = \tau_j \tau_i \quad \text{if } m_{ij} = 2 \tag{R_3}$$

$$\tau_i \sigma_i = \sigma_i \tau_i \text{ for } i \in [n] \tag{R_4}$$

The *positive singular Artin monoid* M_Γ *of type* Γ is given by the presentation

$$M_\Gamma = \operatorname{Mon} \langle \Sigma \cup T \,:\, R_1 \cup R_2 \cup R_3 \cup R_4 \rangle .$$

Our aim is to show that M_Γ is automatic and we prove:

Theorem 1. *Any positive singular Artin monoid of finite type is automatic.*

Birman [6] and Baez [3] introduced such a monoid in the special case where Γ is a path and all labels are 3 (type A in Figure 1); the general case was introduced in [10]. Birman's conjecture (the "desingularisation map" is an embedding of M_Γ into the group algebra of the braid group) was proved for the monoid of singular braids [21] and for the singular Artin monoids of *right angled type* [16] and of type $I_2(m)$ [12]. Antony [1] provides evidence that Birman's conjecture might hold in the general case.

Having considered the positive singular Artin monoids of finite type we turn to a monoid into which they embed, namely the singular Artin monoid.

Let Γ be any Coxeter graph with associated positive singular Artin monoid M_Γ. Define the *singular Artin monoid* of type Γ, denoted by M_Γ^A, to be the monoid defined by the presentation with generators $\Sigma \cup T \cup \Sigma^{-1}$ (the last being the set of formal inverses of Σ), and relations R_1, R_2, R_3 and R_4 (as given above) together with the additional relations

$$\sigma_i \sigma_i^{-1} = \sigma_i^{-1} \sigma_i = 1 \qquad \text{for each } i \ .$$

Our aim is to show that M_Γ^A is also automatic and we prove:

Theorem 2. *Any singular Artin monoid of finite type is automatic.*

4 The Positive Singular Artin Monoid M_Γ

We now collect together some results about the monoid $M = M_\Gamma$ which we will need in this paper. The following three results are proved in [10].

(0.1) The monoid M is left and right cancellative.

(0.2) Any subset $X \subseteq M$ has a least common right (respectively left) multiple precisely when it has a common right (respectively left) multiple. When this is the case, the lcm is unique. Let X be the set of common left divisors of elements x and y. Then X has a common right multiple and therefore a least common multiple that we denote by $\gcd(x, y)$ since it equals the greatest common left divisor of x and y. In particular, any two elements of M have a greatest common left divisor.

(0.3) For $i \neq j$, τ_i and τ_j have a common multiple if and only if $\tau_i \tau_j = \tau_j \tau_i$, in which case this is the least common multiple.

Let Rev be the map on words over $\Sigma \cup T$ which reverses the word. Since $u = v$ is a defining relation precisely when $\mathrm{Rev}(u) = \mathrm{Rev}(v)$ is a defining relation, Rev extends to an anti-endomorphism of M_Γ (i.e. $\mathrm{Rev}(xy) = \mathrm{Rev}(y)\,\mathrm{Rev}(x)$ for any $x, y \in M$) and it is easy to see that:

(0.4) The map Rev is an anti-automorphism of M_Γ of order two.

Observe that all the relations involving a single element of T on each side of the equation are all of the form $\tau_i w = w \tau_j$ where w is a word over Σ and

j may or may not be the same as i. Furthermore, the relations involving more than one element of T on each side involve *only* letters from T; thus we have a homomorphism $\nu : M_\Gamma \to M_\Gamma$ defined on generators by mapping $\sigma_i \mapsto \sigma_i$ and $\tau_i \mapsto 1$. The image under ν is the submonoid $S = S_\Gamma$ generated by Σ called the *positive Artin monoid*; it has presentation given by generators Σ and relations R_1. If $x\tau_i y = z\tau_j w$, where x, y, z, w are elements of S, then, by considering the image under ν, we have that $xy = zw$.

(0.5) If $x\tau_i y = z\tau_j w$ where $x, y, z, w \in S$, then $xy = zw$.

The reflection group of type Γ is the quotient of the Artin monoid S obtained by imposing order 2 on the generators. The Coxeter graph Γ is of finite type (i.e., is a finite union of graphs from Fig. 1) if and only if the set Σ as a common multiple in S (see [7]), or, equivalently (given that S embeds in M), has a common multiple in M. *From now on we will restrict ourselves to the case where Γ is of finite type.*

(0.6) [7] For every finite $X \subseteq S$, the lcm of X always exists. Thus, since S is a submonoid of M, any finite subset X of M where each $x \in X$ is representable by a word over Σ has a least common multiple that is itself representable as a word over Σ.

(0.7) [10] For $x \in S$ and $\tau \in T$, $\mathrm{lcm}(\tau, x)$ always exists, and is of the form $\tau xa = xa\tau'$ for some $a \in S$ and $\tau' \in T$.

Since M_Γ has unique least common multiples whenever common multiples exist, we have a unique least common multiple Δ of Σ. Let

$$Q := \{q \in M \setminus \{1\} \,:\, qp = \Delta \text{ for some } p\}.$$

By preservation of the number of τ's (all relations have same number of τ's on both sides, so all words representing any given element of M have the same number of τ's), all elements of Q are elements of S; in fact (see [7]) they are precisely the non-trivial elements of S which are not expressible in the form $uaav$ for any generator $a \in \Sigma \cup T$; given this, they are said to be *square-free elements.*

(0.8) [7] If $qxy = \Delta$ then each of q, x and y is in Q.

(0.9) [10] There is an automorphism $\overline{\cdot}$ of M_Γ defined by $w\Delta = \Delta\overline{w}$, which, in particular, defines a permutation on T, and a permutation on Σ. This automorphism is either trivial or of order 2, depending on the type of Γ.

By (0.7), $\mathrm{lcm}(\tau, q)$ exists for any $\tau \in T$ and $q \in S$ and is of the form τqa. The following two results give some more information on the element $a \in S$ provided that q is square-free. We can use (0.1), (0.5) and (0.8) to prove:

Lemma 1. *If $q \in Q$ and $\tau \in T$, then the least common multiple of q and τ exists and is of the form $qx\tau_j = \tau qx$ for some $x \in S$ and $\tau_j \in T$, and qx is square free.*

We can then use Lemma 1 to deduce:

Corollary 1. *Let $\tau \in T$ and $z \in M$ with $\tau \preceq z$ and let $q = \gcd(z, \Delta)$. Then $\tau q = q\tau'$ for some $\tau' \in T$ and $\tau q = \mathrm{lcm}(q, \tau)$.*

The restriction of the following result to elements $u, v \in S$ follows from Proposition 2.1 of [19]. As we are only interested in singular Artin monoids of finite type, one could produce an alternative proof which is a little simpler.

Lemma 2. *If $u, v \in M$ then $\gcd(uv, \Delta) = \gcd(u \gcd(v, \Delta), \Delta)$.*

Let $\mathcal{T} \subseteq M$ be the set of elements $\mathrm{lcm}(X)$ for all nonempty subsets $X \subseteq T$ having a common multiple in M. For $x \in M$, let

$$T(x) = \mathrm{lcm}\{t \in T \,:\, t \preceq x\}.$$

For $x \in M \setminus \{1\}$, let

$$\alpha(x) = \begin{cases} \gcd(x, \Delta) & \text{if } \gcd(x, \Delta) \neq 1 \\ T(x) & \text{otherwise.} \end{cases}$$

Observe that $\alpha(x) \in \mathcal{T}$ is equivalent to saying that $\Sigma \not\preceq x$ (which is shorthand for "no element of Σ left divides x"). We can then use Lemma 2 to prove:

Lemma 3. *If $x \in M$ and $p \in Q \cup \mathcal{T}$ then the pair $(\alpha(x), T(x))$ and the element p determine $\alpha(px)$.*

From Corollary 1 one can show:

Lemma 4. *If $p \in Q \cup \mathcal{T}$ and $x \in M$ with $\alpha(px) = p$ then $T(px) = T(pT(x))$.*

Suppose $\tau_i u \tau_j w = w_1 \tau_\ell \tau_k w_2 = w_1 \tau_k \tau_\ell w_2$ for some elements u, w, w_1, w_2 of S. Then, intuitively, the τ's in $\tau_i u \tau_j$ can commute after extension with $w \in S$. Using (0.6) and (0,7) one can prove the following result which shows that they can commute regardless of w:

Lemma 5. *Let $u' = tu\tau_j$ where $u \in S$, $t \in \mathcal{T}$, $j \in [n]$ and such that u' is neither left nor right divisible by Σ. In addition, let $w, w_1, w_2 \in S$, $s \in \mathcal{T}$ and $l \in [n]$ be such that $tu\tau_j w = w_1 s \tau_l w_2 = w_1 \tau_l s w_2$. Then $u = 1$.*

5 The Language of Normal Forms

Recall that $\mathcal{T} \subseteq M$ comprises all the elements $\mathrm{lcm}(X)$ for all nonempty subsets $X \subseteq T$ having a common multiple in M. Since $T \subseteq \mathcal{T}$ and $\Sigma \subseteq Q$, the singular Artin monoid M is generated by $Q \cup \mathcal{T}$. From now on, we will consider words over the generators $Q \cup \mathcal{T}$ as well as products in M of elements of $Q \cup \mathcal{T}$. To avoid confusion, we will use the following conventions.

Let $\varphi : (Q \cup \mathcal{T})^* \to M_\Gamma$ be the natural epimorphism. For words u and v in $(Q \cup \mathcal{T})^*$, we write $u \sim v$ if $\varphi(u) = \varphi(v)$, i.e. if u and v represent the same element of M_Γ. If we want to stress that u and v coincide letter by letter, we

will write $u \equiv v$. This eliminates any use of $u = v$ for words u and v. On the other hand, for elements $x, y \in M$, we will write $x = y$ to denote that they are equal. Since words over $S \cup \mathcal{T}$ of length 1 are actually elements of M as well, we can then write $a = b$ for $a, b \in S \cup \mathcal{T}$. Clearly, in this case, $a = b$, $a \equiv b$, and $a \sim b$ are equivalent. To reiterate, a string $q_1 q_2 \dots q_n$ of elements of $Q \cup \mathcal{T}$ is to be understood as word over $Q \cup \mathcal{T}$ and not as the monoid element represented by this sequence. This monoid element is denoted by $\varphi(q_1 q_2 \dots q_n)$ or, if $n = 2$, simply by $q_1 \cdot q_2$.

Michel (in Section 4 of [19]) defines normal forms from Q^* for the elements of S (again without the assumption of finite type). We extend his idea to the singular Artin monoid of finite type M: for any $x \in M$, we define a unique word $\mathrm{NF}(x)$ over $Q \cup \mathcal{T}$ and later (see Lemma 10) show that the set of all words obtained this way is a regular language in $(Q \cup \mathcal{T})^*$.

Let $\omega(x)$ be the unique element of M with $\alpha(x) \cdot \omega(x) = x$; then the normal form is defined inductively by

$$\mathrm{NF}(1) \equiv \varepsilon \quad \text{and} \quad \mathrm{NF}(x) \equiv \alpha(x)\,\mathrm{NF}(\omega(x)) \text{ for } x \in M \setminus \{1\}.$$

Note that this is well-defined since, as long as $x \neq 1$, we have that $\alpha(x) \not\equiv \varepsilon$, i.e. $\omega(x)$ is properly shorter than x.[1] Let $L \subseteq (Q \cup \mathcal{T})^*$ denote the set of all words $\mathrm{NF}(x)$ for $x \in M$.

The following results relate the normal forms of $x \in M$ and $x \cdot \tau_k$ for some $k \in [n]$. They will become useful later when we construct an automatic structure for M. We first use Corollary 1 to deduce:

Lemma 6. *Let $q_1 q_2 \dots q_p$ be in L, and suppose that the least common multiple of $\varphi(q_1 q_2 \dots q_p)$ and $\tau_i \in \mathcal{T}$ is $\varphi(q_1 q_2 \cdots q_p \tau_k)$ for some $k \in [n]$. Then there is a sequence of integers $i = i_0, i_1, \dots, i_p = k$ such that $\tau_{i_{\ell-1}} q_\ell \sim q_\ell \tau_{i_\ell}$ for each $\ell \in [p]$.*

We then use Lemmas 4 and 6 to prove:

Lemma 7. *Let $p_i \in Q \cup \mathcal{T}$ with $p_1 p_2 \dots p_m \in L$, $x = \varphi(p_1 p_2 \dots p_m)$ and let $k \in [n]$. Then either $\mathrm{NF}(x \cdot \tau_k) \equiv p_1 p_2 \dots p_m \tau_k$ or else there are $\ell \in [m]$ and $j \in [n]$ with $\mathrm{NF}(x \cdot \tau_k) \equiv p_1 p_2 \dots p_{\ell-1}(p_\ell \cdot \tau_j) p_{\ell+1} \dots p_m$. Furthermore, in this latter case, we have that $\mathrm{lcm}(\tau_j, \varphi(p_\ell \dots p_m)) = \varphi(p_\ell \dots p_m \tau_k)$.*

Having described the relation between the normal forms of $x \in M$ and $x \cdot \tau_k$, we now obtain similar results for the normal forms of x and $x \cdot \sigma_k$. Somewhat surprisingly, this turns out to be more involved. Using Corollary 1, we can prove:

Lemma 8. *Let $tw \in L$ with $t \in \mathcal{T}$ and $w \in Q^*$. In addition, let $q \in Q$. Then $\mathrm{NF}(\varphi(twq)) \equiv psu$ for some $p \in Q$, $s \in \mathcal{T}$ and $u \in Q^*$ such that $tp \sim ps$.*

We then use (0.5), Lemma 2, Lemma 5 and Lemma 7 to prove:

[1] Vershinin [22] defines a similar normal form for positive singular braid monoids; the only difference is that he sets $\alpha(x) = \tau_i$ where i is minimal with $\tau_i \preceq x$ if $\gcd(x, \Delta) = 1$. His normal form does not allow us to prove the central Lemma 14.

Lemma 9. *Let $w \equiv w_1 t_1 w_2 \dots w_k t_k w_{k+1} \in L$ with $t_i \in \mathcal{T}$ and $w_i \in Q^*$. Let $q \in Q \cup \{1\}$ and define $q_i \in Q \cup \{1\}$ for $1 \leqslant i \leqslant k+1$ by $q_{k+1} = q$ and $q_i = \gcd(\varphi(t_i w_{i+1} q_{i+1}), \Delta)$. Then there are $u_i \in Q^*$ and $s_i \in \mathcal{T}$ such that*

- $\mathrm{NF}(\varphi(t_i w_{i+1} q_{i+1})) \equiv q_i s_i u_{i+1}$,
- $\mathrm{NF}(\varphi(w_1 q_1)) \equiv u_1$, *and*
- $\mathrm{NF}(\varphi(wq)) \equiv u_1 s_1 u_2 \cdots u_k s_k u_{k+1}$.

6 Automaticity of M_Γ

If $p \in Q \cup \mathcal{T}$ let $W_p \in (\Sigma \cup T)^*$ be some representative of p. Define a homomorphism $\eta : (Q \cup \mathcal{T})^* \to (\Sigma \cup T)^*$ by $\eta(p) = W_p$ and let $K = \eta(L)$. We will show that $(\Sigma \cup T, K)$ is an automatic structure for the positive singular Artin monoid of finite type M_Γ. We first use Lemmas 3 and 4 to prove:

Lemma 10. *The set $L \subseteq (Q \cup \mathcal{T})^\star$ is regular.*

Given that homomorphic images of regular sets are regular and $K = \eta(L)$, we immediately deduce:

Lemma 11. *The set $K \subseteq (\Sigma \cup T)^*$ is regular.*

We then use Lemmas 6 and 7 to prove:

Lemma 12. *If $k \in [n]$ then the relation $L(\tau_k) = \{(u, v) \in L \times L \,:\, u\tau_k \sim v\}$ is rational.*

Our next aim is to prove that the relation $L(q) = \{(u, v) \in L \times L \,:\, uq \sim v\}$ is rational for $q \in Q$. To this aim, we consider the relations

$$H_p = \{(w, u) \in L^2 \,:\, w, u \in Q^*, wp \sim u\}$$
$$R_{p,r} = \{(w, u) \in L^2 \,:\, w, u \in Q^*, wp \sim ru\}$$

for $p, r \in Q$ and prove (using [9]):

Lemma 13. *For $p, r \in Q$, the relations H_p and $R_{p,r}$ are rational.*

We then use Lemma 9 to prove:

Lemma 14. *If $q \in Q$ then the relation $L(q)$ is rational.*

Finally we can prove Theorem 1. Let $\alpha \in \Sigma \cup \mathcal{T}$. Since $\Sigma \subseteq Q$ and $T \subseteq \mathcal{T}$, we can speak of the relation $L(\alpha)$ which is rational by Lemmas 12 and 14. Since rational relations are closed under the application of homomorphisms, the relation

$$K(\alpha) = \{(\eta(u), \eta(v)) \,:\, (u, v) \in L(\alpha)\}$$

is rational as well. Since the relation $K(\alpha)$ is difference bounded, Theorem 1 follows from Proposition 2 and Lemma 11.

7 Automaticity of the Singular Artin Monoid M_Γ^Δ

Antony [2] proved that M_Γ is a submonoid of M_Γ^Δ. Thus Δ, the lcm of Σ in M_Γ, may be considered as an element of M_Γ^Δ, and extending the automorphism $\overline{\cdot}$ to M_Γ^Δ in the natural way ($\overline{\sigma^{-1}} := \overline{\sigma_i}^{-1}$) preserves the property that $w\Delta = \Delta\overline{w}$, where w is now in M_Γ^Δ. Furthermore, since Δ is a common multiple of Σ in M_Γ, for each $\sigma \in \Sigma$, there is a unique $\Delta_\sigma \in M_\Gamma$ defined by the property $\Delta = \Delta_\sigma\sigma$. Thus, for each σ, $\Delta\sigma^{-1} = \Delta_\sigma$ lies in the submonoid M_Γ.

Define a language L^Δ over $Q \cup Q^{-1} \cup \mathcal{T}$ as follows:

$$L^\Delta := \{u^{-1}v \,:\, u \in L \cap Q^*, v \in L \text{ and } \gcd(\varphi(u), \varphi(v)) = 1\}.$$

Let $\pi : (Q \cup Q^{-1} \cup \mathcal{T})^* \to M_\Gamma^\Delta$ be the natural extension of $\varphi : (Q \cup \mathcal{T})^* \to M_\Gamma$; clearly π restricts to a map from L^Δ to M_Γ^Δ. We will write $u \approx v$ whenever $\pi(u) = \pi(v)$ holds. We then have:

Lemma 15. *The language L^Δ is a set of unique normal forms for M_Γ^Δ.*

We can also establish:

Lemma 16. *Let $u_x, u_y \in L \cap Q^*$ and $v_x, v_y \in L$ be such that $x \equiv u_x^{-1}v_x$ and $y \equiv u_y^{-1}v_y$ belong to L^Δ. Furthermore, let $\tau \in T$ and $\sigma \in \Sigma$. Then we have:*

(1) $x\tau \approx y$ if and only if $u_y \equiv u_x$ and $v_y \approx v_x\tau$;
(2) $x\sigma \approx y$ if and only if there exists $q \in Q$ such that $qu_y \approx u_x$ and $qv_y \approx v_x\sigma$;
(3) $x\sigma^{-1} \approx y$ if and only if there exists $q \in Q$ such that $qu_y \approx \Delta u_x$ and $qv_y \approx \overline{v_x}\Delta_\sigma$.

By [9], $(Q, L \cap Q^*)$ is a biautomatic structure for the positive Artin monoid S which is the submonoid of M_Γ generated by Q. Hence the language

$${}_qL = \{u \otimes v \,:\, u, v \in L, \varphi(qu) = \varphi(v)\}$$

is regular for $q \in Q$. This implies that the relation

$$R'(q) = \{(u, v) \,:\, u, v \in L \cap Q^*, \varphi(qu) = \varphi(v)\}$$

is also rational for any $q \in Q$. We can extend $R'(q)$ to all of L:

Lemma 17. *If $q \in Q$ then the relation $R(q) = \{(x, y) \in L \times L \,:\, qx \approx y\}$ is rational.*

We can then use Lemma 16 to deduce:

Lemma 18. *The relation $L^\Delta(t) = \{(x, y) \in L^\Delta \times L^\Delta \,:\, xt \approx y\}$ is rational for any $t \in \Sigma \cup \Sigma^{-1} \cup T$.*

Given all this we can, exactly as in the proof of Theorem 1, deduce Theorem 2.

References

1. Antony, N.: On singular Artin monoids and contributions to Birman's conjecture. Comm. Algebra 33, 4043–4056 (2005)
2. Antony, N.: The natural embedding of positive singular Artin monoids. Comm. Algebra 34, 3329–3346 (2006)
3. Baez, J.C.: Link invariants of finite type and perturbation theory. Lett. Math. Phys. 26, 43–51 (1992)
4. Baumslag, G., Gersten, S.M., Shapiro, M., Short, H.: Automatic groups and amalgams. J. Pure Appl. Algebra 76, 229–316 (1991)
5. Berstel, J.: Transductions and Context-free Languages. Teubner Studienbücher, Stuttgart (1979)
6. Birman, J.S.: New points of view in knot theory. Bull. Amer. Math. Soc. 28, 253–287 (1993)
7. Brieskorn, E., Saito, K.: Artin-Gruppen und Coxeter-Gruppen. Invent. Math. 17, 245–271 (1972)
8. Campbell, C.M., Robertson, E.F., Ruškuc, N., Thomas, R.M.: Automatic semigroups. Theoret. Comput. Sci. 250, 365–391 (2001)
9. Charney, R.: Artin groups of finite type are biautomatic. Math. Ann. 292, 671–683 (1992)
10. Corran, R.: A normal form for a class of monoids including the singular braid monoids. J. Algebra 223, 256–282 (2000)
11. Diekert, V., Rozenberg, G.: The Book of Traces. World Scientific Publ. Co., Singapore (1995)
12. East, J.: Birman's conjecture is true for $i_2(p)$. J. Knot Theory Ramifications 15, 167–177 (2006)
13. Eilenberg, S.: Automata, Languages and Machines, vol. A. Academic Press, New York (1974)
14. Epstein, D.B.A., Cannon, J.W., Holt, D.F., Levy, S.V.F., Paterson, M.S., Thurston, W.P.: Word Processing In Groups. Jones and Bartlett, Boston (1992)
15. Frougny, C., Sakarovitch, J.: Synchronized rational relations of finite and infinite words. Theoret. Comput. Sci. 108, 45–82 (1993)
16. Godelle, E., Paris, L.: On singular Artin monoids. In: Geometric Methods in Group Theory, Contemp. Math., vol. 372, pp. 43–57. Amer. Math. Soc., Providence (2005)
17. Hoffmann, M., Thomas, R.M.: Notions of automaticity in semigroups. Semigroup Forum 66, 337–367 (2003)
18. Hudson, J.F.P.: Regular rewrite systems and automatic structures. In: Semigroups, Automata and Languages, Porto, 1994, pp. 145–152. World Sci. Publishing, River Edge (1996)
19. Michel, J.: A note on words in braid monoids. J. Algebra 215, 366–377 (1999)
20. Otto, F., Sattler-Klein, A., Madlener, K.: Automatic monoids versus monoids with finite convergent presentations. In: Nipkow, T. (ed.) RTA 1998. LNCS, vol. 1379, pp. 32–46. Springer, Heidelberg (1998)
21. Paris, L.: The proof of Birman's conjecture on singular braid monoids. Geom. Topol. 8, 1281–1300 (2004)
22. Vershinin, V.: On the singular braid monoid. St. Petersburg Math. J. 21(5), 693–704 (2010)
23. Wang, B., Hu, Y.: Signature scheme based on the root extraction problem over braid groups. IET Information Security 3(2), 53–59 (2009)

Networks of Evolutionary Processors with Subregular Filters

Jürgen Dassow, Florin Manea*, and Bianca Truthe

Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
PSF 4120, D-39016 Magdeburg, Germany
{dassow,manea,truthe}@iws.cs.uni-magdeburg.de

Abstract. In this paper we propose a hierarchy of classes of languages, generated by networks of evolutionary processors with the filters in several special classes of regular sets. More precisely, we show that the use of filters from the class of ordered, non-counting, power-separating, circular, suffix-closed regular, union-free, definite and combinational languages is as powerful as the use of arbitrary regular languages and yields networks that can generate all the recursively enumerable languages. On the other hand, the use of filters that are only finite languages allows only the generation of regular languages, but not all regular languages can be generated. If we use filters that are monoids, nilpotent languages or commutative regular languages, we obtain the same family of languages which contains non-context-free languages but not all regular languages. These results seem to be of interest because they provide both upper and lower bounds on the classes of languages that one can use as filters in a network of evolutionary processor in order to obtain a complete computational model.

1 Introduction

An important part of theoretical computer science is the study of problems and processes connected with regular sets. In the last years a lot of papers appeared in which, for such problems and processes, the effect of going from arbitrary regular sets to special regular sets was studied. We here mention four such topics.

– It is a classical result that any nondeterministic finite automaton with n states can be transformed into a deterministic one with 2^n states, which accepts the same language, and that this exponential blow-up with respect to the number of states is necessary in the worst cases. In [2], this problem is studied if one restricts to the case that the automata accept special regular languages only. It is shown, that the situation does not change for suffix-closed and star-free regular languages; however, for some classes of definite languages, the size of the deterministic automaton is bounded by $2^{n-1}+1$.

* Also at: *Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, RO-010014 Bucharest, Romania (*flmanea@fmi.unibuc.ro*).* The work of Florin Manea is supported by the *Alexander von Humboldt Foundation*.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 262–273, 2011.

- A number α, $n \leq \alpha \leq 2^n$, is called magic (w.r.t. n), if there is no nondeterministic finite automaton with n states such that the minimal deterministic finite automaton has α states. It is known that no magic numbers exist if $n \geq 3$. This situation changes if one considers subregular families of languages. For instance, only the values α with $n + 1 \leq \alpha \leq 2^{n-1} + 1$ are possible for prefix-free regular languages (see [16]).
- In the last 20 years the behaviour of the (nondeterministic) state complexity under operations is intensively studied, i.e., it is asked for the size of the minimal (non)deterministic finite automaton for the language obtained from languages with given sizes. For many operations, the worst case is exactly determined. It has been shown that one gets smaller sizes if one restricts to special regular languages (see [13], [14], [3], and [17]).
- In order to enlarge the generative power, some mechanisms connected with regular languages were introduced, which control the derivations in context-free grammars. For instance, the sequence of applied rules in a regularly controlled grammar, the current sentential form in a conditional grammar and the levels of the derivation tree in a tree controlled grammar have to belong to given regular languages. In the papers [7], [9], [8], and [11], the change in the generative power, if one restricts to special regular sets, is investigated.

In this paper we continue the research along this direction. We consider the effect of special regular filters for generating evolutionary networks.

Networks of language processors have been introduced in [6] by E. Csuhaj-Varjú and A. Salomaa. Such a network can be considered as a graph where the nodes are sets of productions and at any moment of time a language is associated with a node. In a derivation step any node derives from its language all possible words as its new language. In a communication step any node sends those words to other nodes where the outgoing words have to satisfy an output condition given as a regular language (called output filter), and any node takes words sent by the other nodes if the words satisfy an input condition also given by a regular language (called input filter). The language generated by a network of language processors consists of all (terminal) words which occur in the languages associated with a given node.

Inspired by biological processes, in [4] a special type of networks of language processors was introduced which are called networks with evolutionary processors because the allowed productions model the point mutation known from biology. The sets of productions have to be substitutions of one letter by another letter or insertions of letters or deletion of letters; the nodes are then called substitution node or insertion node or deletion node, respectively. Results on networks of evolutionary processors can be found, e. g., in [4], [5], [18]. For instance. in [5], it was shown that networks of evolutionary processors are complete in that sense that they can generate any recursively enumerable language.

Modifications of evolutionary networks with evolutionary processors concern restrictions in the type of the nodes and the mode of applying a rule. In [1], it is investigated how the generative power behaves if one restricts to networks

with at most two types of nodes only. Moreover, in the case that one allows that some insertions and deletions can only be performed at the begin or end of the word one has also restricted to special regular filters given by random context conditions.

In this paper, we modify the filters. We require that the filters have to belong to a special subset of the set of all regular languages. We show that the use of filters from the class of ordered, non-counting, power-separating, circular, suffix-closed regular, union-free, definite and combinational languages is as powerful as the use of arbitrary regular languages and yields networks that can generate all the recursively enumerable languages. On the other hand, the use of filters that are only finite languages allows only the generation of regular languages, but not all regular languages can be generated. If we use filters that are monoids, nilpotent languages or commutative regular languages, we obtain the same family of languages which contains non-context-free languages but not all regular languages. These results seem to be of interest because they provide both upper and lower bounds on the classes of languages that one can use as filters in a network of evolutionary processor in order to obtain a complete computational model.

By reasons of space we omit some proofs. The omitted proofs can be found in [10] (see http://theo.cs.uni-magdeburg.de/pubs/preprints/pp-afl-2011-01.pdf).

2 Definitions

We assume that the reader is familiar with the basic concepts of formal language theory (see e. g. [19]). We here only recall some notations used in the paper.

By V^* we denote the set of all words (strings) over V (including the empty word λ). The length of a word w is denoted by $|w|$. By V^+ and V^k for some natural number k we denote the set of all non-empty words and the set of all words with length k, respectively. Let V_k be the set of all words over V with a length of at most k, i. e. $V_k = \bigcup_{i=0}^{k} V^i$.

By REG, CF, and RE we denote the families of regular, context-free, and recursively enumerable languages, respectively.

For a language L over V, we set

$$\begin{aligned} Comm(L) &= \{a_{i_1} \dots a_{i_n} \mid a_1 \dots a_n \in L,\ n \geq 1,\ \{i_1, i_2, \dots, i_n\} = \{1, 2, \dots, n\}\}, \\ Circ(L) &= \{vu \mid uv \in L,\ u, v \in V^*\}, \\ Suf(L) &= \{v \mid uv \in L,\ u, v \in V^*\} \end{aligned}$$

We consider the following restrictions for regular languages. Let L be a language and $V = alph(L)$ the minimal alphabet of L. We say that L is

- *combinational* iff it can be represented in the form $L = V^*A$ for some subset $A \subseteq V$,
- *definite* iff it can be represented in the form $L = A \cup V^*B$ where A and B are finite subsets of V^*,
- *nilpotent* iff L is finite or $V^* \setminus L$ is finite,

- *commutative* iff $L = Comm(L)$,
- *circular* iff $L = Circ(L)$,
- *suffix-closed* (or *fully initial* or *multiple-entry* language) iff $xy \in L$ for some $x, y \in V^*$ implies $y \in L$ (or equivalently, $Suf(L) = L$),
- *non-counting* (or star-free) iff there is an integer $k \geq 1$ such that, for any $x, y, z \in V^*$, $xy^k z \in L$ if and only if $xy^{k+1}z \in L$,
- *power-separating* iff for any $x \in V^*$ there is a natural number $m \geq 1$ such that either $J_x^m \cap L = \emptyset$ or $J_x^m \subseteq L$ where $J_x^m = \{x^n \mid n \geq m\}$,
- *ordered* iff L is accepted by some finite automaton $\mathcal{A} = (Z, V, \delta, z_0, F)$ where $(Z, \preceq)$ is a totally ordered set and, for any $a \in V$, the relation $z \preceq z'$ implies the relation $\delta(z, a) \preceq \delta(z', a)$,
- *union-free* iff L can be described by a regular expression which is only built by product and star.

It is obvious that combinational, definite, nilpotent, ordered and union-free languages are regular, whereas non-regular languages of the other types mentioned above exist.

By *COMB*, *DEF*, *NIL*, *COMM*, *CIRC*, *SUF*, *NC*, *PS*, *ORD*, and *UF* we denote the families of all combinational, definite, nilpotent, regular commutative, regular circular, regular suffix-closed, regular non-counting, regular power-separating, ordered, and union-free languages, respectively. Moreover, we add the family *MON* of all languages of the form V^*, where V is an alphabet (languages of *MON* are target sets of monoids; we call them monoidal languages). We set

$$\mathcal{G} = \{FIN, MON, COMB, DEF, NIL, COMM, CIRC, SUF, NC, PS, ORD, UF\}.$$

The relations between families of $\mathcal{G}$ are investigated e. g. in [15] and [20]. and their set-theoretic relations are given in Figure 1.

We call a production $\alpha \to \beta$ a
- substitution if $|\alpha| = |\beta| = 1$,
- deletion if $|\alpha| = 1$ and $\beta = \lambda$.

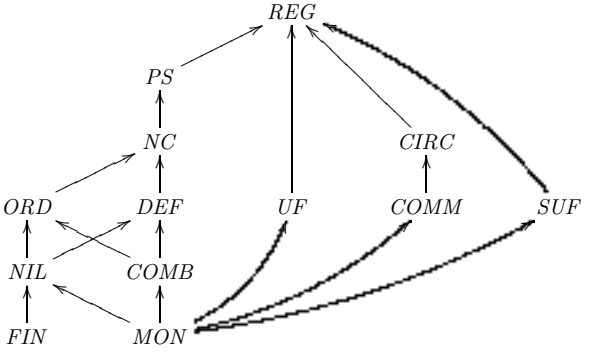


Fig. 1. Hierarchy of subregular languages (an arrow from X to Y denotes $X \subset Y$, and if two families are not connected by a directed path then they are incomparable)

The productions are applied like context-free rewriting rules. We say that a word v derives a word w, written as $v \Longrightarrow w$, if there are words x, y and a production $\alpha \to \beta$ such that $v = x\alpha y$ and $w = x\beta y$. If the rule p applied is important, we write $v \Longrightarrow_p w$.

We introduce insertion as a counterpart of deletion. We write $\lambda \to a$, where a is a letter. The application of an insertion $\lambda \to a$ derives from a word w any word $w_1 a w_2$ with $w = w_1 w_2$ for some (possibly empty) words w_1 and w_2.

We now introduce the basic concept of this paper, the networks of evolutionary processors (NEPs for short).

Definition 1. *Let X be a family of regular languages.*

(i) *A network of evolutionary processors (of size n) with filters of the set X is a tuple*

$$\mathcal{N} = (V, N_1, N_2, \ldots, N_n, E, j)$$

where
- *V is a finite alphabet,*
- *for $1 \leq i \leq n$, $N_i = (M_i, A_i, I_i, O_i)$ where*
  - *M_i is a set of rules of a certain type: $M_i \subseteq \{a \to b \mid a, b \in V\}$ or $M_i \subseteq \{a \to \lambda \mid a \in V\}$ or $M_i \subseteq \{\lambda \to b \mid b \in V\}$,*
  - *A_i is a finite subset of V^*,*
  - *I_i and O_i are languages from X over V,*
- *E is a subset of $\{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$, and*
- *j is a natural number such that $1 \leq j \leq n$.*

(ii) *A configuration C of $\mathcal{N}$ is an n-tuple $C = (C(1), C(2), \ldots, C(n))$ where $C(i)$ is a subset of V^* for $1 \leq i \leq n$.*

(iii) *Let $C = (C(1), C(2), \ldots, C(n))$ and $C' = (C'(1), C'(2), \ldots, C'(n))$ be two configurations of $\mathcal{N}$. We say that C derives C' in one*
- *evolutionary step (written as $C \Longrightarrow C'$) if, for $1 \leq i \leq n$, $C'(i)$ consists of all words $w \in C(i)$ to which no rule of M_i is applicable and of all words w for which there are a word $v \in C(i)$ and a rule $p \in M_i$ such that $v \Longrightarrow_p w$ holds,*
- *communication step (written as $C \vdash C'$) if, for $1 \leq i \leq n$,*

$$C'(i) = (C(i) \setminus O_i) \cup \bigcup_{(k,i) \in E} (C(k) \cap O_k \cap I_i).$$

The computation of an evolutionary network $\mathcal{N}$ is a sequence of configurations $C_t = (C_t(1), C_t(2), \ldots, C_t(n))$, $t \geq 0$, such that
- *$C_0 = (A_1, A_2, \ldots, A_n)$,*
- *for any $t \geq 0$, C_{2t} derives C_{2t+1} in one evolutionary step,*
- *for any $t \geq 0$, C_{2t+1} derives C_{2t+2} in one communication step.*

(iv) *The language $L(\mathcal{N})$ generated by $\mathcal{N}$ is defined as*

$$L(\mathcal{N}) = \bigcup_{t \geq 0} C_t(j)$$

where $C_t = (C_t(1), C_t(2), \ldots, C_t(n))$, $t \geq 0$ is the computation of $\mathcal{N}$.

Intuitively, a network with evolutionary processors is a graph consisting of some, say n, nodes $N_1, N_2, \ldots, N_n$ (called processors) and the set of edges given by E such that there is a directed edge from N_k to N_i if and only if $(k, i) \in E$. Any processor N_i consists of a set of evolutionary rules M_i, a set of words A_i, an input filter I_i and an output filter O_i. We say that N_i is a substitution node or a deletion node or an insertion node if $M_i \subseteq \{a \to b \mid a, b \in V\}$ or $M_i \subseteq \{a \to \lambda \mid a \in V\}$ or $M_i \subseteq \{\lambda \to b \mid b \in V\}$, respectively. The input filter I_i and the output filter O_i control the words which are allowed to enter and to leave the node, respectively. With any node N_i and any time moment $t \geq 0$ we associate a set $C_t(i)$ of words (the words contained in the node at time t). Initially, N_i contains the words of A_i. In an evolutionary step, we derive from $C_t(i)$ all words applying rules from the set M_i. In a communication step, any processor N_i sends out all words $C_t(i) \cap O_i$ (which pass the output filter) to all processors to which a directed edge exists (only the words from $C_t(i) \setminus O_i$ remain in the set associated with N_i) and, moreover, it receives from any processor N_k such that there is an edge from N_k to N_i all words sent by N_k and passing the input filter I_i of N_i, i. e., the processor N_i gets in addition all words of $C_t(k) \cap O_k \cap I_i$. We start with an evolutionary step and then communication steps and evolutionary steps are alternately performed. The language consists of all words which are in the node N_j (also called the output node, j is chosen in advance) at some moment t, $t \geq 0$.

For a family $X \subseteq REG$, we denote the family of languages generated by networks of evolutionary processors where all filters are of type X by $\mathcal{E}(X)$.

The following fact is obvious.

Lemma 1. *Let X and Y be subfamilies of REG such that $X \subseteq Y$. Then the inclusion $\mathcal{E}(X) \subseteq \mathcal{E}(Y)$ holds.*

The following theorem is known (see, e. g., [5]).

Theorem 1. *$\mathcal{E}(REG) = RE$.*

3 Some General Results

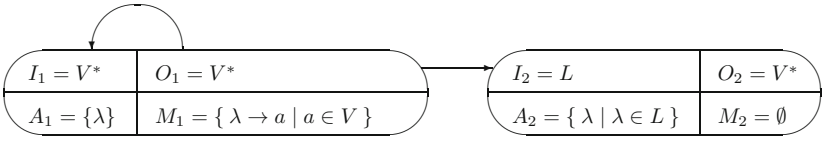
We start with some results which hold for every type of filters.

Lemma 2. *For every network $\mathcal{N}$ of evolutionary processors, there is a network $\mathcal{N}'$ of evolutionary processors that generates the same language as $\mathcal{N}$ and has the property that its output node N' has the form $N' = (\emptyset, \emptyset, I', O')$ for some regular languages I', O' over the network's working alphabet and no edge is leaving N'.*

Theorem 2. *Let $X \in \mathcal{G}$. Then each language $L \in X$ can be generated by a NEP $\mathcal{N}$ with at most two nodes and with filters from X.*

Proof. Let $X = FIN$. Let L be a finite set over V. Then the evolutionary network $(V, (\emptyset, L, \emptyset, \emptyset), \emptyset, 1)$ with all filters from FIN generates L.

If $X \neq FIN$, then $MON \subseteq X$ holds by Figure 1. Moreover, let $L \in X$ be a language over an alphabet V. We construct the NEP $\mathcal{N} = (V, N_1, N_2, E, 2)$ given as



Every word $w \in V^+$ will be derived in node N_1 and be communicated to node N_2 which accepts all words that also belong to L. The language generated by $\mathcal{N}$ is $L(\mathcal{N}) = A_2 \cup (V^+ \cap L) = L$. All filters are of type X.

Corollary 1. *For each class $X \in \mathcal{G}$, we have $X \subseteq \mathcal{E}(X)$.*

Corollary 2. *For each class $X \in \mathcal{G}$, we have $MON \subseteq \mathcal{E}(X)$.*

Proof. By the relations given in Figure 1 and Corollary 1, it is sufficient to show that $MON \subseteq \mathcal{E}(FIN)$. Let V be an alphabet and $L = V^*$. Then the evolutionary network $(V, (\{ \lambda \to a \mid a \in V \}, \{\lambda\}, \emptyset, \emptyset), \emptyset, 1)$ with all filters from FIN generates L. Thus, any monoidal language $L = V^*$ belongs to $\mathcal{E}(FIN)$.

4 Computationally Complete Cases

In this section we present the computational completeness of some families $\mathcal{E}(X)$.

Theorem 3. *$\mathcal{E}(SUF) = RE$ and $\mathcal{E}(CIRC) = RE$.*

Proof. First we show that $\mathcal{E}(SUF) = RE$.

Let L be a recursively enumerable set. Let $\mathcal{N} = (V, N_1, N_2, \ldots, N_n, E, j)$ be a network with evolutionary processors and filters from REG such that $L(\mathcal{N}) = L$. For any node $N_i = (M_i, A_i, I_i, O_i)$, we construct the sets

$$I_i' = \{X\}I_i\{Y\} \cup Suf(I_i)\{Y\} \cup \{\lambda\},$$
$$O_i' = \{X\}O_i\{Y\} \cup Suf(O_i)\{Y\} \cup \{\lambda\},$$

where X and Y are two new symbols. By definition, I_i' and O_i' are suffix-closed. We assume that the network $\mathcal{N}$ has the property $N_j = (\emptyset, \emptyset, I_j, O_j)$ and no edge leaves the output node (according to the previous Lemma).

We consider the network

$$\mathcal{N}' = (V \cup \{X, Y\}, N_1', N_2', \ldots, N_n', N_{n+1}', N_{n+2}', E', n+2)$$

with

$$\begin{aligned} N_i' &= (M_i, \{X\}A_i\{Y\}, I_i', O_i') \text{ for } 1 \le i \le n, \\ N_{n+1}' &= (\{X \to \lambda,\ Y \to \lambda\}, \emptyset, I_j', V^*), \\ N_{n+2}' &= (\emptyset, \emptyset, V^*, \emptyset), \\ E' &= E \cup \{ (i, n+1) \mid (i, j) \in E \} \cup \{ (n+1, n+2) \}. \end{aligned}$$

It is obvious that the filters of N'_{n+1} and N'_{n+2} are suffix-closed, too. Thus $\mathcal{N}'$ is a network of type SUF.

We now prove that $L(\mathcal{N}) = L(\mathcal{N}')$. We start with words of the form XwY and as long as these words are changed according to rules of M_i, $1 \leq i \leq n$, they can only be sent to nodes N'_s, $1 \leq s \leq n$, and N'_{n+1}. Thus we simulate a derivation in $\mathcal{N}$ (in $\mathcal{N}'$ we have an X in front of and a Y behind the word w occurring in $\mathcal{N}$) and get into N'_{n+1} exactly those words XwY whose subword w comes into N_j. Now X and Y are removed and the resulting word w is sent to N'_{n+2}. Other words cannot arrive in N'_{n+2} and other words do not appear in N_j. Hence, $L(\mathcal{N}') = L(\mathcal{N})$.

To show that $\mathcal{E}(CIRC) = RE$, we repeat the previous proof with the following modifications. We set

$$I'_i = Circ(\{X\}I_i\{Y\}) \text{ and } O'_i = Circ(\{X\}O_i\{Y\}) \text{ for } 1 \leq i \leq n.$$

This ensures that $Circ(F) = F$ for all filters F of the new network $\mathcal{N}'$. Then the proof proceeds as in the case of suffix-closed filters.

Theorem 4. $\mathcal{E}(COMB) = \mathcal{E}(DEF) = \mathcal{E}(UF) = RE$.

By the relations shown in Figure 1, Lemma 1, and Theorem 1, we obtain the following theorem.

Theorem 5. $\mathcal{E}(ORD) = \mathcal{E}(NC) = \mathcal{E}(PS) = RE$.

5 Computationally Non-complete Cases

We first discuss the case of finite filters. We start with a certain normal form for networks with finite filters.

Lemma 3. *For each NEP $\mathcal{N}$ with only finite filters, we can construct a NEP $\mathcal{N}'$ with only one processor and finite filters that generates the same language as $\mathcal{N}$.*

Theorem 6. $\mathcal{E}(FIN) \subset REG$.

Proof. Let $\mathcal{N} = (V, N_1, N_2, \ldots, N_n, E, j)$ be a network with finite filters. Obviously, a word w is in N_j if and only if it is in A_j or satisfies I_j or is obtained from a word in N_j by application of a rule in M_j. We set

$$U = \{\, a \mid \lambda \to a \in M_j \,\},\ V' = \{\, a' \mid a \in V \,\},\ \text{and } U' = \{\, a' \mid a \in U \,\}.$$

Let $h : (V \cup V')^* \to V^*$ be the homomorphism defined by

$$h(a) = a \text{ for } a \in V \quad \text{and} \quad h(a') = \begin{cases} \lambda, & \text{for } a' \in U', \\ a, & \text{for } a' \in V' \setminus U', \end{cases}$$

and $\tau : (V \cup V')^* \to V^*$ be the finite substitution where $\tau(a) = \tau(a')$ for $a \in V$ and $\tau(a)$ consists of all $b \in V \cup \{\lambda\}$ such that there are an integers

$s \geq 0$ and $b_0, b_1, \ldots, b_{s-1} \in V$ and $b_s \in V \cup \{\lambda\}$ such that $a = b_0$, $b = b_s$, and $b_i \to b_{i+1} \in M_j$ for $0 \leq i \leq s-1$ (note that $s = 0$ implies $a = b$). Furthermore, let

$$k = \max\{\, |w| \mid w \in O_j \cup I_j \cup A_j \,\} + 1.$$

We note the following facts:

- Assume that there is a word w of length at least k in $L(\mathcal{N})$. Then w is in $C_t(j)$ for some t. By its length, it cannot leave the node, and thus all words which have a length at least k and can be obtained by application of rules of M_j to w belong to $L(\mathcal{N})$, too.
- If w with $|w| \geq k+1$ is in $L(\mathcal{N})$, then w is obtained from a word $v \in L(\mathcal{N})$ of length k by application of rules in M_j (since substitutions and deletions do not increase the length, the shortest words in $L(\mathcal{N})$ with length at least k are obtained by an insertion from a word of length less than k and thus they have length k).

Now it is easy to see that

$$L(\mathcal{N}) = (L(\mathcal{N}) \cap \bigcup_{i=0}^{k-1} V^i) \cup (\tau(h^{-1}(L(\mathcal{N}) \cap V^k)) \cap \bigcup_{i \geq k} V^i)$$

holds. Since finite languages are regular and regular languages are closed under inverse homomorphisms, finite substitutions, intersection, and union, $L(\mathcal{N})$ is regular. Hence $\mathcal{E}(FIN) \subseteq REG$ holds.

Let $V = \{a\}$ and $L = \{a\} \cup \{\, a^n \mid n \geq 3 \,\}$. Obviously, L is regular.

Suppose the language L is generated by a network with only finite filters. Then, by Lemma 3, there is a network $\mathcal{N}$ with only one node $N = (M, A, \emptyset, O)$ that generates L. Since L is infinite, this node must be inserting. Hence, the rule set is $M = \{\, \lambda \to a \,\}$. If the initial set A contains λ then $\lambda \in L(\mathcal{N})$ which is in contrast to $\lambda \notin L$. If the initial set A contains a or aa then the word aa belongs to the generated language $L(\mathcal{N})$ which is in contrast to $aa \notin L$. If the initial set only contains words a^n with $n \geq 3$ then the word a cannot be generated but $a \in L$ which is a contradiction, too. Hence, there is no network with only finite filters that generates L. Thus, $L \in REG \setminus \mathcal{E}(FIN)$.

The following result shows that the use of filters from the remaining language families, i. e., from MON or NIL or $COMM$ leads to the same class of languages.

Theorem 7. $\mathcal{E}(MON) = \mathcal{E}(COMM) = \mathcal{E}(NIL)$.

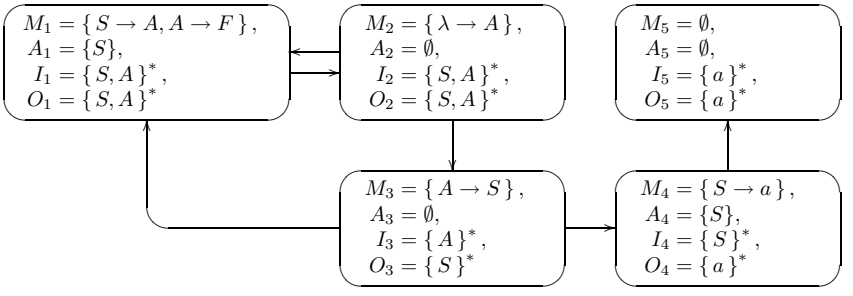
We now present some relations of $\mathcal{E}(MON)$ to other language families.

Theorem 8. $\mathcal{E}(FIN) \subset \mathcal{E}(MON)$.

Proof. Since $FIN \subset NIL$, we obtain $\mathcal{E}(FIN) \subseteq \mathcal{E}(NIL)$ by Lemma 1. By Theorem 2, the nilpotent language $L = \{a\} \cup \{\, a^n \mid n \geq 3 \,\}$ is contained in $\mathcal{E}(NIL)$. However, by the second part of the proof of Theorem 6, L is not contained in $\mathcal{E}(FIN)$. Thus $\mathcal{E}(FIN) \subset \mathcal{E}(NIL)$. The statement now follows from Theorem 7.

Lemma 4. *The family $\mathcal{E}(MON)$ contains a non-semi-linear (hence non-regular and non-context-free) language.*

Proof. Let $V = \{ S, A, F, a \}$ and $\mathcal{N} = (V, N_1, N_2, N_3, N_4, N_5, E, 5)$ be the following network:



In the beginning, we have the word S in node N_1. We consider a word S^n for $n \geq 1$ in node N_1 in an even moment (in the beginning or after a communication step). One occurrence of S is replaced by A, then the word is sent to node N_2 where another copy of A is inserted. This word w goes back to node N_1 and it goes on to node N_3 which takes it if no S appears in the word. If in N_1 the rule $A \to F$ is applied then the symbol F is introduced which cannot be replaced. Due to the output filter O_1, the word will be trapped in N_1 for ever. If, in the word w, no S is present then the only rule which can be applied is $A \to F$ and the cycle is stopped. If w still contains an S then it is replaced by A and N_2 inserts another A. So, the words move between N_1 and N_2 where alternatingly an S is replaced by A and an A is inserted until the word only contains As. The word is then A^{n+1}. Hence, the number of letters has been doubled.

In N_3, each A is replaced by S. The word is S^{n+1} when it leaves N_3. It moves to N_1 and to N_4. In N_1, the cycle starts again with a word S^m for $m \geq 1$. All arriving words in N_4 have the form S^n with $n \geq 2$. In order to cover also the case $n = 1$, the initial language of this node consists of S. In N_4, every letter S is replaced by the symbol a before the word leaves to node and moves to the output node N_5.

Hence, $L(\mathcal{N}) = \{ a^{2^n} \mid n \geq 0 \}$.

Corollary 3. *$NIL \subset \mathcal{E}(MON)$ and $COMM \subset \mathcal{E}(MON)$.*

Proof. The inclusions follow from Corollary 1 and Theorem 7. The strictness follows from Lemma 4.

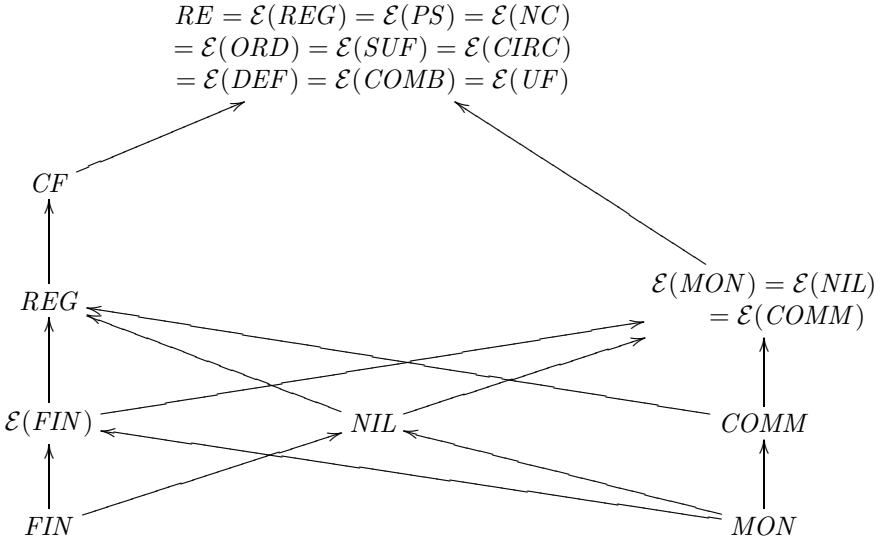
Finally, we give a result which can be understood as a lower bound for the generative power of monoidal filters.

Theorem 9. *Let L be a semi-linear language. Then $Comm(L) \in \mathcal{E}(MON)$.*

6 Conclusion

If we combine all the results of the preceding sections, we get the following diagram which we state as a theorem.

Theorem 10. *The following diagram holds.*



The subregular classes considered in this paper are defined by combinatorial or algebraic properties of the languages. In [12], subclasses of REG defined by descriptional complexity have been considered. Let REG_n be the set of regular languages which can be accepted by deterministic finite automata. Then we have

$$REG_1 \subset REG_2 \subset REG_3 \subset \cdots \subset REG_n \subset \cdots \subset REG.$$

By Lemma 4 and [12], Lemma 4.1 and Theorems 4.3, 4.4., and 4.5, we get

$$\mathcal{E}(REG_1) \subset \mathcal{E}(MON) \subset \mathcal{E}(REG_2) = \mathcal{E}(REG_3) = \cdots = RE$$

and the incomparability of $\mathcal{E}(REG_1)$ with REG and CF.

References

1. Alhazov, A., Dassow, J., Martín-Vide, C., Rogozhin, Y., Truthe, B.: On networks of evolutionary processors with nodes of two types. Fundamenta Informaticae 91, 1–15 (2009)
2. Bordihn, H., Holzer, M., Kutrib, M.: Determinization of finite automata accepting subregular languages. Theoretical Computer Science 410, 3209–3222 (2009)
3. Brzozowski, J., Jirásková, G., Li, B.: Quotient complexity of ideal languages. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 208–221. Springer, Heidelberg (2010)

4. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: Solving NP-complete problems with networks of evolutionary processors. In: Mira, J., Prieto, A.G. (eds.) IWANN 2001. LNCS, vol. 2084, pp. 621–628. Springer, Heidelberg (2001)
5. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: Networks of evolutionary processors. Acta Informatica 39(6-7), 517–529 (2003)
6. Csuhaj-Varjú, E., Salomaa, A.: Networks of parallel language processors. In: Păun, G., Salomaa, A. (eds.) New Trends in Formal Languages. LNCS, vol. 1218, pp. 299–318. Springer, Heidelberg (1997)
7. Dassow, J.: Subregularly controlled derivations: the context-free case. Rostocker Mathematisches Kolloquium 34, 61–70 (1988)
8. Dassow, J.: Grammars with commutative, circular, and locally testable conditions. In: Automata, Formal Languages, and Related Topics – Dedicated to Ferenc Gécseg on the Occasion of his 70th Birthday, pp. 27–37. University of Szeged (2009)
9. Dassow, J., Hornig, H.: Conditional grammars with subregular conditions. In: Proc. Internat. Conf. Words, Languages and Combinatorics II, pp. 71–86. World Scientific, Singapore (1994)
10. Dassow, J., Manea, F., Truthe, B.: Networks of evolutionary processors with subregular filters. Technical report, Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik (2011), http://theo.cs.uni-magdeburg.de/pubs/preprints/pp-afl-2011-01.pdf
11. Dassow, J., Stiebe, R., Truthe, B.: Generative capacity of subregularly tree controlled grammars. International Journal of Foundations of Computer Science 21, 723–740 (2010)
12. Dassow, J., Truthe, B.: On networks of evolutionary processors with filters accepted by two-state-automata. Fundamenta Informaticae (to appear)
13. Han, Y.S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. Theoretical Computer Science 410(27–29), 2537–2548 (2009)
14. Han, Y.S., Salomaa, K., Wood, D.: Nondeterministic state complexity of basic operations for prefix-suffix-free regular languages. Fundamenta Informaticae 90(1-2), 93–106 (2009)
15. Havel, I.M.: The theory of regular events II. Kybernetika 5(6), 520–544 (1969)
16. Holzer, M., Jakobi, S., Kutrib, M.: The magic number problem for subregular language families. In: Proceedings of 12th Internat. Workshop Descriptional Complexity of Formal Systems, pp. 135–146. University of Saskatchewan, Saskatoon (2010)
17. Jirásková, G., Masopust, T.: Complexity in union-free languages. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 255–266. Springer, Heidelberg (2010)
18. Martín-Vide, C., Mitrana, V.: Networks of evolutionary processors: Results and perspectives. In: Molecular Computational Models: Unconventional Approaches, pp. 78–114 (2005)
19. Rozenberg, G., Salomaa, A.: Handbook of Formal Languages. Springer, Berlin (1997)
20. Wiedemann, B.: Vergleich der Leistungsfähigkeit endlicher determinierter Automaten. Diplomarbeit, Universität Rostock (1978)

Decision Problems for Interval Markov Chains*

Benoît Delahaye[1], Kim G. Larsen[2], Axel Legay[3],
Mikkel L. Pedersen[2], and Andrzej Wąsowski[4]

[1] Université de Rennes 1/IRISA, France
[2] Aalborg University, Denmark
[3] INRIA/IRISA, France
[4] IT University of Copenhagen, Denmark

Abstract. Interval Markov Chains (IMCs) are the base of a classic probabilistic specification theory by Larsen and Jonsson in 1991. They are also a popular abstraction for probabilistic systems.

In this paper we study complexity of several problems for this abstraction, that stem from compositional modeling methodologies. In particular we close the complexity gap for thorough refinement of two IMCs and for deciding the existence of a common implementation for an unbounded number of IMCs, showing that these problems are EXPTIME-complete. We also prove that deciding consistency of an IMC is polynomial and discuss suitable notions of determinism for such specifications.

1 Introduction

Interval Markov Chains (IMCs for short) extend Markov Chains, by allowing to specify intervals of possible probabilities on state transitions. IMCs have been introduced by Larsen and Jonsson [10] as a *specification* formalism—a basis for a stepwise-refinement-like modeling method, where initial designs are very abstract and underspecified, and are then made continuously more precise, until they are concrete. Unlike richer specification models such as Constraint Markov Chains [4], IMCs are difficult to use for compositional specification due to lack of basic modeling operators. To address this, we study complexity and algorithms for deciding consistency of conjunctive sets of IMC specifications.

In [10] Jonsson and Larsen have introduced refinement for IMCs, but have not determined its computational complexity. We complete their work on refinement by classifying its complexity and characterizing it using structural coinductive algorithms in the style of simulation.

Consider the issue of combining multiple specifications of the same system. It turns out that conjunction of IMCs cannot be expressed as an IMC itself, due to a lack of expressiveness of intervals. Let us demonstrate this using a simple specification of a user of a coffee machine. Let the model prescribe that a typical user orders coffee with milk with probability $x \in [0, 0.5]$ and black coffee with

* This work was supported by the European STREP-COMBEST project no. 215543, by VKR Centre of Excellence MT-LAB, and by an "Action de Recherche Collaborative" ARC (TP)I.

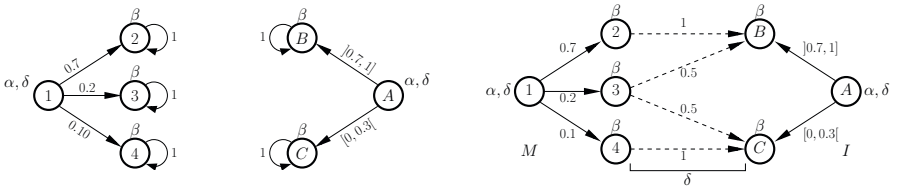
A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 274–285, 2011.

probability $y \in [0.2, 0.7]$ (customers also buy tea with probability $t \in [0, 0.5]$). The vendor of the machine delivers another specification, which prescribes that the machine is serviceable only if coffee (white or black) is ordered with some probability $z \in [0.4, 0.8]$ from among other beverages, otherwise it will run out of coffee powder too frequently, or the powder becomes too old. A conjunction of these two models would describe users who have use patterns compatible with this particular machine. Such a conjunction effectively requires that all the interval constraints are satisfied and that $z = x + y$ holds. However, the solution of this constraint is not described by an interval over x and y. This can be seen by pointing out an extremal point, which is not a solution, while all its coordinates take part in some solution. Say $x = 0$ and $y = 0.2$ violates the interval for z, while for each of these two values it is possible to select another one in such a way that z's constraint is also held (for example $(x = 0, y = 0.4)$ and $(x = 0.2, y = 0.2)$). Thus the solution space is not an interval over x and y.

This lack of closure properties for IMCs motivates us to address the problem of reasoning about conjunction without constructing it — the, so called, common implementation problem. In this paper we provide algorithms and complexity results for consistency, common implementation and refinement of IMCs, in order to enable compositional modeling. We contribute the following new results:

- In [10] a *thorough refinement* (TR) between IMCs is defined as an inclusion of implementation sets. We define suitable notions of determinism for IMCs, and show that for deterministic IMCs TR coincides with two simulation-like preorders (the *weak refinement* and *strong refinement*), for which there exist co-inductive algorithms terminating in a polynomial number of iterations.
- We show that the thorough refinement procedure given in [10] can be implemented in single exponential time. Furthermore we provide a lower bound, concluding that TR is EXPTIME-complete. While the reduction from TR of modal transition systems [3] used to provide this lower bound is conceptually simple, it requires a rather involved proof of correctness, namely that it preserves sets of implementations in a sound and complete manner.
- A polynomial procedure for checking whether an IMC is *consistent* (C), i.e. it admits a Markov Chain as an implementation.
- An exponential procedure for checking whether k IMCs are consistent in the sense that they share a Markov Chain satisfying all—a *common implementation* (CI). We show that this problem is EXPTIME-complete.
- As a special case we observe, that CI is PTIME for any constant value of k. In particular checking whether two specifications can be simultaneously satisfied, and synthesizing their shared implementation can be done in polynomial time.

For functional analysis of discrete-time non-probabilistic systems, the theory of Modal Transition Systems (MTSs) [15] provides a specification formalism supporting refinement, conjunction and parallel composition. Earlier we have obtained EXPTIME-completeness both for the corresponding notion of CI [2] and of TR [3] for MTSs. In [10] it is shown that IMCs properly contain MTSs, which puts our new results in a somewhat surprising light: in the complexity



(a) A Markov Chain M (b) An IMC I (c) An example of satisfaction relation

Fig. 1. Examples of Markov Chains, Interval Markov Chains and satisfaction relation

theoretic sense, and as far as CI and TR are considered, the generalization of modalities by probabilities does come for free.

The paper proceeds as follows. In Section 2 we introduce the basic definitions. All results in subsequent sections are new and ours. In Section 3 we discuss deciding TR and other refinement procedures. We expand on the interplay of determinism and refinements in Section 4. The problems of C and CI are addressed in Section 5. We close by discussing the results and related work in Section 6. Due to space constraints, some algorithms and proofs are given in a long version of this paper [6].

2 Background

We shall now introduce the basic definitions used throughout the paper. In the following we will write $\mathsf{Intervals}_{[0,1]}$ for the set of all closed, half-open and open intervals included in $[0, 1]$.

We begin with settling notation for Markov Chains. A Markov Chain (sometimes MC in short) is a tuple $C = \langle P, p_0, \pi, A, V_C \rangle$, where P is a set of states containing the initial state p_0, A is a set of atomic propositions, $V_C : P \to 2^A$ is a state valuation labeling states with propositions, and $\pi : P \to \mathsf{Distr}(P)$ is a probability distribution assignment such that $\sum_{p' \in P} \pi(p)(p') = 1$ for all $p \in P$. The probability distribution assignment is the only component that is relaxed in IMCs:

Definition 1 (Interval Markov Chain). *An* Interval Markov Chain *is a tuple* $I = \langle Q, q_0, \varphi, A, V_I \rangle$*, where* Q *is a set of states containing the initial state* q_0*,* A *is a set of atomic propositions,* $V_I : Q \to 2^A$ *is a state valuation, and* $\varphi : Q \to (Q \to \mathsf{Intervals}_{[0,1]})$*, which for each* $q \in Q$ *and* $q' \in Q$ *gives an interval of probabilities.*

Instead of a distribution, as in MCs, in IMCs we have a function mapping elementary events (target states) to intervals of probabilities. We interpret this function as a constraint over distributions. This is expressed in our notation as follows. Given a state $q \in Q$ and a distribution $\sigma \in \mathsf{Distr}(Q)$, we say that $\sigma \in \varphi(q)$ iff $\sigma(q') \in \varphi(q)(q')$ for all $q' \in Q$. Occasionally, it is convenient to think of a Markov Chain as an IMC, in which all probability intervals are closed point intervals.

We visualize IMCs as automata with intervals on transitions. As an example, consider the IMC in Figure 1b. It has two outgoing transitions from the initial state A. No arc is drawn between states if the probability is zero (or more precisely the interval is $[0, 0]$), so in the example there is zero probability of going from state A to A, or from B to C, etc. Otherwise the probability distribution over successors of A is constrained to fall into $]0.7, 1]$ and $[0, 0.3[$ for B and C respectively. States B and C have valuation β, whereas state A has valuation α, δ. Figure 1a presents a Markov Chain using the same convention, modulo the intervals. Notice that our formalism does not allow "sink states" with no outgoing transitions. In the figures, states with no outgoing transitions are meant to have a self-loop transition with probability 1 (a closed point interval).

There are three known ways of defining refinement for IMCs: strong refinement (introduced as *simulation* in [10]), weak refinement (introduced under the name of *probabilistic simulation* in [7]), and thorough refinement (introduced as *refinement* in [10]). We recall their formal definitions:

Definition 2 (Strong Refinement). *Let* $I_1 = \langle Q, q_0, \varphi_1, A, V_1 \rangle$ *and* $I_2 = \langle S, s_0, \varphi_2, A, V_2 \rangle$ *be IMCs. A relation* $\mathcal{R} \subseteq Q \times S$ *is a* strong refinement relation *if whenever* $q \mathcal{R} s$ *then*

1. *The valuation sets agree:* $V_1(q) = V_2(s)$ *and*
2. *There exists a correspondence function* $\delta : Q \to (S \to [0, 1])$ *such that, for all* $\sigma \in \mathsf{Distr}(Q)$*, if* $\sigma \in \varphi_1(q)$*, then*
   (a) *for all* $q' \in Q$ *such that* $\sigma(q') > 0$*,* $\delta(q')$ *is a distribution on* S*,*
   (b) *for all* $s' \in S$*, we have* $\sum_{q' \in Q} \sigma(q') \cdot \delta(q')(s') \in \varphi_2(s)(s')$*, and*
   (c) *for all* $q' \in Q$ *and* $s' \in S$*, if* $\delta(q')(s') > 0$*, then* $q' \mathcal{R} s'$*.*

I_1 *strongly refines* I_2*, or* $I_1 \leq_{\mathsf{S}} I_2$*, iff there exists a strong refinement containing* (q_0, s_0)*.*

A strong refinement relation requires the existence of a single correspondence, which witnesses satisfaction for any resolution of probability constraint over successors of q and s. Figure 2a illustrates such a correspondence between states A and α of two IMCs. The correspondence function is given by labels on the dashed lines. It is easy to see that, regardless of how the probability constraints are resolved, the correspondence function distributes the probability mass in a fashion satisfying α.

A *weak refinement* relation requires that, for any resolution of probability constraint over successors in I_1, there exists a correspondence function, which witnesses satisfaction of I_2. The formal definition of weak refinement is identical to Def. 2, except that the condition opening Point (2) is replaced by a weaker one:

Definition 3 (Weak Refinement). *Let* $I_1 = \langle Q, q_0, \varphi_1, A, V_1 \rangle$ *and* $I_2 = \langle S, s_0, \varphi_2, A, V_2 \rangle$ *be IMCs. A relation* $\mathcal{R} \subseteq Q \times S$ *is a* weak refinement relation *if whenever* $q \mathcal{R} s$*, then*

1. *The valuation sets agree:* $V_1(q) = V_2(s)$ *and*
2. *For each* $\sigma \in \mathsf{Distr}(Q)$ *such that* $\sigma \in \varphi_1(q)$*, there exists a correspondence function* $\delta : Q \to (S \to [0,1])$ *such that*
   (a) *for all* $q' \in Q$ *such that* $\sigma(q') > 0$*,* $\delta(q')$ *is a distribution on* S*,*
   (b) *for all* $s' \in S$*, we have* $\sum_{q' \in Q} \sigma(q') \cdot \delta(q')(s') \in \varphi_2(s)(s')$*, and*
   (c) *for all* $q' \in Q$ *and* $s' \in S$*, if* $\delta(q')(s') > 0$*, then* $q' \mathcal{R} s'$*.*

I_1 *weakly refines* I_2*, or* $I_1 \leq_{\mathsf{W}} I_2$*, iff there exists a weak refinement containing* (q_0, s_0)*.*

Figure 2b illustrates a weak refinement between states A and α of another two IMCs. Here, x stands for a value in $[0.2, 1]$ (arbitrary choice of probability of going to state C from A). Notably, for each choice of x, there exists $p \in [0,1]$ such that $p \cdot x \in [0, 0.6]$ and $(1-p) \cdot x \in [0.2, 0.4]$.

Satisfaction Relation. This relation establishes compatibility of Markov Chains (implementations) and IMCs (specifications). The original definition has been presented in [10,11]. Consider a Markov chain $C = \langle P, p_0, \pi, A, V_C \rangle$ as an IMC with only closed point interval probabilities, and let $I = \langle Q, q_0, \varphi, A, V_I \rangle$ be an IMC. We say that C satisfies I, written $C \models I$, iff there exists a weak/strong refinement relation $\mathcal{R} \subseteq P \times Q$, called a *satisfaction relation*, containing (p_0, q_0). Remark that when C is a Markov Chain, the weak and strong notions of refinement coincide. Whenever $C \models I$, C is called an *implementation* of I. The set of implementations of I is written $[\![I]\!]$. Figure 1c presents an example of satisfaction on states 1 and A. The correspondence function is specified using labels on the dashed arrows i.e. the probability mass going from state 1 to 3 is distributed to state B and C with half going to each.

We say that a state q of an IMC is *consistent* if its interval constraint $\varphi(q)$ is satisfiable, i.e. there exists a distribution $\sigma \in \mathsf{Distr}(Q)$ satisfying $\varphi(q)$. Obviously, for a given IMC, it is sufficient that all its states are consistent in order to guarantee that the IMC is consistent itself—there exists a Markov Chain satisfying it. We discuss the problem of establishing consistency in a sound and complete manner in Section 5.

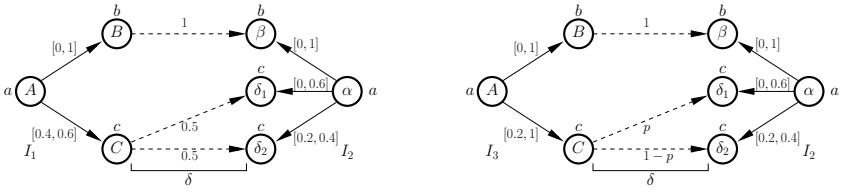
Finally, we introduce the thorough refinement as defined in [10]:

Definition 4 (Thorough Refinement). *IMC* I_1 thoroughly refines *IMC* I_2*, written* $I_1 \leq_{\mathsf{T}} I_2$*, iff each implementation of* I_1 *implements* I_2*:* $[\![I_1]\!] \subseteq [\![I_2]\!]$

Thorough refinement is the ultimate refinement relation for any specification formalism, as it is based on the semantics of the models.

3 Refinement Relations

In this section, we compare the expressiveness of the refinement relations. It is not hard to see that both strong and weak refinements soundly approximate the thorough refinement (since they are transitive and degrade to satisfaction if the



(a) Illustration of a strong refinement relation between an IMC I_1 and an IMC I_2.

(b) Illustration of a weak refinement relation between an IMC I_3 and an IMC I_2.

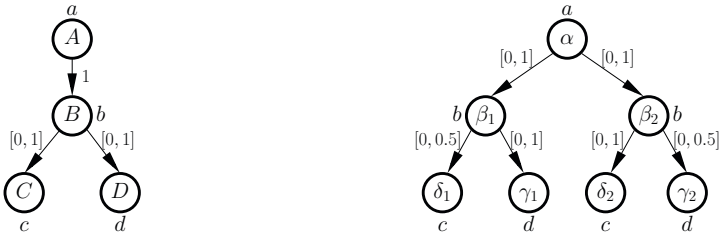
Fig. 2. Illustration of strong and weak refinement relations

left argument is a Markov Chain). The converse does not hold. We will now discuss procedures to compute weak and strong refinements, and then compare the granularity of these relations, which will lead us to procedures for computing thorough refinement. Observe that both refinements are decidable, as they only rely on the first order theory of real numbers. In concrete cases below the calculations can be done more efficiently due to convexity of solution spaces for interval constraints.

Weak and Strong Refinement. Consider two IMCs $I_1 = \langle P, o_1, \varphi_1, A, V_1\rangle$ and $I_2 = \langle Q, o_2, \varphi_2, A, V_2\rangle$. Informally, checking whether a given relation $\mathcal{R} \subseteq P \times Q$ is a weak refinement relation reduces to checking, for each pair $(p, q) \in \mathcal{R}$, whether the following formula is true: $\forall \pi \in \varphi_1(p),\ \exists \delta : P \to (Q \to [0, 1])$ such that $\pi \times \delta$ satisfies a system of linear equations / inequations. Since the set of distributions satisfying $\varphi_1(p)$ is convex, checking such a system is exponential in the number of variables, here $|P| \cdot |Q|$. As a consequence, checking whether a relation on $P \times Q$ is a weak refinement relation is exponential in $|P| \cdot |Q|$. For strong refinement relations, the only difference appears in the formula that must be checked: $\exists \delta : P \to (Q \to [0, 1])$ such that $\forall \pi \in \varphi_1(p)$, we have that $\pi \times \delta$ satisfies a system of linear equations / inequations. Therefore, checking whether a relation on $P \times Q$ is a strong refinement relation is also exponential in $|P| \cdot |Q|$.

Deciding whether weak (strong) refinement holds between I_1 and I_2 can be done in the usual coinductive fashion by considering the total relation $P \times Q$ and successively removing all the pairs that do not satisfy the above formulae. The refinement holds iff the relation we reach contains the pair (o_1, o_2). The algorithm will terminate after at most $|P| \cdot |Q|$ iterations. This gives an upper bound on the complexity to establish strong and weak refinements: a polynomial number of iterations over an exponential step. This upper bound may be loose. One could try to reuse techniques for nonstochastic systems [9] in order to reduce the number of iterations. This is left to future work.

Granularity. In [10] an informal statement is made that the strong refinement is strictly stronger (finer) than the thorough refinement: $(\leq_{\mathsf{T}}) \supsetneq (\leq_{\mathsf{S}})$. In [7] the weak refinement is introduced, but without discussing its relations to neither the strong nor the thorough refinement. The following theorem resolves all open issues in relations between the three:



(a) IMC I_4 (b) IMC I_5

Fig. 3. IMCs I_4 and I_5 such that I_4 thoroughly but not weakly refines I_5

Theorem 1. *The thorough refinement is strictly weaker than the weak refinement, which is strictly weaker than the strong refinement:* $(\leq_{\mathsf{T}}) \supsetneq (\leq_{\mathsf{W}}) \supsetneq (\leq_{\mathsf{S}})$.

The first inequality is shown by exhibiting IMCs I_4 and I_5 such that I_4 thoroughly but not weakly refines I_5 (Figure 3). All implementations of I_4 satisfy I_5, but state B cannot refine any of β_1 or β_2: Let σ be a distribution admitted in B giving probability 1 to state C. Because of the interval $[0, 0.5]$ on the transition from β_1 to δ_1, at least 0.5 must be assigned to γ_1, but C and γ_1 cannot be related. A similar argument shows that B cannot refine β_2. The second inequality is shown by demonstrating two other IMCs, I_3 and I_2 such that I_3 weakly but not strongly refines I_2 (Figure 2b). State A weakly refines state α: Given a value x for the transition $A \to C$, we can split it in order to match both transitions $\alpha \xrightarrow{p \cdot x} \delta_1$ and $\alpha \xrightarrow{(1-p) \cdot x} \delta_2$. Define $\delta(C)(\delta_1) = p$ and $\delta(C)(\delta_2) = (1 - p)$, with $p = 0$ if $0.2 \leq x \leq 0.4$, $p = \frac{x-0.3}{x}$ if $0.4 < x < 0.8$, and $p = 0.6$ if $0.8 \leq x$. The correspondence function δ witnesses weak refinement between A and α. However, there is no such value of p that would work uniformly for all x, which is required by the strong refinement.

Deciding Thorough Refinement. As weak and strong refinements are strictly stronger than thorough refinement, it is interesting to investigate complexity of deciding TR. In [10] a procedure computing TR is given, albeit without a complexity class, which we establish now, closing the problem:

Theorem 2. *The decision problem TR of establishing whether there exists a thorough refinement between two given IMCs is EXPTIME-complete.*

The upper-bound in checking whether I_1 thoroughly refines I_2 is shown by observing that the complexity of the subset-simulation algorithm of [10] is $O(|Q| \cdot 2^{|P|})$, where Q and P are the set of states of I_1 and I_2, respectively (see [6]).

Summarizing, all three refinements are in EXPTIME. Still, weak refinement seems easier to check than thorough refinement. For TR, the number of iterations on the state-space of the relation is exponential while it is only polynomial for the weak refinement. Also, the constraint solved at each iteration involves a single quantifier alternation for the weak, and three alternations for the thorough refinement.

(a) A MTS M (b) The IMC $\widehat{M}$

Fig. 4. An example of the translation from Modal Transition Systems to IMCs

The lower bound of Theorem 2 is shown by a polynomial reduction of the thorough refinement problem for modal transition systems to TR of IMCs. The former problem is known to be EXPTIME-complete [3].

A modal transition system (an MTS in short) [15] is a tuple $M = (S, s_0, A, \rightarrow, \dashrightarrow)$, where S is the set of states, s_0 is the initial state, and $\rightarrow \subseteq S \times A \times S$ are the transitions that *must* be taken and $\dashrightarrow \subseteq S \times A \times S$ are the transitions that *may* be taken. In addition, it is assumed that $(\rightarrow) \subseteq (\dashrightarrow)$. An implementation of an MTS is a labelled transition system, i.e., an MTS where $(\rightarrow) = (\dashrightarrow)$. Formal definitions of refinement and satisfaction for MTSs are given in [6].

We describe here a translation of MTSs into IMCs which preserves implementations, while we delegate the technicalities of the proof to [6]. We assume we only work with modal transition systems that have no deadlock-states, in the sense that each state has at least one outgoing must transition. It is easy to transform two arbitrary MTSs into deadlock-free ones without affecting the thorough refinement between them [6].

The IMC $\widehat{M}$ corresponding to a MTS $M = (S, s_0, A, \rightarrow, \dashrightarrow)$ is defined by the tuple $\widehat{M} = \langle Q, q_0, A \cup \{\epsilon\}, \varphi, V \rangle$ where $Q = S \times (\{\epsilon\} \cup A)$, $q_0 = (s_0, \epsilon)$, for all $(s, x) \in Q$, $V((s, x)) = \{x\}$ and φ is defined as follows: for all $t, s \in S$ and $b, a \in (\{\epsilon\} \cup A)$, $\varphi((t, b))((s, a)) =]0, 1]$ if $t \xrightarrow{a} s$; $\varphi((t, b))((s, a)) = [0, 0]$ if $t \not\dashrightarrow^{a} s$; and $\varphi((t, b))((s, a)) = [0, 1]$ otherwise. The encoding is illustrated in Figure 4.

Now one can show that $I \models M$ iff $[\![\widehat{I}]\!] \subseteq [\![\widehat{M}]\!]$, and use this to show that the reduction preserves thorough refinement. This observation, which shows how deep is the link between IMCs and modal transition systems, is formalized in the following theorem lifting the syntactic reduction to the level of extensional semantics:

Theorem 3. *Let M and M' be two Modal Transition Systems and $\widehat{M}$ and $\widehat{M'}$ be the corresponding IMCs defined as above. We have*

$$M \leq_{\mathsf{T}} M' \iff \widehat{M} \leq_{\mathsf{T}} \widehat{M'}$$

Crucially the translation is polynomial. Thus if we had a subexponential algorithm for TR of IMCs, we could use it to obtain a subexponential algorithm for TR of MTSs, which is impossible [3].

4 Determinism

Although both are in EXPTIME, deciding weak refinement is easier than deciding thorough refinement. Nevertheless, since these two refinements do not coincide in general, a procedure to check weak refinement cannot be used to decide thorough refinement.

Observe that weak refinement has a syntactic definition very much like simulation for transition systems. On the other hand thorough refinement is a semantic concept, just as trace inclusion for transition systems. It is well known that simulation and trace inclusion coincide for deterministic automata. Similarly for MTSs it is known that TR coincides with modal refinement for deterministic objects. It is thus natural to define deterministic IMCs and check whether thorough and weak refinements coincide on these objects.

In our context, an IMC is deterministic if, from a given state, one cannot reach two states that share common atomic propositions.

Definition 5 (Determinism). *An IMC* $I = \langle Q, q_0, \varphi, A, V \rangle$ *is* deterministic *iff for all states* $q, r, s \in Q$*, if there exists a distribution* $\sigma \in \varphi(q)$ *such that* $\sigma(r) > 0$ *and* $\sigma(s) > 0$*, then* $V(r) \neq V(s)$*.*

Determinism ensures that two states reachable *with the same admissible distribution* always have different valuations. In a semantic interpretation this means that there exists no implementation of I, in which two states with the same valuation can be successors of the same source state. Another, slightly more syntactic but semantically equivalent notion of determinism is given in [6].

It is worth mentioning that deterministic IMCs are a strict subclass of IMCs. Figure 5 shows an IMC I whose set of implementations cannot be represented by a deterministic IMC.

We now state the main theorem of the section that shows that for deterministic IMCs, the weak refinement, and indeed also the strong refinement, correctly capture the thorough refinement:

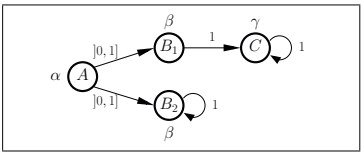


Fig. 5. An IMC I whose implementations cannot be captured by a deterministic IMC

Theorem 4. *Given two deterministic IMCs* I *and* I' *with no inconsistent states, it holds that* $I \leq_{\mathsf{T}} I'$ *iff* $I \leq_{\mathsf{W}} I'$ *iff* $I \leq_{\mathsf{S}} I'$*.*

5 Common Implementation and Consistency

We now turn our attention to the problem of implementation of several IMC specifications by the same probabilistic system modeled as a Markov Chain. We start with a formal definition of the problem:

Definition 6 (Common Implementation (CI)). *Given* $k > 1$ *IMCs* I_i*,* $i = 1 \ldots k$*, does there exist a Markov Chain* C *such that* $C \models I_i$ *for all* i*?*

Somewhat surprisingly we find out that, similarly to the case of TR, the CI problem is not harder for IMCs than for modal transition systems:

Theorem 5. *Deciding the existence of a CI between k IMCs is EXPTIME-complete.*

We sketch the line of argument below, delegating to [6] for details. To establish a lower bound for CI of IMCs, we reduce from CI of modal transition systems, which is known to be EXPTIME-complete [2]. For a set of modal transition systems M_i, $i = 1 \dots k$, translate each M_i, into an IMC $\widehat{M_i}$, using the same rules as in Section 3. It turns out that the set of created IMCs has a common implementation if and only if the original modal transition systems had. Since the translation is polynomial, the problem of CI for IMCs has to be at least EXPTIME-hard (otherwise it would give a sub-EXPTIME algorithm for CI of MTSs).

To address the upper bound we first propose a simple construction to check if there exists a CI for two IMCs. We start with the definition of *consistency relation* that witnesses a common implementation between two IMCs.

Definition 7. *Let $I_1 = \langle Q_1, q_0^1, \varphi_1, A, V_1 \rangle$ and $I_2 = \langle Q_2, q_0^2, \varphi_2, A, V_2 \rangle$ be IMCs. The relation $\mathcal{R} \subseteq Q_1 \times Q_2$ is a* consistency relation *on the states of I_1 and I_2 iff, whenever $(u, v) \in \mathcal{R}$, then*

- *$V_1(u) = V_2(v)$ and*
- *there exists a distribution $\rho \in \mathsf{Distr}(Q_1 \times Q_2)$ such that*
  1. *$\forall u' \in Q_1 : \sum_{v' \in Q_2} \rho(u', v') \in \varphi_1(u)(u') \ \wedge \ \forall v' \in Q_2 : \sum_{u' \in Q_1} \rho(u', v') \in \varphi_2(v)(v')$, and*
  2. *$\forall (u', v') \in Q_1 \times Q_2$, if $\rho(u', v') > 0$, then $(u', v') \in \mathcal{R}$.*

It can be shown that two IMCs indeed have a common implementation if and only if there exists a consistency relation containing their initial states. The consistency relation can be computed in polynomial time using a standard coinductive fixpoint iteration, where pairs violating Definition 7 are successively removed from $Q_1 \times Q_2$. Each iteration requires solving a polynomial number of linear systems, which can be done in polynomial time [14]. For the general problem of common implementation of k IMCs, we can extend the above definition of consistency relation to the k-ary relation in the obvious way, and the algorithm becomes exponential in the number of IMCs k, as the size of the state space $\prod_{i=1}^{k} |Q_i|$ is exponential in k.

As a side effect we observe that, exactly like MTSs, CI becomes polynomial for any constant value of k, i.e. when the number of components to be checked is bounded by a constant.

Consistency. A related problem is the one of checking consistency of a single IMC I, i.e. whether there exists a Markov chain M such that $M \models I$.

Definition 8 (Consistency (C)). *Given an IMC I, does it hold that $[\![I]\!] \neq \emptyset$?*

It turns out that, in the complexity theoretic sense, this problem is easy:

Theorem 6. *The problem C, to decide if a single IMC is consistent, is polynomial time solvable.*

Given an IMC $I = \langle Q, q_0, \varphi, A, V \rangle$, this problem can be solved by constructing a consistency relation over $Q \times Q$ (as if searching for a common implementation of I with itself). There exists an implementation of I iff there exists a consistency relation containing (q_0, q_0). Obviously, this can be checked in polynomial time.

The fact that C can be decided in polynomial time casts an interesting light on the ability of IMCs to express inconsistency. On one hand, one can clearly specify inconsistent states in IMCs (simply by giving intervals for successor probabilities that cannot be satisfied by any distribution). On the other hand, this inconsistency appears to be local. It does not induce any global constraints on implementations; it does not affect consistency of other states. In this sense IMCs resemble modal transition systems (which at all disallow expressing inconsistency), and are weaker than *mixed transition systems* [5]. Mixed transition systems relax the requirement of modal transition systems, not requiring that $(\rightarrow) \subseteq (\dashrightarrow)$. It is known that C is trivial for modal transition systems, but EXPTIME-complete for mixed transition systems [2]. Clearly, with a polynomial time C, IMCs cannot possibly express global behaviour inconsistencies in the style of mixed transition systems, where the problem is much harder.

We conclude the section by observing that, given the IMC I and a consistency relation $\mathcal{R} \subseteq Q \times Q$, it is possible to derive a *pruned* IMC $I^* = \langle Q^*, q_0^*, \varphi^*, A, V^* \rangle$ that contains no inconsistent states and accepts the same set of implementations as I. The construction of I^* is as follows: $Q^* = \{q \in Q | (q, q) \in \mathcal{R}\}$, $q_0^* = q_0$, $V^*(q^*) = V(q^*)$ for all $q^* \in Q^*$, and for all $q_1^*, q_2^* \in Q^*$, $\varphi^*(q_1^*)(q_2^*) = \varphi(q_1^*)(q_2^*)$.

6 Related Work and Conclusion

This paper provides new results for IMCs [10] that is a specification formalism for probabilistic systems. We have studied the expressiveness and complexity of three refinement preorders for IMCs. The results are of interest as existing articles on IMCs often use one of these preorders to compare specifications (for abstraction) [10,12,7]. We have established complexity bounds and decision procedures for these relations, first introduced in [10]. Finally, we have studied the common implementation problem. Our solution is constructive in the sense that it can build such a common implementation.

There exist many other specification formalisms for describing and analyzing stochastic systems; the list includes process algebras [1,16] or logical frameworks, [8]. We believe that IMCs is a good unification model. A logical representation is suited for conjunction, but nor for refinement and vice-versa for process algebra. As an example, it is not clear how one can synthesize a MC (an implementation) that satisfies two Probabilistic Computation Tree Logic formulas.

In [12,13], Katoen et al. have proposed an extension of IMCs to the continuous timed setting. It would be interesting to see our results extend to this new model.

References

[1] Andova, S.: Process algebra with probabilistic choice. In: Katoen, J.-P. (ed.) AMAST-ARTS 1999, ARTS 1999, and AMAST-WS 1999. LNCS, vol. 1601, pp. 111–129. Springer, Heidelberg (1999)

[2] Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wąsowski, A.: Modal and mixed specifications: key decision problems and their complexities. MSC 20(01), 75–103 (2010)

[3] Beneš, N., Křetínský, J., Larsen, K.G., Srba, J.: Checking thorough refinement on modal transition systems is exptime-complete. In: Leucker, M., Morgan, C. (eds.) ICTAC 2009. LNCS, vol. 5684, pp. 112–126. Springer, Heidelberg (2009)

[4] Caillaud, B., Delahaye, B., Larsen, K.G., Legay, A., Pedersen, M.L., Wąsowski, A.: Compositional design methodology with constraint markov chains. In: QEST. IEEE Computer Society, Los Alamitos (2010)

[5] Dams, D.: Abstract Interpretation and Partition Refinement for Model Checking. PhD thesis, Eindhoven University of Technology (July 1996)

[6] Delahaye, B., Larsen, K.G., Legay, A., Pedersen, M.L., Wąsowski, A.: Decision problems for interval markov chains (2011), http://www.irisa.fr/s4/people/benoit.delahaye/rapports/LATA11-long.pdf

[7] Fecher, H., Leucker, M., Wolf, V.: Don't Know in probabilistic systems. In: Valmari, A. (ed.) SPIN 2006. LNCS, vol. 3925, pp. 71–88. Springer, Heidelberg (2006)

[8] Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Asp. Comput. 6(5), 512–535 (1994)

[9] Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: Proc. FOCS 1995, pp. 453–462 (1995)

[10] Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: LICS, pp. 266–277. IEEE Computer, Los Alamitos (1991)

[11] Jonsson, B., Larsen, K.G., Yi, W.: Probabilistic extensions of process algebras. In: Handbook of Process Algebra, pp. 685–710. Elsevier, Amsterdam (2001)

[12] Katoen, J., Klink, D., Leucker, M., Wolf, V.: Three-valued abstraction for continuous-time Markov chains. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 311–324. Springer, Heidelberg (2007)

[13] Katoen, J., Klink, D., Neuhäußer, M.R.: Compositional abstraction for stochastic systems. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 195–211. Springer, Heidelberg (2009)

[14] Khachiyan, L.G.: A polynomial algorithm in linear programming. Dokl. Akad. Nauk SSSR 244(5), 1093–1096 (1979)

[15] Larsen, K.G.: Modal specifications. In: Sifakis, J. (ed.) AVMS 1989. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1990)

[16] López, N., Núñez, M.: An overview of probabilistic process algebras and their equivalences. In: Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P., Siegle, M. (eds.) Validation of Stochastic Systems. LNCS, vol. 2925, pp. 89–123. Springer, Heidelberg (2004)

Classifying Regular Languages via Cascade Products of Automata

Marcus Gelderie

RWTH Aachen, Lehrstuhl für Informatik 7, D-52056 Aachen, Germany
gelderie@automata.rwth-aachen.de

Abstract. Building on the celebrated Krohn-Rhodes Theorem we characterize classes of regular languages in terms of the cascade decompositions of minimal DFA of languages in those classes. More precisely we provide characterizations for the classes of piecewise testable languages and commutative languages. To this end we use biased resets, which are resets in the classical sense, that can change their state at most once. Next, we introduce the concept of the scope of a cascade product of reset automata in order to capture a notion of locality inside a cascade product and show that there exist constant bounds on the scope for certain classes of languages. Finally we investigate the impact of biased resets in a product of resets on the dot-depth of languages recognized by this product. This investigation allows us to refine an upper bound on the dot-depth of a language, given by Cohen and Brzozowski.

1 Introduction

A significant result in the structure theory of regular languages is the Krohn-Rhodes Theorem [7], which states that any finite automaton can be decomposed into simple "prime factors" (a detailed exposition is given in [4,6,9,10]).

We use the Krohn-Rhodes Theorem to characterize classes of regular languages in terms of the decompositions of the corresponding minimal automata. In [8] this has been done for star-free languages by giving an alternative proof for the famous Schützenberger Theorem [11]. In [1] $\mathcal{R}$-trivial languages are characterized (among other things) by proving structural properties of the cascade products covering their minimal automata. We continue these studies, in an attempt to improve our understanding of the potential of automata decompositions for classifying regular languages, an approach which is as yet not well developed in comparison to the structure theory of regular languages based on algebraic methods (wreath product and block product decomposition of monoids, see [14]).

We treat here the case of piecewise testable and commutative languages, as well as the star-free languages in their classification by the dot-depth hierarchy. To this end, we use the concept of a *biased reset* (called a *half reset* in [1]) and introduce *locally i-triggered cascade products* in order to characterize piecewise testable languages. For commutative languages we use the notion of one letter automaton (OLA) and a corresponding one letter cascade product. We show that a language is commutative iff its minimal automaton is covered by a direct

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 286–297, 2011.

product of a one letter cascade product of biased resets and one letter simple cyclic grouplike automata.

Next we introduce the notion of *scope* of resets within a cascade product in order to further refine our analysis of the Krohn-Rhodes decomposition. Informally speaking, the scope of a product of resets is the maximal number of preceding automata, to which any given factor is still sensitive. The scope measures a notion of locality in the product. As initial results we show that the scope of cascade products recognizing $\mathcal{R}$-trivial languages is bounded by 2 and that for piecewise testable languages, it is bounded by 1.

Finally we pick up a result from Cohen and Brzozowski [2], which bounds the dot-depth of a star-free language by the number of factors of a cascade product of resets recognizing it. We show that this result can be refined by counting blocks of biased resets as a single reset. To this end we show that multiplying (w.r.t. the cascade product) an arbitrary number of biased resets to an automaton $\mathfrak{A}$ increases the dot-depth of languages recognized by the product by at most one (compared to the dot-depth of languages already recognized by $\mathfrak{A}$).

The present paper is based on the diploma thesis [5].

2 Preliminaries

A *semiautomaton* is a tuple $\mathfrak{A} = (Q, \Gamma, \delta^{\mathfrak{A}})$, where Q is a finite set of *states*, Γ is a finite set of *letters*, called the *input alphabet of* $\mathfrak{A}$ and $\delta^{\mathfrak{A}} : \Gamma \to Q^Q$ is the *state transition function* assigning a mapping $\overline{a}^{\mathfrak{A}} : Q \to Q$ to each letter $a \in \Gamma$. By function composition we can extend these mappings to words $w = a_1 \cdots a_n \in \Gamma^*$ by setting $\overline{w}^{\mathfrak{A}}(q) = \overline{a_n}^{\mathfrak{A}}(\overline{a_{n-1}}^{\mathfrak{A}}(\cdots \overline{a_1}^{\mathfrak{A}}(q) \cdots))$[1]. A *subsemiautomaton* of $\mathfrak{A}$ is a structure $\mathfrak{A}_0 = (Q_0, \Gamma, \delta_0^{\mathfrak{A}})$, where $Q_0 \subseteq Q$ is closed under the mappings $\delta^{\mathfrak{A}}(a)$ for all $a \in \Gamma$ and $\delta_0^{\mathfrak{A}}(a)$ is the restriction of $\delta^{\mathfrak{A}}(a)$ to Q_0, $a \in \Gamma$. A *homomorphism* from $\mathfrak{A} = (Q, \Gamma, \delta^{\mathfrak{A}})$ to $\mathfrak{B} = (P, \Gamma, \delta^{\mathfrak{B}})$ is a mapping $\varphi : Q \to P$ with $\varphi(\overline{a}^{\mathfrak{A}}(q)) = \overline{a}^{\mathfrak{B}}(\varphi(q))$ for all $q \in Q$ and $a \in \Gamma$. $\mathfrak{A}$ *covers* a semiautomaton $\mathfrak{B}$ (of the same input alphabet), written $\mathfrak{B} \leq \mathfrak{A}$, if $\mathfrak{B}$ is the image of a subsemiautomaton of $\mathfrak{A}$ under some homomorphism φ.

A *deterministic finite automaton (DFA)* is a semiautomaton $\mathfrak{A} = (Q, \Gamma, \delta)$ with a designated *initial state* $q_0 \in Q$ and a set $F \subseteq Q$ of *final states*. In this situation we sometimes write $(\mathfrak{A}, q_0, F)$ or (using the same symbol for the DFA and the corresponding semiautomaton) $\mathfrak{A} = (Q, \Gamma, q_0, \delta, F)$. If $\mathfrak{A}$ is a DFA, then the *language accepted by* $\mathfrak{A}$ is denoted by $L(\mathfrak{A}) = \{w \in \Gamma^* | \overline{w}^{\mathfrak{A}}(q_0) \in F\}$. A single semiautomaton constitutes the foundation of several DFA. Hence, a semiautomaton "recognizes" a set of languages: $\mathcal{L}(\mathfrak{A}) = \{L \subseteq \Gamma^* | \exists q_0 \in Q, F \subseteq Q : L = L(\mathfrak{A}, q_0, F)\}$. If $L \in \mathcal{L}(\mathfrak{A})$ we say L is *recognized* by $\mathfrak{A}$. Given a regular language $L \subseteq \Gamma^*$ we denote the *canonical DFA for* L by $\mathfrak{A}_L$. As noted above, we often identify a DFA with the underlying semiautomaton. This identification is used in the following proposition, the proof of which is left to the reader:

Proposition 1. *Given a semiautomaton* $\mathfrak{A}$ *and a regular language* L*, we have* $L \in \mathcal{L}(\mathfrak{A})$ *iff* $\mathfrak{A}_L \leq \mathfrak{A}$.

[1] Note that the empty word induces the identity mapping.

In what follows we will often deal with two kinds of semiautomata: *resets* and *permutation* automata. A reset automaton[2] is a semiautomaton $\mathfrak{R} = (\{0,1\}, \Gamma, \delta^{\mathfrak{R}})$, where for any input $a \in \Gamma$ the induced mapping $\overline{a}^{\mathfrak{R}}$ is either the identity on $\mathbb{B} := \{0,1\}$ or has constant value $x \in \mathbb{B}$. Conversely, a permutation automaton is an automaton $\mathfrak{P} = (Q, \Gamma, \delta^{\mathfrak{P}})$, such that every input $a \in \Gamma$ induces a permutation (that is, a bijective function).

Recall that a monoid M *divides* a monoid N, written $M \prec N$, if there exists a surjective monoid homomorphism $\psi : N_0 \to M$ from a submonoid N_0 of N onto M. As with coverings of automata, the division relation is transitive and reflexive. We recall that the transition monoid of an automaton is the set $M(\mathfrak{A})$ of all mappings $\overline{w}^{\mathfrak{A}} : Q \to Q$ for $w \in \Gamma^*$. In the special case where $\mathfrak{A}$ is a permutation automaton, the monoid $M(\mathfrak{A})$ is a group G. If $\mathfrak{A}$ has precisely $|G|$ states we call $\mathfrak{A}$ a *grouplike automaton*. Notice that in this case, we may identify the states from Q with elements from G. G is the group *associated* with $\mathfrak{A}$. A grouplike automaton $\mathfrak{G}$ is simple (cyclic) if the associated group is simple (cyclic). Since the number of states is equal to $|G|$ it makes sense to speak of the *order of* $\mathfrak{G}$, which we define to be the order of G.

Given two automata $\mathfrak{A} = (Q, \Gamma, \delta^{\mathfrak{A}})$ and $\mathfrak{B} = (P, \Gamma \times Q, \delta^{\mathfrak{B}})$ we define

$$\begin{aligned} \delta^{\mathfrak{A}*\mathfrak{B}}(a) := \overline{a}^{\mathfrak{A}*\mathfrak{B}} &: Q \times P \to Q \times P \\ (q,p) &\mapsto (\overline{a}^{\mathfrak{A}}(q), \overline{(a,q)}^{\mathfrak{B}}(p)) \end{aligned}$$

The automaton $\mathfrak{A} * \mathfrak{B} = (Q \times P, \Gamma, \delta^{\mathfrak{A}*\mathfrak{B}})$ is called the *cascade product of* $\mathfrak{A}$ *and* $\mathfrak{B}$. We recall a few important properties of the cascade product:

Theorem 1 (see [6]). *Let $\mathfrak{A}$, $\mathfrak{B}$ and $\mathfrak{C}$ be semiautomata with input alphabets of the suitable format. Then the following hold:*

(1) $(\mathfrak{A} * \mathfrak{B}) * \mathfrak{C} = \mathfrak{A} * (\mathfrak{B} * \mathfrak{C})$
(2) If $\mathfrak{A} \leq \mathfrak{B}$ *then* $\mathfrak{C} * \mathfrak{A} \leq \mathfrak{C} * \mathfrak{B}$

The following theorem is the basis for our task of characterizing language classes:

Theorem 2 (Krohn, Rhodes, [7]). *Let $\mathfrak{A}$ be a semiautomaton. Then*

$$\mathfrak{A} \leq \mathfrak{F}_1 * \cdots * \mathfrak{F}_n$$

for semiautomata $\mathfrak{F}_i$, such that each $\mathfrak{F}_i$ is either a reset or a simple grouplike automaton with $M(\mathfrak{F}_i) \prec M(\mathfrak{A})$.

A detailed exposition of the Krohn-Rhodes Theorem is given in [3,4,6]. We will call a decomposition of an automaton $\mathfrak{A}$, which is of the form stated in Theorem 2, a *Krohn-Rhodes decomposition (of* $\mathfrak{A}$*)*.

[2] Reset automata are often introduced in a more general fashion, allowing for an arbitrary number of states. However, if a reset automaton has more than 2 states, it can be covered by a direct product of 2-state reset automata (see, for instance, [6]).

3 Piecewise Testable Languages

In this section we want to characterize piecewise testable languages in terms of their cascade products. Recall that a language is *piecewise testable* iff it is a Boolean combination of expressions of the form $\Gamma^* a_1 \Gamma^* \cdots \Gamma^* a_n \Gamma^*$ for letters $a_1, \ldots, a_n \in \Gamma$, $n \in \mathbb{N}_0$ (if $n = 0$ we obtain Γ^*). There are several characterizations for piecewise testable languages (see for instance [1,9,10,12,15,13]). It should be mentioned that the characterization from [13] (stated in terms of matrices over a semiring) is very similar to the one we give below, yet not stated in terms of semidirect products. This result is again mentioned in [15] in the context of a discussion of restricted semidirect products. The decomposition obtained thereby is indeed very close the one we give below. As we give a purely automaton theoretic proof (the proof from [15] is purely algebraic) it would be interesting to investigate whether one can be obtained from the other.

Definition 1. *Let $\mathfrak{R}_1 * \cdots * \mathfrak{R}_n$ be a cascade product of resets.*

(a) We say the reset $\mathfrak{R}_i$ has scope k *if for every pair of inputs $(a, (x_1, \ldots, x_{i-1}))$ and $(a, (y_1, \ldots, y_{i-1}))$ with $y_j = x_j$ for all $i - k \leq j \leq i - 1$ the induced mappings are equal, i.e.*

$$\overline{(a, (x_1, \ldots, x_{i-1}))}^{\mathfrak{R}_i} = \overline{(a, (y_1, \ldots, y_{i-1}))}^{\mathfrak{R}_i}$$

(b) The scope of $\mathfrak{R}_i$ *is the minimal number k, such that $\mathfrak{R}_i$ has scope k. The scope of the product $\mathfrak{R}_1 * \cdots * \mathfrak{R}_n$ is the maximal scope of a reset $\mathfrak{R}_i$.*

(c) If $b \in \mathbb{B}$, we call the product above strictly locally b-triggered*, if it has scope 1 and for every reset $\mathfrak{R}_i$ every input of the form $(a, (x_1, \ldots, x_{i-2}, 1 - b))$ induces the identity mapping. A product is called* locally b-triggered*, if it is a direct product of strictly locally b-triggered products.*

In other words, the scope of a reset counts the number of resets preceding it in a cascade, such that it is sensitive to the state of these resets. For example, a cascade product degenerates to a direct product iff it has scope 0. Strictly locally b-triggered products add another constraint to this, namely that the reset may only alter its state if the immediately preceding reset is in state b. We will return to the scope of a cascade product in Sec. 5.

A reset $\mathfrak{R} = (\mathbb{B}, \Gamma, \delta)$ is *b-biased*, where $b \in \mathbb{B}$, if for every input $a \in \Gamma$ we have $\overline{a}^{\mathfrak{R}} = \mathrm{id}_{\mathbb{B}}$ or for all $q \in \mathbb{B}$ we have $\overline{a}^{\mathfrak{R}}(q) = b$. In the second case, we write $\overline{a}^{\mathfrak{R}} \equiv b$. A reset $\mathfrak{R}$ is *biased* if it is b-biased for some b. We can now state:

Theorem 3. *A language L is piecewise testable iff it is recognized by a locally b-triggered cascade product of b-biased resets.*

Notice that this yields a Krohn-Rhodes decomposition of $\mathfrak{A}_L$ by Proposition 1. We will dedicate the remainder of this section to proving this result. To this end we first verify that all piecewise testable languages are recognized by a locally b-triggered cascade product of b-biased resets. Without loss of generality we will henceforth assume $b = 1$ (otherwise we rename the states).

By the definition of piecewise testable languages above it is sufficient to show that every language of the form $L = \Gamma^* a_1 \Gamma^* \cdots \Gamma^* a_n \Gamma^*$ is recognized by such a product: Boolean combinations can be recognized by direct products and a direct product of locally 1-triggered products is again locally 1-triggered. Given L, we define n 1-biased resets $\mathfrak{R}_1, \ldots, \mathfrak{R}_n$ by setting[3]:

$$\overline{(a,(x_1,\ldots,x_{i-1}))}^{\mathfrak{R}_i}(b) = \begin{cases} 1 \text{ if } x_{i-1} = 1 \wedge a = a_i \\ b \text{ otherwise} \end{cases}$$

for all $b \in \mathbb{B}$, $a \in \Gamma$ and $(x_1, \ldots, x_{i-1}) \in \mathbb{B}^{i-1}$. Then every $\mathfrak{R}_i$ is 1-biased and $\mathfrak{R}_1 * \cdots * \mathfrak{R}_n$ is strictly locally 1-triggered and evidently accepts L with initial state $q_0 = (0, \ldots, 0)$ and final states $\{(1, \ldots, 1)\}$.

Before embarking on showing the converse, we need a bit of preparation:

Remark 1. Let $\mathfrak{R} = (\mathbb{B}, \Gamma, \delta)$ be a 1-biased reset and let $\Gamma_1 = \{a \in \Gamma | \overline{a}^{\mathfrak{R}} \equiv 1\}$. Then[4] $\mathcal{L}(\mathfrak{R}) = \{\emptyset, \Gamma^*, \bigcup_{a\in\Gamma_1} \Gamma^* a \Gamma^*, \bigcap_{a\in\Gamma_1} \overline{\Gamma^* a \Gamma^*}\}$. In particular all languages from $\mathcal{L}(\mathfrak{R})$ are piecewise testable.

We now investigate the languages recognized by cascade products of biased resets:

Lemma 1. *Let $\mathfrak{A} = (Q, \Gamma, \delta^{\mathfrak{A}})$ be a semiautomaton and let $\mathfrak{R} = (\mathbb{B}, \Gamma \times Q, \delta^{\mathfrak{R}})$ be a 1-biased reset. Then every language recognized by $\mathfrak{A} * \mathfrak{R}$ is a finite union of languages of one of the following three forms (for suitable $p_0, p_F \in Q$):*

(1) $L \in \mathcal{L}(\mathfrak{A})$

(2) $L = \bigcup_{(a,q)\in S_1} L(\mathfrak{A}, p_0, \{q\}) \cdot a \cdot L(\mathfrak{A}, \overline{a}^{\mathfrak{A}}(q), \{p_F\})$

(3) $L = \left(\overline{\bigcup_{(a,q)\in S_1} L(\mathfrak{A}, p_0, \{q\}) \cdot a \cdot \Gamma^*}\right) \cap L(\mathfrak{A}, p_0, \{p_F\})$

where $S_1 = \{(a, q) \in \Gamma \times Q \mid \overline{(a,q)}^{\mathfrak{R}} \equiv 1\}$.

The proof of this lemma is not difficult, but omitted due to space constraints. The reader is referred to [5].

Lemma 2. *If $\mathfrak{A} := \mathfrak{R}_1 * \cdots * \mathfrak{R}_n$ is a strictly locally 1-triggered cascade product of 1-biased resets $\mathfrak{R}_i = (\mathbb{B}, \Gamma \times \mathbb{B}^{i-1}, \delta^{\mathfrak{R}_i})$, $i = 1, \ldots, n$, then every language recognized by $\mathfrak{A}$ with accepting states $\{(x_1, \ldots, x_n) | x_n = 1\} \subseteq \mathbb{B}^n$ is a finite union of languages of the form $\Gamma^* a_1 \Gamma^* \cdots \Gamma^* a_m \Gamma^*$, $m \in \mathbb{N}_0$.*

The proof of this lemma is by induction on n and uses Lemma 1 and Remark 1. For details the reader is again referred to [5].

This enables us to complete the proof of Theorem 3. Let $\mathfrak{A}$ be a locally 1-triggered cascade product of 1-biased resets. Then $\mathfrak{A}$ is a direct product of strictly

[3] We use the convention $\Gamma \times \mathbb{B}^0 = \Gamma$ and accordingly $(a, (x_1, \ldots, x_{i-1})) = a$ if $i = 1$.

[4] Given a language $L \subseteq \Gamma^*$ we denote the complement language by $\overline{L} = \Gamma^* \setminus L$.

locally 1-triggered cascade products of 1-biased resets. Therefore all languages from $\mathcal{L}(\mathfrak{A})$ are Boolean combinations of the languages recognized by strictly locally 1-triggered products. Since the piecewise testable languages form a Boolean algebra, it is sufficient to show that all strictly locally 1-triggered cascade products of 1-biased resets recognize only piecewise testable languages.

Hence assume that $\mathfrak{A} := \mathfrak{R}_1 * \cdots * \mathfrak{R}_n$ is strictly locally 1-triggered. Let $q_0 \in \mathbb{B}^n$ and $F = \{(f_{1,1}, \ldots, f_{1,n}), \ldots, (f_{r,1}, \ldots, f_{r,n})\}$. Since $L(\mathfrak{A}, q_0, F) = \bigcup_{i=1}^{r} L(\mathfrak{A}, q_0, \{(f_{i,1}, \ldots, f_{i,n})\})$ we may assume that $F = \{(f_1, \ldots, f_n)\}$. Suppose $q_0 = (q_{0,1}, \ldots, q_{0,n})$ and let $q_{0,i} = 1$. Then, because $\mathfrak{A}$ is strictly locally 1-triggered, $L = L(\mathfrak{A}, q_0, F)$ is the intersection of the languages $J = L(\mathfrak{R}_1 * \cdots * \mathfrak{R}_i, (q_{0,1}, \ldots, q_{0,i}), \{(f_1, \ldots, f_i)\})$ and $K = L(\tilde{\mathfrak{R}}_{i+1} * \mathfrak{R}_{i+2} * \cdots * \mathfrak{R}_n, (q_{0,i+1}, \ldots, q_{0,n}), \{(f_{i+1}, \ldots, f_n)\})$ where $\tilde{\mathfrak{R}}_{i+1}$ is obtained from $\mathfrak{R}_{i+1}$ by treating all inputs $a \in \Gamma$ as $(a, (0, \ldots, 0, 1))$. Since both resulting products are again strictly locally 1-triggered, we may assume that $q_0 = (0, \ldots, 0)$.

Now assume that $f_n = 1$. Then, since $\mathfrak{A}$ is strictly locally 1-triggered and since all resets are 1-biased, we have $f_1 = \ldots = f_n = 1$ or the language $L = L(\mathfrak{A}, q_0, \{(f_1, \ldots, f_n)\})$ is empty. Hence $L(\mathfrak{A}, q_0, \{(x_1, \ldots, x_n) | x_n = 1\}) = L$, since $(1, \ldots, 1)$ is the only state with $x_n = 1$ reachable from q_0. By Lemma 2 we see that L is a finite union of languages of the form $\Gamma^* a_1 \Gamma^* \cdots \Gamma^* a_r \Gamma^*$.

If $f_n = 0$ we pick $i \in \{1, \ldots, n-1\}$ maximal (if it exists) with $f_i = 1$. If no such i exists, then clearly $L = \overline{L(\mathfrak{R}_1, 0, \{1\})}$, which is piecewise testable. Hence we assume such an index i exists. Since $\mathfrak{A}$ is strictly locally 1-triggered we see that $f_1 = \cdots = f_i = 1$, since otherwise $(f_1, \ldots, f_n)$ is again unreachable from q_0. Furthermore we must have $f_{i+2} = \cdots = f_n = 0$ for the same reason. This implies that $\mathfrak{R}_{i+2}, \ldots, \mathfrak{R}_n$ are irrelevant to the acceptance behavior of $(\mathfrak{A}, q_0, (f_1, \ldots, f_n))$. Thus we may assume that $i = n - 1$.

Using the results from the case $f_n = 1$ we get that $K := L(\mathfrak{R}_1 * \cdots * \mathfrak{R}_{n-1}, (0, \ldots, 0), \{(1, \ldots, 1)\})$ is a finite union of languages $\Gamma^* a_1 \Gamma^* \cdots \Gamma^* a_r \Gamma^*$. Now by Lemma 1 L is a finite union of languages of the form $\overline{K \sigma \Gamma^*} \cap K$. Since the piecewise testable languages are closed under the Boolean operations this concludes the proof.

4 Commutative Languages

In this section we embed the well known results on commutative languages into our framework. We first recall the definition. For $w = a_1 \cdots a_n \in \Gamma^*$ let $\mathrm{Perm}(w) = \{a_{\pi(1)} \cdots a_{\pi(n)} | \pi \in S_n\}$, where S_n denotes the symmetric group on n points. A language L is *commutative* if $\mathrm{Perm}(w) \subseteq L$ for every $w \in L$. This is evidently the case iff $M(L)$ is commutative iff $\mathfrak{A}_L$ is commutative. Recall that a semiautomaton is commutative if $\overline{w}^{\mathfrak{A}}(p) = q$ implies $\overline{v}^{\mathfrak{A}}(p) = q$ for all $v \in \mathrm{Perm}(w)$.

Definition 2. *Let $L \subseteq \Gamma^*$ be regular.*

(a) If there exists $N \subseteq \mathbb{N}_0$ *and* $a_0 \in \Gamma$ *with* $L = \{w \in \Gamma^* \mid |w|_{a_0} \in N\}$*, then* L *is called* 1-semilinear (with respect to a_0)[5,6].

(b) A semiautomaton $\mathfrak{A}$*, such that there exists a letter* $a_0 \in \Gamma$ *with* $\overline{a_0}^{\mathfrak{A}} \neq \mathrm{id}$ *and* $\overline{a}^{\mathfrak{A}} = \mathrm{id}$ *for all* $a_0 \neq a \in \Gamma$ *is called* one letter automaton (OLA) (with respect to a_0).

1-semilinear languages are commutative. Clearly languages accepted by OLA are 1-semilinear. Furthermore 1-semilinear languages yield canonical DFA, which are OLA (for the proof one can use, for instance, Moore's minimization algorithm). A cascade product, such that the automaton it defines is an OLA, is a *one letter cascade product*. We recall that a language is commutative iff it is a Boolean combination of 1-semilinear languages (for details the reader is referred to [10]).

We now turn towards characterizing OLA by their Krohn-Rhodes decompositions. We first observe that minimal OLA (i.e. those that are canonical DFA for some (1-semilinear) language L) have a very simple form. If $\mathfrak{A}_L = (Q, \Gamma, q_0, \delta, F)$ is the minimal DFA for a 1-semilinear language L (with respect to $a \in \Gamma$) then there exist $i < j$ minimal, such that $\overline{w_i}^{\mathfrak{A}_L}(q_0) = \overline{w_j}^{\mathfrak{A}_L}(q_0)$, where $w_k = a^k$ for $k \in \mathbb{N}_0$. Since $\mathfrak{A}_L$ is an OLA w.r.t. a all states from Q occur in the sequence $\overline{w_0}^{\mathfrak{A}_L}(q_0), \overline{w_1}^{\mathfrak{A}_L}(q_0), \ldots, \overline{w_{j-1}}^{\mathfrak{A}_L}(q_0)$ ($\mathfrak{A}_L$ is minimal). Set $q_k := \overline{w_k}^{\mathfrak{A}_L}(q_0)$. Then we obtain two disjoint sets $Q_{tail} = \{q_0, \ldots, q_{i-1}\}$ and $Q_{loop} = \{q_i, \ldots, q_{j-1}\}$.

Lemma 3. *Let* $\mathfrak{A}_L$ *be an OLA. Then* $\mathfrak{A}_L \leq \mathfrak{R}_1 * \cdots * \mathfrak{R}_i \times \mathfrak{C}$*, where* $\mathfrak{R}_1 * \cdots * \mathfrak{R}_i$ *is a one letter cascade product of 1-biased resets and* $\mathfrak{C}$ *is a cyclic one letter grouplike automaton of order* $j - i$ *(where* $i < j$ *are as above).*

Proof. Choose i and j as in the previous paragraph. For $\mathfrak{R}_1 * \cdots * \mathfrak{R}_i$ we pick the product recognizing $(\Gamma^* a \Gamma^*)^i$ as constructed in Sec. 3. We define $\mathfrak{C} = (Q_{loop}, \Gamma, \delta')$ by $\delta'(x) = \overline{x}^{\mathfrak{A}_L}|_{Q_{loop}}$ for $x \in \Gamma$. Pick $r \in Q_{loop}$ such that $\overline{w_i}^{\mathfrak{C}}(r) = q_i$, i.e. r is chosen such that after seeing i a's we end up in the first state of Q_{loop} visited when starting from q_0. Define $A \subseteq \mathbb{B}^i \times Q_{loop}$ by starting out from $(0, \ldots, 0, r)$ and adding all states reachable in $\mathfrak{R}_1 * \cdots * \mathfrak{R}_i \times \mathfrak{C}$. Then A defines a subsemiautomaton. Notice that for $(x_1, \ldots, x_i, q) \in A$ there exists $0 \leq k \leq i$ such that $x_1 = \cdots = x_k = 1$ and $x_{k+1} = \cdots = x_i = 0$. Denote this integer k by $\max(x_1, \ldots, x_i)$. Then we define $\psi : A \to Q$ by $(x_1, \ldots, x_i, q) \mapsto q_{\max(x_1,\ldots,x_i)}$ if $\max(x_1, \ldots, x_i) < i$ and q otherwise. Is is left to the reader to verify that ψ is a surjective homomorphism onto $\mathfrak{A}_L$. □

The grouplike automaton $\mathfrak{C}$ in the lemma above need not be simple. Hence we decompose $\mathfrak{C}$ further in order to arrive at a Krohn-Rhodes decomposition. It is well known that for every finite cyclic group C we have $C \cong \mathbb{Z}/m\mathbb{Z} \cong \times_{i=1}^{s} \mathbb{Z}/m_i\mathbb{Z}$ where $|C| = m = \prod_{i=1}^{s} m_i$ and the m_i are pairwise coprime prime powers. We will give an automaton theoretic equivalent of this fact. Denote by C_n the group $(\mathbb{Z}/n\mathbb{Z}, +)$, $n \in \mathbb{N}$, and denote the n-class of an integer i by $[i]_n$. Returning to

[5] The name results from the fact that the Parikh image of such a language is determined by just one dimension.

[6] Here $|w|_a$ denotes the number of occurrences of the letter $a \in \Gamma$ in w.

the grouplike automaton from the previous lemma, let the order of $\mathfrak{C}$ be m. Define $\mathfrak{H} = (C_m, \Gamma, \delta^{\mathfrak{H}})$ where $\overline{x}^{\mathfrak{H}}([i]_m) = [i+1]_m$ if $x = a$ and $[i]_m$ otherwise, $x \in \Gamma$. Evidently $\mathfrak{C} \cong \mathfrak{H}$ and so in particular $\mathfrak{C} \leq \mathfrak{H}$. Define $\mathfrak{C}_i = (C_{m_i}, \Gamma, \delta^{\mathfrak{C}_i})$ in the same way as $\mathfrak{H}$ (that is for $x \in \Gamma$ let $\overline{x}^{\mathfrak{C}_i}([k]_{m_i}) = [k+1]_{m_i}$ if $x = a$ and $[k]_{m_i}$ otherwise). Then evidently all $\mathfrak{C}_i$ are one letter cyclic grouplike automata and we have $\mathfrak{H} \cong \mathfrak{C}_1 \times \cdots \times \mathfrak{C}_s$. For the proof of this statement, one uses the (group) isomorphism obtained from classical group theory and verifies that it is also a homomorphism of automata. It remains to decompose grouplike one letter automata of the form $\mathfrak{C} = (C_{p^k}, \Gamma, \delta)$ for some prime p and some $k \in \mathbb{N}$.

Lemma 4. *$\mathfrak{C} \leq (C_p, \Gamma, \delta^1) * \cdots * (C_p, \Gamma \times C_p^{k-1}, \delta^k)$, for cyclic grouplike automata $(C_p, \Gamma \times C_p^{i-1}, \delta^i)$, $1 \leq i \leq k$. The cascade product defines an OLA.*

Proof. The proof is by induction on k. If $k = 1$ there is nothing to show. For the induction step we define $\mathfrak{D} = (C_p, \Gamma, \delta^{\mathfrak{D}})$ by $\overline{x}^{\mathfrak{D}}([i]_p) = [i+1]_p$ if $x = a \in \Gamma$ and $[i]_p$ otherwise, $x \in \Gamma$. Then define $\mathfrak{H} = (C_{p^{k-1}}, \Gamma \times C_p, \delta^{\mathfrak{H}})$ by $\overline{(a,[p-1]_p)}^{\mathfrak{H}}([i]_{p^{k-1}}) = [i+1]_{p^{k-1}}$ and $\overline{(x,[r]_p)}^{\mathfrak{H}}([i]_{p^{k-1}}) = [i]_{p^{k-1}}$ for all $(x,[r]_p) \neq (a,[p-1]_p)$. Observe that $\mathfrak{D} * \mathfrak{H}$ is an OLA. By the induction hypothesis and Theorem 1 we are done if we show that $\mathfrak{C} \leq \mathfrak{D} * \mathfrak{H}$. To this end define $\varphi : C_p \times C_{p^{k-1}} \to C_{p^k}$ by $\varphi([i]_p, [j]_{p^{k-1}}) = [(i \mod p) + j \cdot p]_{p^k}$. This mapping is well-defined (as one easily verifies) and is a homomorphism of semiautomata. We only treat the case of the letter a, the case of the remaining letters being trivial. We have $\overline{a}^{\mathfrak{C}}(\varphi([p-1]_p, [j]_{p^{k-1}})) = [(p-1+j\cdot p)+1]_{p^k} = \varphi([0]_p, [j+1]_{p^{k-1}})$. If $0 \leq i < p-1$, then $\overline{a}^{\mathfrak{C}}(\varphi([i]_p, [j]_{p^{k-1}})) = [i+j\cdot p+1]_{p^k} = \varphi([i+1]_p, [j]_{p^{k-1}})$. □

In summary, we have shown:

Theorem 4. *Let $L \subseteq \Gamma^*$. The following are equivalent:*

(1) L is commutative
(2) $\mathfrak{A}_L \leq \left(\times_{i=1}^{k} \mathfrak{R}_{i,1} * \cdots * \mathfrak{R}_{i,n_i}\right) \times \left(\times_{i=1}^{r} \mathfrak{C}_{i,1} * \cdots * \mathfrak{C}_{i,m_i}\right)$, where all cascade products define one letter automata, all resets are biased, all grouplike automata are cyclic of prime order and the orders of $\mathfrak{C}_{i,j}$ and $\mathfrak{C}_{i',j'}$ are equal iff $i = i'$.

5 The Scope of Cascade Products

In Definition 1 we introduced the scope of a cascade product. Notice this definition was only stated for cascade products consisting of resets. We are therefore only dealing with star-free languages. We now want to use this notion to investigate language classes: Given a class $\mathcal{C}$ of star-free languages (e.g. piecewise testable languages, $\mathcal{R}$-trivial languages etc.), what can be said about the scope of a cascade product recognizing the languages from $\mathcal{C}$? If there exists $k \in \mathbb{N}_0$, such that every $L \in \mathcal{C}$ is recognized by a cascade product of scope at most k, then we say $\mathcal{C}$ has *constant scope* or has *scope k*. From Theorem 3 we can deduce:

Proposition 2. *The class of piecewise testable languages has scope 1.*

Recall that a language is $\mathcal{R}$-trivial if its syntactic monoid $M(L)$ is $\mathcal{R}$-trivial, i.e. if $mM(L) = nM(L)$ iff $m = n$ for all $m, n \in M(L)$. We have:

Theorem 5. *The class of $\mathcal{R}$-trivial languages has scope 2.*

The idea of the proof is to decompose an $\mathcal{R}$-trivial language L into a union of left-deterministic products[7] and then to show how to cover the minimal automata for such products by a scope 2 product of biased resets. For a detailed proof, which we omit due to space constraints, the reader is referred to [5].

6 Cascade Products and Dot-Depth

In this section we will make extensive use of first order logic. We will be using formulas from the logic $\mathsf{FO}[\min, \max, (P_a)_{a\in\Gamma}, <, S]$. Such formulas will be interpreted in *word models*: Let $w = b_1 \cdots b_m \in \Gamma^*$. Then the model associated with w is denoted $\underline{w} = (\{1, \ldots, m\}, 1, m, (\{i | b_i = a\})_{a\in\Gamma}, <, S)$ where $S(x, y)$ iff $y = x + 1$ and $<$ is the usual order on natural numbers. If φ is a sentence we write $L(\varphi) = \{w \in \Gamma^* | \ \underline{w} \models \varphi\}$ for the language *specified* or *accepted* by φ. If $\varphi(\bar{x})$ has free variables $\bar{x} = (x_1, \ldots, x_n)$ then we write $(\underline{w}, \bar{k}) \models \varphi(\bar{x})$ for $\bar{k} = (k_1, \ldots, k_n)$ if φ holds in $\underline{w}$ with x_i interpreted by k_i.

As usual we denote by Σ_n the set of FO-formulas, which are equivalent to a formula in prenex normal form with n quantifier alternations beginning with a block of existential quantifiers, e.g. $\exists\bar{x}_1\forall\bar{x}_2 \cdots Q_n\bar{x}_n\varphi(\bar{x}_1, \ldots, \bar{x}_n)$ where φ is quantifier free and Q_n is existential iff n odd. We then define Π_n to be the set of formulas, the negation of which is in Σ_n and we set $\Delta_n = \Sigma_n \cap \Pi_n$. Given a set of formulas Φ we define $\mathsf{BC}(\Phi)$ to be the set of all Boolean combinations of formulas in Φ. Immediately from the definitions one gets $\mathsf{BC}(\Sigma_n) = \mathsf{BC}(\Pi_n) \subseteq \Delta_{n+1}$ and $\mathsf{BC}(\Delta_n) = \Delta_n$ for all $n \in \mathbb{N}$. Also, we recall that disjunctions and conjunction of Σ_n (resp. Π_n) formulas is again a Σ_n (resp. Π_n) formula. Given a set Φ of formulas, we often write $L \in \Phi$ if $L = L(\varphi)$ for some sentence $\varphi \in \Phi$.

We now recall the *dot-depth* of star-free languages. To this end let $\mathcal{B}_0$ be the set of all finite and co-finite languages. For $n \in \mathbb{N}_0$ define $\mathcal{B}_{n+1}$ to be the set of all Boolean combinations of languages $L_1a_1L_2 \cdots L_{n-1}a_{n-1}L_n$ where $L_1, \ldots, L_n \in \mathcal{B}_n$ and $a_1, \ldots, a_n \in \Gamma$. One can show that $\bigcup_{n\in\mathbb{N}_0} \mathcal{B}_n$ is the set of star-free languages[8]. The *dot-depth* of a language L is the number $n \in \mathbb{N}$, such that $L \in \mathcal{B}_n \setminus \mathcal{B}_{n-1}$ (or 0 if $L \in \mathcal{B}_0$). The dot-depth is intimately tied to logic:

Theorem 6 (Thomas, [16]). *$L \in \mathcal{B}_n$ iff $L \in \mathsf{BC}(\Sigma_n)$ for $n \in \mathbb{N}$.*

Notice the theorem makes no statement about level 0. The following theorem is due to Brzozowski and Cohen. In its original formulation it did not make any

[7] Recall a product $\Gamma_0a_1\Gamma_1a_2\Gamma_2 \cdots a_n\Gamma_n$ is *left-deterministic* if for every $i = 1, \ldots, n$ we have $a_i \notin \Gamma_{i-1}$ (see [1,9] for details).

[8] See [2,9,10] for details.

references to logic. We give an alternative proof using logic and can thereby place the languages more precisely within a given level of the hierarchy[9]:

Theorem 7 (Cohen, Brzozowski, [2]). *Let L be recognized by a cascade product $\mathfrak{R}_1 * \cdots * \mathfrak{R}_n$ of n resets. Then $L \in \Delta_{n+1}$ and so has dot-depth at most $n+1$.*

Proof. The proof is by induction on n. The induction start is left to the reader. For the induction step write $\mathfrak{A} = \mathfrak{R}_1 * \cdots * \mathfrak{R}_{n-1} = (Q, \Gamma, \delta^{\mathfrak{A}})$. Then $\mathcal{L}(\mathfrak{A}) \subseteq \Delta_n$ by the induction hypothesis. Let $L(q,b) = L(\mathfrak{A} * \mathfrak{R}_n, (q_0, b_0), \{(q,b)\})$. Then every language in $\mathcal{L}(\mathfrak{A} * \mathfrak{R}_n)$ is a union of languages of this form. It is sufficient to show $L(q,b) \in \Sigma_{n+1}$ for all states (q,b). The claim follows from the fact that $\overline{L(q,b)} \in \Sigma_{n+1}$ as well, hence $L(q,b) \in \Sigma_{n+1} \cap \Pi_{n+1} = \Delta_{n+1}$.

We pick a formula $\varphi_q(x) \in \Delta_n$, such that for $w = a_1 \cdots a_m \in \Gamma^*$ we have $(\underline{w}, k) \models \varphi_q(x)$ iff $a_1 \cdots a_k \in L(q) = L(\mathfrak{A}, q_0, \{q\})$ for $q \in Q$. Denote by $\Gamma^{(i)} \subseteq \Gamma \times Q$ the set of inputs inducing the constant i-mapping in $\mathfrak{R}_n$ for $i \in \mathbb{B}$. Then $L(q,b)$ is specified by $\varphi_q(\max) \wedge \exists x \exists z_1 \forall y \forall z_2 \Big(\bigvee_{(a,q') \in \Gamma^{(b)}} \varphi_{q'}(x) \wedge P_a(z_1) \wedge S(x,z_1) \wedge \Big((S(y,z_2) \wedge y > x) \rightarrow \bigwedge_{(a,q') \in \Gamma^{(1-b)}} (\neg \varphi_{q'}(y) \vee \neg P_a(z_2)) \Big) \Big)$, which is in Σ_{n+1} since $\varphi_q \in \Delta_n$ for all $q \in Q$. □

We will now refine the result from Theorem 7. However, before moving on, we recall from [1] that the state set Q of a cascade product of biased resets is *partially ordered*, i.e. there exists a partial order $\preceq$ on Q, such that for all $a \in \Gamma$ and all $q \in Q$ we have $q \preceq \overline{a}(q)$. We note that we can extend this partial order, to a total order, which is still compatible with the transitions in the way just outlined. We will say the state set is *ordered*.

Lemma 5. *Let $\mathfrak{A} = (Q, \Gamma, \delta^{\mathfrak{A}})$ be a semiautomaton, such that $\mathcal{L}(\mathfrak{A}) \subseteq \Delta_n$ for some $n \in \mathbb{N}$. Then $\mathcal{L}(\mathfrak{A} * \mathfrak{R}_1 * \cdots * \mathfrak{R}_k) \subseteq \Delta_{n+1}$, where $\mathfrak{R}_i$ is a biased reset for $i = 1, \ldots, k$, $k \in \mathbb{N}$.*

Proof. Let $S = \{1, \ldots, r\}$ be the state space of $\mathfrak{O} = \mathfrak{R}_1 * \cdots * \mathfrak{R}_k$. We may assume the compatible order on S to coincides with $\leq$ (the usual order on $\mathbb{N}$). Denote by $\Gamma_{i,j} \subseteq \Gamma \times Q$ the set of inputs which map i to j. Notice that in a run $\mathfrak{O}$ can change its state at most $r-1$ times.

Write $\mathfrak{B} := \mathfrak{A} * \mathfrak{O}$. Let $L = L(\mathfrak{B}, (q_0, i_0), \{(q_F, f)\})$, where $q_0, q_F \in Q$ and $i_0, f \in S$. Then all languages in $\mathcal{L}(\mathfrak{B})$ are unions of languages of this form (and therefore defined by disjunctions of the corresponding formulas). For $q \in Q$ and $w = a_1 \cdots a_m \in \Gamma^*$ let $\varphi_q(x) \in \Delta_n$ be such that $(\underline{w}, t) \models \varphi_q(x)$ iff $a_1 \cdots a_t \in L(\mathfrak{A}, q_0, \{q\})$. Then clearly $L(\mathfrak{A}, q_0, \{q\}) = L(\varphi_q(\max))$. We treat only the case $i_0 \neq f$. The other case requires an adjustment term, which checks the possibility of staying in $i_0 = f$. Define θ to be

[9] The result from [2] yields $L \in \mathcal{B}_{n+1} = \mathsf{BC}(\Sigma_{n+1})$, which is a superset of Δ_{n+1}.

$$\bigvee_{j=1}^{r-1} \exists x_1 \cdots \exists x_j \forall y_0 \cdots \forall y_j \Big(y_0 < x_1 < y_1 \cdots < y_{j-1} < x_j < y_j \wedge \bigvee_{(i_1,\ldots,i_{j-1}) \in S^{j-1}} \Big(\psi_{i_0,i_1}(x_1) \wedge \ldots \wedge \psi_{i_{j-1},f}(x_j) \wedge \bigwedge_{k=0}^{j-1} \bigwedge_{s \neq i_k} \neg \psi_{i_k,s}(y_k) \wedge \bigwedge_{s \neq f} \neg \psi_{f,s}(y_j) \Big) \Big)$$

where $\psi_{t,u}(x) \equiv \big(x = \min \wedge \bigvee_{(a,q_0)\in\Gamma_{t,u}} P_a(\min)\big) \vee \big(x > \min \wedge \exists y(S(y,x) \wedge \bigwedge_{(a,q)\in\Gamma_{t,u}} \varphi_q(y) \wedge P_a(x))\big)$. $\psi_{t,u}(x)$ verifies that $\mathfrak{O}$ changes to state u upon reading the letter at position x if $\mathfrak{O}$ was in state t before. θ says that for some suitable j we have precisely j state changes leading from i_0 to f. Since we can replace $\exists y(S(y,x) \wedge \cdots$ by $\forall y(S(y,x) \rightarrow \cdots$, we have $\psi_{i,j} \in \Delta_n$, hence $\theta \in \Sigma_{n+1}$. Again (see proof of Theorem 7) we conclude $\theta \in \Delta_{n+1}$ (the complement language is also in Σ_{n+1}). Clearly $L = L(\theta \wedge \varphi_{q_F}(\max))$, which is a Δ_{n+1} formula. □

Hence biased resets as factors in a cascade product have a limited impact on the dot-depth. We now define the *biased reset complexity* of a cascade product. Informally, the biased reset complexity is the number of resets in a product, where every block of biased resets is counted as a single reset. For instance, indicating biased resets by a square and non-biased resets by a circle, the following product has biased reset complexity 6:

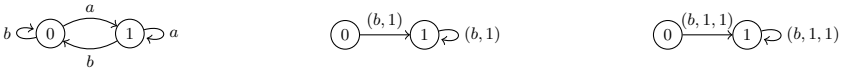
● ■ ■ ■ ● ■ ■ ● ●

More formally, let $\mathfrak{R}_1 * \cdots * \mathfrak{R}_n$ be a cascade product of resets. Let $B = \{(i,j) | \mathfrak{R}_k \text{ biased for } i \leq k \leq j \text{ and } \mathfrak{R}_{i-1}, \mathfrak{R}_{j+1} \text{ not biased}\}$. Let m be the number of biased resets in the product. The biased reset complexity is $n - m + |B|$. The following theorem is now immediate:

Theorem 8. *Let $\mathfrak{R}_1 * \cdots * \mathfrak{R}_n$ be a cascade product of resets with biased reset complexity k. Then $\mathcal{L}(\mathfrak{R}_1 * \cdots * \mathfrak{R}_n) \subseteq \Delta_{k+1}$.*

The following example shows that the bound given in Theorem 8 is not tight.

Example 1. Consider $L := \Gamma^* a \Gamma^* b \Gamma^* a \Gamma^* b \Gamma^*$ where $\Gamma = \{a, b\}$. Then define three resets as depicted in Fig. 1. The cascade product $\mathfrak{R}_1 * \mathfrak{R}_2 * \mathfrak{R}_3$ recognizes L with initial state $(0,0,0)$ and final states $\{(0,1,1),(1,1,1)\}$. Notice that Theorem 7 yields $L \in \Delta_4$, Theorem 8 yields $L \in \Delta_3$, but $L \in \Sigma_1$ as one easily verifies.



(a) $\mathfrak{R}_1$ (b) $\mathfrak{R}_2$ (c) $\mathfrak{R}_3$

Fig. 1. Resets for the language L. Inputs, which have not been specified, are assumed to induce the identity mapping.

7 Conclusion

Motivated by the Krohn-Rhodes Theorem we classified several classes of regular languages via their cascade decompositions. The concepts used, respectively introduced in this study, were biased resets, locally triggered cascade products, and the scope and the biased reset complexity of cascade products.

This paper gives initial results on the introduced concepts; it leads to several interesting questions left open here. For example, it should be answered whether for each n there is a star-free language L_n which needs at least scope n in the cascade decomposition of any (minimal) automaton accepting L_n. As another direction for future research (connected with the final result), we mention that the biased reset complexity of decompositions of automata for star-free languages seems to deserve a closer study.

Acknowledgments. The author thanks his supervisor Wolfgang Thomas and the reviewers of this paper for their advice and many helpful suggestions.

References

1. Brzozowski, J.A., Fich, F.E.: Languages of J-trivial monoids. Journal of Computer and System Sciences 20(1), 32–49 (1980)
2. Cohen, R.S., Brzozowski, J.A.: Dot-depth of star-free events. Journal of Computer and System Sciences 5(1), 1–16 (1971)
3. Eilenberg, S.: Automata, Languages, and Machines. Pure and Applied Mathematics, vol. A. Elsevier, Burlington (1974)
4. Eilenberg, S.: Automata, Languages, and Machines. Pure and Applied Mathematics, vol. B. Elsevier, Burlington (1974)
5. Gelderie, M.: Classifying regular languages via cascade products of automata. Diploma thesis, RWTH Aachen University (2011)
6. Ginzburg, A.: Algebraic Theory of Automata. Academic Press, New York (1968)
7. Krohn, K., Rhodes, J.: Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. Trans. Amer. Math. Soc. 116, 450–464 (1965)
8. Meyer, A.R.: A note on star-free events. J. ACM 16(2), 220–225 (1969)
9. Pin, J.E.: Varieties Of Formal Languages. Plenum Publishing Co., New York (1986)
10. Pin, J.E.: Syntactic semigroups. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Word, Language, Grammar, vol. 1, pp. 679–746. Springer-Verlag New York, Inc., New York (1997)
11. Schützenberger, M.P.: On finite monoids having only trivial subgroups. Information and Control 8(2), 190–194 (1965)
12. Simon, I.: Piecewise testable events. In: Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages, pp. 214–222. Springer, London (1975)
13. Straubing, H.: On finite J-trivial monoids. Semigroup Forum 19, 107–110 (1980)
14. Straubing, H.: Finite Automata, Formal Logic, and Circuit Complexity. Birkhauser Verlag, Basel (1994)
15. Straubing, H., Thérien, D.: Partially ordered finite monoids and a theorem of I. Simon. J. Algebra 119(2), 393–399 (1988)
16. Thomas, W.: Classifying regular events in symbolic logic. Journal of Computer and System Sciences 25(3), 360–376 (1982)

The Block Structure of Successor Morphisms

Jana Hadravová

Faculty of Mathematics and Physics, Charles University
186 75 Praha 8, Sokolovská 83, Czech Republic
hadravova@ff.cuni.cz

Abstract. Let $g_1, h_1 : \{a,b\}^* \to \{a,b\}^*$ be *successor morphisms* of non-periodic binary morphisms g, h. Suppose that g, h have a minimal solution which is not simple. Let $(e,f),(e',f')$ be *letter blocks* of g_1, h_1, that is, a prefix minimal pairs of words such that $g_1(e) = h_1(f)$ and $g_1(e') = h_1(f')$. In this paper we will show that if the primitive roots of words $\{g_1(a),\ g_1(b),\ h_1(a),\ h_1(b),\ e,\ f,\ e',\ f'\}$ have the length at least two, then the length of both letter blocks $(e,f),(e',f')$ is bounded by a constant.

1 Introduction

An equality word, also called a solution, of morphisms $g, h : \Sigma^* \to \Delta^*$ is a word w satisfying $g(w) = h(w)$. All equality words of the morphisms g, h constitute the set $\mathrm{Eq}(g,h)$, which is called the equality language of g and h. The concept of equality language was first introduced in [12] and since then has been widely studied. Equality languages have achieved particular importance in the representation theory of formal languages since every recursively enumerable language can be effectively found as a morphic image of an equality language, see [1].

The simplest non-trivial equality language is the equality language of binary morphisms. Although the binary equality language may seem at first sight very simple, in reality its structure is still unknown. One of the important results concerning binary equality languages is that the set of all binary equality languages containing at least one non-empty word is recursive (see [2], a complete proof given in [5]). This algorithmic problem known as the PCP(2) is a special variant of the *Post Correspondence Problem* (PCP), whose undecidability was proved by E. Post in [10].

As the PCP is one of the most emblematic algorithmic problems, lot of attention has been paid to its special variants. It is known that the PCP remains undecidable for instances whose domain alphabet is of size at least seven. On the other hand, the *marked version of the PCP*, that is when image words of both morphisms start with different letters of the target alphabet, was proved to be decidable (see [6]). The complexity of described decision algorithms for both the PCP(2) and the marked PCP is exponential. However, in case of the PCP(2), a polynomial algorithm was found in 2002 (see [7]).

The leading role in the proof of decidability of the PCP(2) is played by a reduction sequence of *successor morphisms*. Successor morphisms in the reduction sequence of g, h are in most cases strictly simpler than the previous pair

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 298–309, 2011.

of morphisms. A measure of 'simplicity' of a morphism in this case is its *suffix complexity*, which is simply the number of different non-empty suffixes of either of two image words. Although during the process we can reach a pair of morphisms with the same suffix complexity as its predecessor, these exceptions are known and can be treated separately. The complete classification of morphisms with stable suffix complexity can be found in [8].

Importantly, since elements of binary equality language are also transformed into simpler structures during the process of creation of the reduction sequence, determining the length of this sequence would be useful in the analysis of the structure of binary equality languages. Notice, however, that even though the problem of finding the structure of binary equality languages uses some facts from the proof of the PCP(2), both problems are significantly different.

Let $g_1, h_1 : \{a,b\}^* \to \{a,b\}^*$ be a pair of successor morphisms of g, h. Suppose that morphisms g, h have a minimal solution which does not have unique overflows. Let $(e,f),(e',f')$ be *letter blocks* of g_1, h_1, that is, a prefix minimal pairs of words such that $g_1(e) = h_1(f)$ and $g_1(e') = h_1(f')$. In this paper we will show that the length of both letter blocks $(e,f),(e',f')$ is bounded by a constant unless any of the words from $\{g_1(a), g_1(b), h_1(a), h_1(b), e, f, e', f'\}$ are a power of a letter.

From the bound for the length of the letter blocks of morphisms g_1, h_1 we can get directly the bound for the suffix complexity of their successors and thus a bound for the length of the reduction sequence of successor morphisms of g, h.

2 Basic Concepts and Definitions

The standard terminology and basic facts of combinatorics on words (see for example [9] and [11]) will be used throughout the text. Particularly, a reader shall recall that a binary morphism $g : \{a,b\}^* \to \Delta^*$ is *marked* if the image words $g(a)$ and $g(b)$ start with different letters. We shall use $u \leq_{\mathrm{p}} v$ when u is a *prefix* of v and $u <_{\mathrm{p}} v$ when u is a non-empty *proper prefix* of v. Similarly, $u \leq_{\mathrm{s}} v$ expresses the fact that u is a *suffix* of v and $u <_{\mathrm{s}} v$ means that u is a non-empty *proper suffix* of u. The set of all suffixes of a word u will be denoted by $\mathrm{Suff}(u)$. Two words are *suffix comparable* if one is a suffix of another. The *greatest common prefix* of two words u and v shall be denoted by $u \wedge v$. A (one-way) *infinite word* composed of infinite number of copies of a word u will be denoted by u^ω. By the *length of a word* u we mean the number of its letters and we denote it by $|u|$. Similarly, the number of occurrences of the letter a in u will be denoted by $|u|_a$. It should be also mentioned that the *primitive root* of a word u is the shortest word p such that $u = p^k$ for some positive k. The notation $\mathrm{pref}_k(u)$ will be used for the word $v \leq_{\mathrm{p}} u$ such that $|v| = k$. Since any alphabet can be encoded by two letters, we will suppose that $g : \{a,b\}^* \to \{a,b\}^*$.

It is a well-known fact that two words u, v commute if and only if they have the same primitive root. A binary morphism $g : \{a,b\}^* \to \{a,b\}^*$ is called non-periodic if its image words $g(a)$ and $g(b)$ do not commute. In what follows, we will be interested only in non-periodic morphisms.

Binary morphims have the following very important property: For each non-periodic binary morphism $g : \{a,b\}^* \to \{a,b\}^*$ there is a uniquely given marked (non-periodic) binary morphism $g_{\mathrm{m}} : \{a,b\}^* \to \{a,b\}^*$ and a word z_g such that for all words $w \in \{a,b\}^*$ we have $g(w) = z_g g_{\mathrm{m}}(w) z_g^{-1}$. One should note that z_g is in fact equal to $g(ab) \wedge g(ba)$ (see [11], p. 348).

Critical overflow. Let $g, h : \{a,b\}^* \to \{a,b\}^*$ be two binary non-periodic morphisms and g_{m}, h_{m} be their marked versions. If the words z_g and z_h are suffix comparable, we define the *critical overflow* $c(g,h)$ of g, h by

$$c(g,h) = z_h z_g^{-1} \ .$$

Note that if the original morphisms g and h are marked, the critical overflow exists and is empty.

Solution. Let $g, h : \{a,b\}^* \to \{a,b\}^*$ be two binary morphisms. A word w is a *solution* of g, h if $g(w) = h(w)$. We say that a solution w is *minimal* if it is not a prefix of any other solution. A solution w is called *simple* if all overflows are unique. That is, if w_1, $w_1 u$, w_2 and $w_2 u'$ are prefixes of w^ω such that

$$g(w_1)z = h(w_2) \qquad \text{and} \qquad g(w_1 u)z = h(w_2 u')$$

for some word $z \in \{a,b\}^* \cup (\{a,b\}^{-1})^*$, then $|u| = |u'| = k|w|$ for some $k \in \mathbb{N}_+$. Notice that every simple solution is minimal. However, the reverse implication does not hold (see Example 1).

Block decomposition of a solution. It is known that every minimal non-simple solution decomposes into simple structures called *letter blocks*. The letter block of binary morphisms g, h is a (prefix) minimal pair of words (e, f) such that

$$c(g,h)g(e) = h(f)c(g,h) \ . \tag{1}$$

Notice that if both morphisms g and h are marked, the letter blocks are just minimal pairs of words (e, f) such that $g(e) = h(f)$. It follows from (1) that for morphisms g, h, the corresponding marked morphisms $g_{\mathrm{m}}, h_{\mathrm{m}}$ have the same letter blocks as g, h. For each pair of binary morphisms there are at most two different letter blocks (e, f), (e', f') such that ef and $e'f'$ are non-empty. Moreover, $\mathrm{pref}_1(e) \neq \mathrm{pref}_1(e')$ and $\mathrm{pref}_1(f) \neq \mathrm{pref}_1(f')$.

For every minimal non-simple solution w of binary morphisms g and h there is a sequence

$$(u_0, v_0), \ldots, (u_n, v_n)$$

such that $w = u_0 \ldots u_n = v_0 \ldots v_n$ and $(u_n u_0, v_n v_0)$ and (u_i, v_i), $0 < i < n$, are letter blocks of g and h. This sequence is called a block decomposition of the solution w (see Fig. 1).

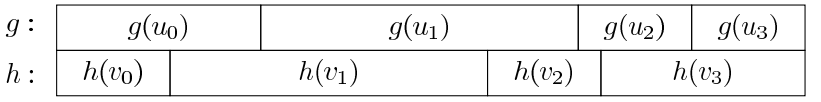


Fig. 1. Block decomposition of a solution

Successor morphisms. Morphisms $g_1, h_1 : \{a,b\}^* \to \{a,b\}^*$ are called *successor morphisms* if there exist non-periodic binary morphisms $g, h : \{a,b\}^* \to \{a,b\}^*$ such that $(g_1(a), h_1(a))$ and $(g_1(b), h_1(b))$ are their letter blocks.

Quite naturally, morphisms g, h will be called predecessor morphisms. Obviously, the predecessor morphisms are not uniquely given. Notice as well that successor morphisms are necessarily marked. This follows from the fact that for two different letter blocks (e, f), (e', f') we have $\mathrm{pref}_1(e) \neq \mathrm{pref}_1(e')$ and $\mathrm{pref}_1(f) \neq \mathrm{pref}_1(f')$.

Definition. A pair of successor morphisms (g_1, h_1) is called *saturated* if at least one pair of their predecessor morphisms has a non-simple minimal solution.

The following example shows that the condition that at least one pair of predecessor morphisms has a solution does not imply that the other pairs of predecessor morphisms of g_1 and h_1 have to possess a solution as well.

Example 1. Let $g_1, h_1 : \{a,b\}^* \to \{a,b\}^*$ be binary morphisms defined in the following way:

$$\begin{aligned} g_1(a) &= a, & h_1(a) &= ab, \\ g_1(b) &= bb, & h_1(b) &= b \ . \end{aligned}$$

It is easy to check that the following two pairs of morphisms (g, h) and (g', h') are predecessor morphisms of (g_1, h_1):

$$\begin{aligned} g(a) &= abb, & h(a) &= a, \\ g(b) &= b, & h(b) &= bb, \end{aligned}$$

$$\begin{aligned} g'(a) &= babab, & h'(a) &= b, \\ g'(b) &= ab, & h'(b) &= baba \ . \end{aligned}$$

The morphisms g, h have a minimal solution $w = abb$. Moreover, this solution is non-simple since it has two overflows which are the same. Namely, for $w_1 = a, w_2 = ab$ and $u = b^2, u' = b$ we have

$$g(w_1) = h(w_2) \qquad \text{and} \qquad g(w_1 u) = h(w_2 u') \ ,$$

but $|u|$ is not a multiple of $|w|$. Therefore, the pair of morphisms (g_1, h_1) is saturated. However, it can be checked easily that the pair (g', h') does not have any solution.

Definition. Let $g, h : \{a, b\}^* \to \{a, b\}^*$ be binary morphisms and let (e, f), (e', f') be their letter blocks. A pair of morphisms (g, h) is called *combinatorially rich* if the primitive roots of words $\{g(a), g(b), h(a), h(b), e, f, e', f'\}$ have the length at least two.

Notice that since the length of the primitive roots is at least two, each image word as well as each letter block word of combinatorially rich morphisms have to contain both letters a and b at least once.

We can now formulate our main result. Up to the morphisms which are not combinatorially rich, the block structure of successor morphisms is quite restricted. In fact, the length of both letter blocks of combinatorially rich successor morphisms is bounded by a constant.

Theorem 1. *Let $g_1, h_1 : \{a, b\}^* \to \{a, b\}^*$ be a pair of combinatorially rich saturated successor morphisms and let $(e, f), (e', f')$ be their letter blocks. Then $|e| + |e'| < 1412$ and $|f| + |f'| < 68$ or vice versa.*

The proof of the theorem is based on the analysis of the suffix complexity of successor morphisms of g_1, h_1.

Suffix complexity. The suffix complexity $\sigma(g)$ of a morphism g is defined as the number of different non-empty suffixes of $g(a)$ or $g(b)$. Formally,

$$\sigma(g) = |\{u, u \neq \epsilon \text{ and } u \in \mathrm{Suff}(g(a)) \cup \mathrm{Suff}(g(b))\}| \ .$$

The suffix complexity of a pair of morphisms (g, h) is defined as the sum of their suffix complexities. The concept was first introduced in the proof of the decidability of the binary PCP. It is known that suffix complexity of successor morphisms is less than or equal to the suffix complexity of its predecessors (see [2]). In this sense each successor morphism is simpler than its predecessor.

The suffix complexity of a morphism g is related to its length in the following way:

$$\frac{1}{2}(|g(a)| + |g(b)|) < \sigma(g) \leq |g(a)| + |g(b)| \ . \tag{2}$$

The suffix complexity of successor morphisms can be bounded by the number of certain types of overflows inside the letter blocks of their predecessor. Suppose that g_1, h_1 are successor morphisms of g, h and let $(e, f), (e', f')$ be letter blocks of g, h. Let $g_\mathrm{m}, h_\mathrm{m}$ be corresponding marked morphisms of g, h. We define a set O_g as the set of all non-empty suffixes of $g_\mathrm{m}(a)$ or $g_\mathrm{m}(b)$ which occur as overflows inside at least one of the letter blocks. More formally, $u \in O_g$ if u is a non-empty suffix of $g_\mathrm{m}(a)$ or $g_\mathrm{m}(b)$ and there are words e_1, e_2, f_1 and f_2 such that

$$g_\mathrm{m}(e_1) = h_\mathrm{m}(f_1)u \qquad \text{and} \qquad ug_\mathrm{m}(e_2) = h_\mathrm{m}(f_2) \ .$$

Each $u \in O_g$ determines uniquely a non-empty word $f_2 \in \mathrm{Suff}(f) \cup \mathrm{Suff}(f')$. This defines a mapping

$$\pi_g : O_g \to \{\mathrm{Suff}(f) \cup \mathrm{Suff}(f')\} \backslash \{\epsilon\} \ .$$

Since words f and f' are in fact words $h_1(a)$ and $h_1(b)$, we have obtained a mapping between O_g and $\sigma(h_1)$. Notice that this mapping is surjective, since for each non-empty $s \in \mathrm{Suff}(f)$ we can find $u \in O_g$ as $g_{\mathrm{m}}(e_1)(h_{\mathrm{m}}(fs^{-1}))^{-1}$ where e_1 is a minimal prefix of e such that $|g_{\mathrm{m}}(e_1)| > |h_{\mathrm{m}}(fs^{-1})|$. Surjectivity of π_g immediately implies the following bound for the suffix complexity of h_1:

$$\sigma(h_1) \leq |O_g| \ . \tag{3}$$

Now, one can easily find a bound for the length of the letter blocks of g, h simply by combining (2) and (3):

$$|f| + |f'| = |h_1(a)| + |h_1(b)| < 2\sigma(h_1) \leq 2|O_g| \ . \tag{4}$$

Similarly, we can define the set O_h and obtain

$$|e| + |e'| = |g_1(a)| + |g_1(b)| < 2\sigma(g_1) \leq 2|O_h| \ . \tag{5}$$

3 One Letter Bound for Successor Morphisms

We have seen that in order to prove Theorem 1, it is enough to find a bound for the cardinality of the sets O_{g_1} and O_{h_1}. Obviously, if the morphisms g_1, h_1 had bounded lengths, this task would be trivial. However, the difficulty of finding a length bound for morphisms g_1, h_1 is one of the reasons why the structure of binary equality languages remains unknown. Fortunately, it has turned out that to prove our theorem it is enough to bound the number of just one of the letters a, b inside the image words of g_1 and h_1. This section will discuss the problem of obtaining a one letter bound in more details.

The following lemma deals with the structure of sufficiently long letter blocks. In fact, it is a variation of the result obtained in [4] for simple solutions. This should not be surprising, since both simple solutions and letter blocks are simple structures. The proof uses practically same methods as the proof of the similar result in [3] and therefore will be omitted.

Lemma 1. *Let $g, h : \{a, b\}^* \to \{a, b\}^*$ be binary morphisms which have a non-simple minimal solution. Let (e, f), (e', f') be letter blocks of g, h and suppose that the word $h(b)$ is the longest of all four image words $\{g(a), h(a), g(b), h(b)\}$. If $|f|_b \geq 9$ or $|f'|_b \geq 9$, then one of the two letter blocks belongs to the following set:*

$$\{(b^i, p^j), (a^i, p^j), (p^i, b^j), (s^i, b^j)\} \tag{6}$$

where $i, j \in \mathbb{N}$ and p, s are primitive words such that $|p|_b = 1$ and $|s|_a = 1$.

Notice that the assumption that the word $h(b)$ is the longest of all four image words $\{g(a), h(a), g(b), h(b)\}$ in the previous lemma allows us to distinguish between letters a and b.

It follows from the previous lemma that at least one morphism in a pair of combinatorially rich saturated successor morphisms has a bounded number of one of the letters.

Corollary 1. *Let $g_1, h_1 : \{a,b\}^* \to \{a,b\}^*$ be a pair of combinatorially rich saturated successor morphisms. Then $|h_1(a)|_d \leq 8$ and $|h_1(b)|_d \leq 8$, or $|g_1(a)|_d \leq 8$ and $|g_1(b)|_d \leq 8$, for some letter $d \in \{a,b\}$.*

Proof. Let g, h be predecessor morphisms of g_1 and h_1. Since g_1, h_1 are saturated, we can suppose that g, h have a minimal non-simple solution. Let $(e,f),(e',f')$ be letter blocks of g, h and suppose that $h(b)$ is the longest of all four image words $\{g(a), h(a), g(b), h(b)\}$. By Lemma 1, either $|f|_b \leq 8$ and $|f'|_b \leq 8$, or one of the letter blocks belongs to the set (6). The latter possibility means that either $(g_1(a), h_1(a))$ or $(g_1(b), h_1(b))$ belongs to (6). This contradicts the assumption that g_1, h_1 are combinatorially rich morphisms. Therefore, $|h_1(a)|_b = |f|_b \leq 8$ and $|h_1(b)|_b = |f'|_b \leq 8$. □

In the previous lemma we have proved that at least one morphism in a pair of combinatorially rich saturated successor morphisms is bounded in one of the letters. However, in order to prove our theorem, we need to find the similar bound for the remaining morphism as well. We will start with the following lemma:

Lemma 2. *Let $g_1, h_1 : \{a,b\}^* \to \{a,b\}^*$ be a pair of combinatorially rich saturated successor morphisms and suppose that for a letter $d \in \{a,b\}$ it holds $|h_1(a)|_d \leq 8$ and $|h_1(b)|_d \leq 8$. Then $|g_1(a)|_d \leq 8$ or $|g_1(b)|_d \leq 8$.*

Proof. Let g, h be predecessors of g_1 and h_1. We can suppose that g, h have a minimal non-simple solution w. Let $(e,f),(e',f')$ be letter blocks of g, h. Since w is minimal and non-simple, each of the two letter blocks has to be included in the block decomposition of w at least once. Looking at the block decomposition of w we obtain the following equality for the letter d:

$$|w|_d = i|e|_d + j|e'|_d = i|f|_d + j|f'|_d,$$

where $i \geq 1$ and $j \geq 1$. Therefore, $|e|_d \geq |f|_d$ if and only if $|e'|_d \leq |f'|_d$. In other words, either $|e|_d$ or $|e'|_d$ is less than or equal to eight, since by our assumption $|f|_d = |h_1(a)|_d \leq 8$ and $|f'|_d = |h_1(b)|_d \leq 8$. This completes the proof. □

We have seen that there is a one letter bound for three out of four image words of a pair of combinatorially rich saturated successor morphisms. It turns out that finding a bound for the fourth word is much more complicated. The rest of this section is dedicated to this task.

The following slightly technical lemma is based on properties of cyclic words discussed in [4] and [3]. Because of space limitation, the proof will be omitted.

Lemma 3. *Let $g, h : \{a,b\}^* \to \{a,b\}^*$ be binary marked morphisms and suppose that*

$$g(u_1 c^i u_2) = v_1 h(d u_3 d u_4 d) v_2$$

for some letters $c, d \in \{a,b\}$ and words $u_1, u_2, u_3, u_4 \in \{a,b\}^*$, $v_1, v_2 \in \{a,b\}^*$ such that $|h(d)| \geq |g(c)|$, $v_2 v_1 \in \operatorname{Im} h$, $|g(u_1)| \leq |v_1|$ and $|g(u_2)| \leq |v_2|$. Then either $u_1 u_2 \in c^+$, or $g(c)$ commutes with $h(v)$, for some word $v \in \{a,b\}^+$.

Notice that if in previous lemma $g(c)$ commutes with $h(v)$, then $g(c^j) = h(v^k)$, for some $j, k \in \mathbb{N}_+$. Let (e, f) be a letter block of morphisms g, h such that $c \leq_{\mathrm{p}} e$. From the prefix minimality of (e, f) we obtain that $e \in c^+$. Thus, if we suppose that morphisms g, h from the previous lemma are combinatorially rich marked morphisms, then the words $g(c)$ and $h(v)$ cannot commute and therefore, necessarily $u_1u_2 \in c^+$.

The key ingredient in finding a bound for the remaining image word of successor morphisms g_1, h_1 is the fact that under certain conditions the block decomposition of a solution of predecessor morphisms g, h cannot contain long sequences of the same letter block. This is expressed more generally in the following way:

Lemma 4. *Let $g_1, h_1 : \{a, b\}^* \to \{a, b\}^*$ be a pair of combinatorially rich marked morphisms and suppose that $|h_1(a)|_b \leq 8$, $|h_1(b)|_b \leq 8$, and $|h_1(b)|_b < |g_1(b)|_b$. Let w be a word such that $zg_1(w) = h_1(w)z$, for a word $z \in \{a, b\}^* \cup \{a^{-1}, b^{-1}\}^*$, and $|w|_b \neq 0$. Then a^{49} is not a factor of w.*

Proof. Suppose for a contradiction that a^{49} is a factor of w. Since (g_1, h_1) is a combinatorially rich pair of morphisms, we know that $|g_1(a)|_b \neq 0$, otherwise the length of the primitive root of $g_1(a)$ would be one. Notice that since $zg_1(w) = h_1(w)z$, we can find words u_1, u_2, v_1, v_2, v such that

$$g_1(w) = g_1(u_1a^{49}u_2) = v_1h_1(v)v_2 \ ,$$

$v_2v_1 \in \operatorname{Im} h_1$, $|g_1(u_1)| \leq |v_1|$ and $|g_1(u_2)| \leq |v_2|$. Let v be the longest possible word satisfying this property. Then $|h_1(v)|_b \geq 33|g_1(a)|_b \geq 33$ because $|g_1(a)|_b \neq 0$ and we suppose that $|h_1(a)|_b \leq 8$ and $|h_1(b)|_b \leq 8$. And thus $|v| \geq 5$. Therefore, either $|v|_a \geq 3$ or $|v|_b \geq 3$. Since $|g_1(w)| = |h_1(w)|$, also

$$|w|_a(|h_1(a)|_b - |g_1(a)|_b) = |w|_b(|g_1(b)|_b - |h_1(b)|_b) \ .$$

From the fact that w contains both letters a and b and the assumption that $|h_1(b)|_b < |g_1(b)|_b$, we can see that $|h_1(a)|_b > |g_1(a)|_b$. Then, if $h_1(a)$ is a factor of $g_1(a)^\omega$, we have as well $|h_1(a)| > |g_1(a)|$. We are left with the following three cases:

1) $|v|_a \geq 3$. Then $h_1(a)$ is a factor of $g_1(a)^\omega$ and $|h_1(a)| > |g_1(a)|$. Therefore, we can apply Lemma 3 and obtain that $w \in a^+$, a contradiction with the assumption that $|w|_b \neq 0$.
2) $|v|_b \geq 3$ and $|h_1(b)| \geq |g_1(a)|$. Again we can apply Lemma 3 and get that $w \in a^+$, a contradiction with the assumption that $|w|_b \neq 0$.
3) $|v|_b \geq 3$, $|v|_a \leq 2$ and $|h_1(b)| < |g_1(a)|$. We will show that in this case there are at least five $g_1(a)$ inside $g_1(a)^{49}$ covered by $h_1(b)^\omega$. Then we can just exchange morphisms g_1 and h_1 and apply Lemma 3. We will obtain that $w \in b^+$ and the proof will be complete. Let $u_1, u_2, u_3 \in b^*$ be words such that $v = u_1a^{i_1}u_2a^{i_2}u_3$, where $i_1, i_2 \in \{0, 1\}$. Since $|h_1(v)|_b \geq 33|g_1(a)|_b$ and $|h_1(a)|_b \leq 8 \leq 8|g_1(a)|_b$, we have $|h_1(u_j)|_b > 5|g_1(a)|_b$ for at least one $j \in \{1, 2, 3\}$. Since $h_1(u_j)$ is a factor $g_1(a)^\omega$, we have as well $|h_1(u_j)| > 5|g_1(a)|$ and the proof is complete. □

In order to illustrate the connection between the previous lemma and the block decomposition of a solution of predecessor morphisms, suppose that g_1, h_1 are combinatorially rich successor morphisms of g, h. Let w be a non-simple minimal solution of g, h and let

$$(u_0, v_0), \ldots, (u_n, v_n)$$

be a block decomposition of w. Then $w = u_0 \ldots u_n = v_0 \ldots v_n$, $(u_n u_0, v_n v_0) = (g_1(c_n), h_1(c_n))$ and $(u_i, v_i) = (g_1(c_i), h_1(c_i))$, $0 < i < n$, where $c_i \in \{a, b\}$, $0 < i \leq n$. We can suppose that $|v_0| \leq |u_0|$. Then

$$(v_0^{-1} u_0) g_1(c_1 \ldots c_{n-1} c_n) = h_1(c_1 \ldots c_{n-1} c_n)(v_0^{-1} u_0) \ .$$

Since w is a non-simple minimal solution, the block decomposition includes each letter block at least once. Therefore, for $w' = c_1 \ldots c_n$ we have $|w'|_b \neq 0$ and $|w'|_a \neq 0$. By Corollary 1, we can suppose that $|h_1(a)|_d \leq 8$ and $|h_1(b)|_d \leq 8$, for a letter $d \in \{a, b\}$. If we suppose that $|h_1(b)|_d < |g_1(b)|_d$, then from the previous lemma we obtain that c^{49} is not a factor of w', where $c \neq d$. Therefore, the length of a maximal continuous sequence of letter blocks corresponding to the letter c in the block decomposition of w is at most 48.

Finally, let us formulate the following lemma which establishes a bound for the number of one of the letters inside the remaining image word of successor morphisms.

Lemma 5. *Let $g_1, h_1 : \{a, b\}^* \to \{a, b\}^*$ be a pair of combinatorially rich saturated successor morphisms. Suppose that the number of b's in the image words $h_1(a), h_1(b)$ and $g_1(a)$ is less than or equal to eight. Then $|g_1(b)|_b \leq 344$.*

Proof. Suppose, for a contradiction, that $|g_1(b)|_b \geq 345$. Then $|g_1(b)|_b > |h_1(b)|_b$. From the assumption that g_1, h_1 are combinatorially rich morphisms, we obtain that $|g_1(a)|_b \geq 1$. Let $g, h : \{a, b\}^* \to \{a, b\}^*$ be predecessor morphisms of g_1 and h_1 and let w be their non-simple minimal solution with the block decomposition:

$$(u_0, v_0), \ldots, (u_n, v_n) \ .$$

Then $w = u_0 \ldots u_n = v_0 \ldots v_n$, $(u_n u_0, v_n v_0) = (g_1(c_n), h_1(c_n))$ and $(u_i, v_i) = (g_1(c_i), h_1(c_i))$, $0 < i < n$, where $c_i \in \{a, b\}$, $0 < i \leq n$. Let $w' = c_1 \ldots c_n$. Notice that

$$(v_0^{-1} u_0) g_1(w') = h_1(w')(v_0^{-1} u_0) \ ,$$

and since w is a non-simple minimal solution, $|w'|_b \neq 0$ and $|w'|_a \neq 0$. It follows from the $|g_1(w')| = |h_1(w')|$ that

$$|w'|_b(|g_1(b)|_b - |h_1(b)|_b) = |w'|_a(|h_1(a)|_b - |g_1(a)|_b) \ .$$

Since $|g_1(b)|_b > |h_1(b)|_b$, necessarily $|g_1(a)|_b < |h_1(a)|_b$. Then it follows from inequalities $|g_1(a)|_b \leq 8$, $|h_1(a)|_b \leq 8$, $|h_1(b)|_b \leq 8$ that

$$7|w'|_a \geq (345 - 8)|w'|_b = 337|w'|_b > 7 \cdot 48|w'|_b \ .$$

By the pigeonhole principle, there is a consecutive sequence of 49 letters a inside the word w', which contradicts Lemma 4. □

4 Proof of Theorem 1

In this concluding section, we will argue that there is a bound for the cardinality of the sets O_{g_1} and O_{h_1} based on the previously found bounds for the number of one of the letters inside the the image words of successor morphisms. Then, having in mind (4) and (5), the proof of Theorem 1 will become substantially easy.

The next lemma (see [4]) is an easy application of the fact that for a marked morphism g there is at most one factorization of a word w into the image words $g(a)$ and $g(b)$.

Lemma 6. *Let g be a marked morphism and u, v, w be words satisfying*

$$g(u) \wedge w <_{\mathrm{p}} g(v) \wedge w \ .$$

Then $g(u) \wedge w = g(u \wedge v)$.

The following claim plays an important role in counting overflows inside the set O_{g_1} of the type a^+. For combinatorially rich marked morphisms g, h, there are at most two different g-overflows of the type a^+ inside the letter blocks of g and h.

Lemma 7. *Let $g, h : \{a, b\} \to \{a, b\}^*$ be marked binary morphisms and let $(e, f), (e', f')$ be their letter blocks. Let $(e_1, f_1), (e_2, f_2)$ and (e_3, f_3) be prefixes of (e, f) or (e', f') such that*

$$g(e_1) = h(f_1)a^i, \qquad g(e_2) = h(f_2)a^j \qquad \text{and} \qquad g(e_3) = h(f_3)a^k,$$

where $1 \leq i < j < k$. Then g, h are not combinatorially rich morphisms.

Proof. Suppose for contradiction that g, h are combinatorially rich morphisms. Let h_a be an image word $h(a)$ or $h(b)$ starting with a. Since g, h are combinatorially rich morphisms, there is a number $m \in \mathbb{N}_+$ such that $a^m b \leq_{\mathrm{p}} h_a$. Notice that since the morphism h is marked, overflows a^i, a^j and a^k cannot be longer than m. Then, there are words u, v such that

$$a^{m-j}b \leq_{\mathrm{p}} g(u),$$
$$a^{m-i}b \leq_{\mathrm{p}} g(v) \ .$$

Therefore, $g(u) \wedge a^\omega <_{\mathrm{p}} g(v) \wedge a^\omega$. According to Lemma 6, we get that $g(u \wedge v) = a^{m-j}$. Then $g(a) \in a^+$ or $g(b) \in a^+$ which is in contradiction to g, h being combinatorially rich morphisms. □

The same lemma as the previous one can be formulated for h-overflows; there are at most two different h-overflows of the type a^+ inside the letter blocks of g and h.

Finally, we will demonstrate that the cardinality of the sets O_{g_1} and O_{h_1} is bounded.

Lemma 8. *Let $g_1, h_1 : \{a,b\}^* \to \{a,b\}^*$ be a pair of combinatorially rich saturated successor morphisms. Then $|O_{g_1}| \leq 706$ and $|O_{h_1}| \leq 34$ or vice versa.*

Proof. Firstly, let us recall that successor morphisms are marked. Therefore, the set O_{g_1} consists of all words u such that u is a non-empty suffix of $g_1(a)$ or $g_1(b)$ and there are words e_1, e_2, f_1 and f_2 such that

$$g_1(e_1) = h_1(f_1)u \qquad \text{and} \qquad ug_1(e_2) = h_1(f_2)\ .$$

According to Corollary 1 and Lemma 2, we can suppose that the number of b's in image words $h_1(a), h_1(b), g_1(a)$ is less than or equal to eight. Applying Lemma 5, we then obtain that $|g_1(b)|_b \leq 344$. Let us count the elements of O_{h_1} which are suffixes of $h_1(a)$:
- starting with b: Since $|h_1(a)|_b \leq 8$, the number of different suffixes of $h_1(a)$ starting with the letter b is at most eight.
- starting with a: Since g_1, h_1 are combinatorially rich morphisms, there is a number $m \in \mathbb{N}_+$ such that $a^m b \leq_{\mathrm{p}} g_1(a)$ or $a^m b \leq_{\mathrm{p}} g_1(b)$. Then the number of possible overflows starting with the letter a longer than $m+1$ is at most eight. It remains to deal with overflows which have at most m letters. Since the morphism g_1 is marked, these overflows have to be of the form a^+. But by Lemma 7, there are at most two different h_1-overflows of the form a^+ inside the letter blocks of g_1, h_1.
Suffixes of $h_1(b)$**,** $g_1(a)$ **and** $g_1(b)$ can be counted in the same way as suffixes of $h_1(a)$. Again, we would find at most eight different suffixes starting with the letter b and eight different sufficiently long suffixes starting with the letter a for $h_1(b)$ and $g_1(a)$. In the case of $g_1(b)$, the only difference is that $|g_1(b)|_b \leq 344$. Therefore, there are at most 344 different suffixes starting with the letter b and 344 different sufficiently long suffixes starting with the letter a.

Having counted also overflows of the type a^+, we obtain that the cardinality of O_{h_1} is less than or equal to 34 and cardinality of O_{g_1} is less than or equal to 706, which completes the proof. □

Now, it becomes easy to prove our theorem.

Proof of Theorem 1. From (4) and (5), we obtain that

$$|f| + |f'| < 2|O_{g_1}|, \qquad |e| + |e'| < 2|O_{h_1}|\ .$$

By Lemma 8, we get

$$|f| + |f'| < 1412, \qquad |e| + |e'| < 68\ ,$$

or vice versa. This completes the proof. □

5 Reduction Sequence of Successor Morphisms

We have already mentioned that by the straightforward application of Theorem 1 we can find a bound for the reduction sequence of successors of morphisms g, h

which have non-simple minimal solution. Indeed, if $(g_i, h_i)_{i\geq 1}$ is the sequence of successors of non-periodic binary morphisms $g, h : \{a, b\}^* \to \{a, b\}^*$ with decreasing suffix complexity, than it follows from (4) and (5) that the length of the sequence is at most $|O_{g_1}| + |O_{h_1}| + 1 = 741$ unless any of the image words $\{g_i(a), g_i(b), h_i(a), h_i(b)\}_{\{i=1,2\}}$ are a power of a letter.

Knowing this bound is particularly important in the analysis of the structure of binary equality languages. The specific application in this field is a question for further research.

References

1. Culik II, K.: A purely homomorphic characterization of recursively enumerable sets. J. ACM 26(2), 345–350 (1979)
2. Ehrenfeucht, A., Karhumäki, J., Rozenberg, G.: The (generalized) Post Correspondence Problem with lists consisting of two words is decidable. Theoretical Computer Science 21, 119–144 (1982)
3. Hadravová, J.: A length bound for binary equality words. Comment. Math. Univ. Carolinae (to appear)
4. Hadravová, J., Holub, Š.: Large simple binary equality words. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 396–407. Springer, Heidelberg (2008)
5. Halava, V., Harju, T., Hirvensalo, M.: Binary (generalized) post correspondence problem. Theoretical Computer Science 276(1-2), 183–204 (2002)
6. Halava, V., Hirvensalo, M., de Wolf, R.: Marked PCP is decidable. Theoretical Computer Science 255, 193–204 (2001)
7. Halava, V., Holub, Š.: Reduction tree of the binary generalized Post Correspondence Problem. Internat. J. Found. Comput. Sci. 22(2), 473–490 (2011)
8. Holub, Š.: Binary morphisms with stable suffix complexity. Internat. J. Found. Comput. Sci. (to appear)
9. Lothaire, M.: Combinatorics on words. Addison-Wesley, Reading (1983)
10. Post, E.L.: A variant of a recursively unsolvable problem. Bulletin of the American Mathematical Society 52, 264–268 (1946)
11. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. word, language, grammar, vol. 1. Springer-Verlag New York, Inc., USA (1997)
12. Salomaa, A.: Equality sets for homomorphisms of free monoids. Acta Cybernetica 4(1), 127–139 (1978)

Models for Quantitative Distributed Systems and Multi-Valued Logics

Martin Huschenbett*

Technische Universität Ilmenau, Institut für Theoretische Informatik,
D-98963 Ilmenau, Germany
martin.huschenbett@tu-ilmenau.de

Abstract. We investigate weighted asynchronous cellular automata with weights in valuation monoids. These automata form a distributed extension of weighted finite automata and allow us to model concurrency. Valuation monoids are abstract weight structures that include semirings and (non-distributive) bounded lattices but also offer the possibility to model average behaviors. We prove that weighted asynchronous cellular automata and weighted finite automata which satisfy an I-diamond property are equally expressive. The main result of this paper gives a characterization of this expressiveness by weighted MSO logics.

1 Introduction

During the last decades, a fruitful connection between automata and logics revealed. This process started with Büchi [3] who used finite automata to obtain decidabilty results for logical problems. In particular, during his investigations he characterized the expressive powers of finite automata, i.e. the class of regular languages, by means of monadic second-order logics (MSO logics). However, finite automata are suitable only for modeling qualitative systems.

The interest in modeling quantitative systems led to several specialized extensions of finite automata like probabilistic automata and lattice automata [18]. A more generic approach is provided by the theory of weighted automata [8]. A weighted automaton is essentially a finite automaton with the additional feature that weights from an arbitrary semiring are assigned to each transition. Using the operations of the semiring such an automaton assigns weights to words. Although there is a well developed theory of weighted automata and the first prominent result was established by Schützenberger [21] already in the 1960s, semiring weighted logics were not taken into account for a long time. Droste & Gastin [7] closed this gap by characterizing the expressiveness of weighted automata by weighted MSO logics. Nowadays, such logics are still an ongoing subject of research [1,11].

Despite the very general nature of semirings there are quantitative aspects which cannot be modeled in this framework of weighted automata. These are aspects like the average consumption of some resource [4] or weights from –

* This paper is based on my Master's thesis [16] which I wrote at Universität Leipzig.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 310–322, 2011.

potentially non-distributive – bounded lattices [12], which are used in multi-valued logics. Recently, Droste & Meinecke [10] introduced valuation monoids to capture this variety of possible weight structures within one uniform framework. They studied weighted automata over such structures and characterized their expressive powers by certain fragments of weighted MSO logics.

All the automaton models mentioned so far take words as input and can consequently model concurrency only as interleaving. A valuable concept for modeling distributed systems are traces [6] in combination with Zielonka's asynchronous cellular automata [23]. Generalizing Büchi's result, Thomas [22] characterized the expressiveness of these automata again by MSO logics. Later on, this connection was extended to infinite traces by Ebinger & Muscholl [13].

Weighted asynchronous cellular automata with weights from a commutative semiring were introduced by Kuske [19], who showed them to be equally expressive as weighted I-diamond automata as well as weighted automata with a trace closed behavior. Afterwards, Meinecke [20] characterized this expressive power by weighted MSO logics. The combination of these results [14] generalizes Büchi's theorem to a distributed and semiring weighted setting. This naturally raises the question whether a similar characterization also exists when the weights are taken from a valuation monoid instead of a semiring. Therefore, this paper is devoted to the investigation of that issue.

The main results are as follows. First, we present weighted asynchronous cellular automata over valuation monoids as a model for quantitative distributed systems. Moreover, we define weighted I-diamond automata for modeling the interleaving behavior of such systems and show that both kinds of weighted automata are equally expressive. Second, we introduce weighted MSO logics over valuation monoids. The main theorem of this paper characterizes the expressiveness of weighted asynchronous cellular automata by this logics. In the end, we mention how the logics can be slightly extended if the valuation monoid has some additional properties. Altogether, we provide a joint extension of the results of Droste & Gastin [7], Fichtner, Kuske & Meinecke [14], and Droste & Meinecke [10].

2 Background: Traces and Asynchronous Cellular Automata

This section is intended to give the necessary background in trace theory. For a more general overview we refer the reader to [5,6].

2.1 Traces and Finite Automata

The architecture of a distributed system is modeled by a graph $(\mathcal{L}, \mathcal{D})$ consisting of a non-empty, finite set $\mathcal{L}$ of *locations* and a symmetric and reflexive *dependence relation* $\mathcal{D}$ on $\mathcal{L}$. For any $\ell \in \mathcal{L}$ the set of all $m \in \mathcal{L}$ with $(\ell, m) \in \mathcal{D}$ is denoted with $\mathcal{D}(\ell)$. *For the rest of this paper, we fix the graph* $(\mathcal{L}, \mathcal{D})$.

A *distributed alphabet* is a family $\Sigma = (\Sigma_\ell)_{\ell \in \mathcal{L}}$ of mutually disjoint alphabets. Abusing notation, we denote the set $\bigcup_{\ell \in \mathcal{L}} \Sigma_\ell$ by Σ as well. Moreover, we obtain a map $\mathrm{loc}\colon \Sigma \to \mathcal{L}$ by putting $\mathrm{loc}(a) = \ell$ for every $a \in \Sigma_\ell$.

A *trace* over Σ is a triplet $t = (V, \sqsubseteq, \lambda)$ consisting of a non-empty, finite set V, a partial order $\sqsubseteq$ on V, and a labeling $\lambda\colon V \to \Sigma$ such that for all $x, y \in V$ the following two conditions are satisfied, where $\mathrm{loc}_t = \mathrm{loc} \circ \lambda\colon V \to \mathcal{L}$:

(i) if $(\mathrm{loc}_t(x), \mathrm{loc}_t(y)) \in \mathcal{D}$, then $x \sqsubseteq y$ or $y \sqsubseteq x$,
(ii) if $x \sqsubset y$ and there is no $z \in V$ with $x \sqsubset z \sqsubset y$, then $(\mathrm{loc}_t(x), \mathrm{loc}_t(y)) \in \mathcal{D}$.

The set of all (isomorphism classes of) traces is denoted with $\mathbb{T}(\Sigma)$ and subsets $L \subseteq \mathbb{T}(\Sigma)$ are called *trace languages*.

To each word $w = a_1 \dots a_n \in \Sigma^+$ we assign a trace $\mathrm{trc}(w) = (V, \sqsubseteq, \lambda) \in \mathbb{T}(\Sigma)$ as follows: $V = \{1, \dots, n\}$, $\lambda(i) = a_i$, and $\sqsubseteq$ is the transitive closure of

$$E = \{\, (i, j) \in V \times V \mid i \leq j \text{ and } (\mathrm{loc}(a_i), \mathrm{loc}(a_j)) \in \mathcal{D} \,\} \quad .$$

In this way, we obtain a surjective map $\mathrm{trc}\colon \Sigma^+ \to \mathbb{T}(\Sigma)$. In the case of $|\mathcal{L}| = 1$, the order $\sqsubseteq$ is the natural linear order on V and the map $\mathrm{trc}\colon \Sigma^+ \to \mathbb{T}(\Sigma)$ is a bijection. Thus, we can consider words as a special case of traces.

Two words $w, w' \in \Sigma^+$ are called *trace equivalent* if $\mathrm{trc}(w) = \mathrm{trc}(w')$. The *independence relation* $\mathcal{I}$ is the set of all pairs $(a, b) \in \Sigma \times \Sigma$ with $(\mathrm{loc}(a), \mathrm{loc}(b)) \notin \mathcal{D}$. It is well known that two words are trace equivalent iff they are related by the least equivalence relation $\sim_{\mathcal{I}}$ on Σ^+ satisfying $uabv \sim_{\mathcal{I}} ubav$ for all $(a, b) \in \mathcal{I}$ and $u, v \in \Sigma^*$.

A *finite automaton* over Σ (*FA* for short) is a tuple $\mathcal{M} = (Q, I, T, F)$ consisting of a finite set Q of states, a transition relation $T \subseteq Q \times \Sigma \times Q$, and two sets $I, F \subseteq Q$ of initial and final states. A *run* of $\mathcal{M}$ on a word $a_1 \dots a_n \in \Sigma^+$ is a sequence $\sigma = (q_0, a_1, q_1)(q_1, a_2, q_2) \dots (q_{n-1}, a_n, q_n) \in T^n$ with $q_0 \in I$. The run σ is *successful* if $q_n \in F$. The language *recognized* by $\mathcal{M}$ is the set $L(\mathcal{M})$ of all words $w \in \Sigma^+$ which admit a successful run.

An FA $\mathcal{M}$ has the *$\mathcal{I}$-diamond property* if for all $p, q, r \in Q$ and $(a, b) \in \mathcal{I}$ with $(p, a, q), (q, b, r) \in T$ there is some $q' \in Q$ such that $(p, b, q'), (q', a, r) \in T$. In this situation the language recognized by $\mathcal{M}$ is *trace closed*, i.e., for all $u, v \in \Sigma^+$ with $u \sim_{\mathcal{I}} v$ we have $u \in L(\mathcal{M})$ iff $v \in L(\mathcal{M})$. Therefore, FAs with the $\mathcal{I}$-diamond property can be used as recognizers for trace languages.

2.2 Asynchronous Cellular Automata

Asynchronous cellular automata [23] are a distributed extension of classical finite automata. An *asynchronous cellular automaton* over Σ (*ACA* for short) is a tuple $\mathcal{A} = ((Q_\ell)_{\ell \in \mathcal{L}}, I, (T_\ell)_{\ell \in \mathcal{L}}, F)$ consisting of a finite set of local states Q_ℓ and a local transition relation $T_\ell \subseteq \left(\prod_{m \in \mathcal{D}(\ell)} Q_m\right) \times \Sigma_\ell \times Q_\ell$ for each $\ell \in \mathcal{L}$, and two sets $I, F \subseteq \prod_{\ell \in \mathcal{L}} Q_\ell$ of global initial and final states. The ACA $\mathcal{A}$ is *deterministic* if I is a singleton and each T_ℓ is a partial map $\left(\prod_{m \in \mathcal{D}(\ell)} Q_m\right) \times \Sigma_\ell \to Q_\ell$.

For any transition $\tau = ((p_m)_{m \in \mathcal{D}(\ell)}, a, q) \in T_\ell$ we put $\mathrm{read}_m(\tau) = p_m$ for $m \in \mathcal{D}(\ell)$, $\mathrm{act}(\tau) = a$, and $\mathrm{write}(\tau) = q$. A *run* of $\mathcal{A}$ on a trace $t = (V, \sqsubseteq, \lambda) \in$

$\mathbb{T}(\Sigma)$ is a pair $\rho = (\iota, r)$ consisting of a global initial state $\iota = (\iota_m)_{m\in\mathcal{L}} \in I$ and a map $r\colon V \to \bigcup_{\ell\in\mathcal{L}} T_\ell$ such that for every $x \in V$ the following two conditions are satisfied, where $\tau = r(x)$ and $\ell = \mathrm{loc}_t(x)$:

(i) $\tau \in T_\ell$ and $\mathrm{act}(\tau) = \lambda(x)$, and
(ii) for every $m \in \mathcal{D}(\ell)$ we have either $\mathrm{read}_m(\tau) = \mathrm{write}(r(y_{\max}))$ where $y_{\max}$ is the greatest (w.r.t. $\sqsubseteq$) $y \in V$ with $y \sqsubset x$ and $\mathrm{loc}_t(y) = m$, or $\mathrm{read}_m(\tau) = \iota_m$ if no such y exists.

The run ρ is *successful* if there is some $f = (f_m)_{m\in\mathcal{L}} \in F$ such that for every $m \in \mathcal{L}$ we have either $f_m = \mathrm{write}(r(x_{\max}))$ where $x_{\max}$ is the greatest (w.r.t. $\sqsubseteq$) $x \in V$ with $\mathrm{loc}_t(x) = m$, or $f_m = \iota_m$ if no such x exists. The language *recognized* by $\mathcal{A}$ is the set $L(\mathcal{A})$ of all traces $t \in \mathbb{T}(\Sigma)$ which admit a successful run. The connection between ACAs and FAs with the $\mathcal{I}$-diamond property was established by Zielonka:

Theorem 2.1 (Zielonka [23]). *Let $L \subseteq \mathbb{T}(\Sigma)$ be a trace language. The following are equivalent:*

(1) L is recognized by some ACA over Σ,
(2) L is recognized by some deterministic ACA over Σ,
(3) $\mathrm{trc}^{-1}(L)$ is recognized by some FA over Σ,
(4) $\mathrm{trc}^{-1}(L)$ is recognized by some deterministic FA over Σ having the $\mathcal{I}$-diamond property.

2.3 MSO Logic on Traces

For the rest of this section, we provide two disjoint infinite sets $\mathcal{V}_0$ and $\mathcal{V}_1$ of elementary and set variables, respectively. The syntax of MSO logics over Σ is given by the grammar

$$\varphi ::= x \le y \mid \lambda(x) = a \mid x \in X \mid \neg\varphi \mid \varphi \wedge \varphi \mid \forall x\, \varphi \mid \forall X\, \varphi \ ,$$

where $a \in \Sigma$, $x, y \in \mathcal{V}_0$, and $X \in \mathcal{V}_1$. Let φ be an MSO-sentence, i.e., an MSO-formula without free variables. For every trace $t = (V, \sqsubseteq, \lambda) \in \mathbb{T}(\Sigma)$ there is an obvious meaning of $t \models \varphi$ under the assumption that variables from $\mathcal{V}_0$ resp. $\mathcal{V}_1$ range over elements resp. subsets of V and $\le$ is interpreted by $\sqsubseteq$. The language *defined* by φ is the set $L(\varphi)$ of all traces $t \in \mathbb{T}(\Sigma)$ with $t \models \varphi$. The connection between ACAs and MSO logics was established by Thomas:

Theorem 2.2 (Thomas [22]). *Let $L \subseteq \mathbb{T}(\Sigma)$ be a trace language. Then L is recognized by some ACA over Σ if and only if L is defined by some MSO-sentence φ over Σ.*

3 Weighted Asynchronous Cellular Automata

In this section we want to extend the model of asynchronous cellular automata by adding transition weights. As weight structures we consider valuation monoids, which were recently introduced by Droste & Meinecke [10]. The highlight of this section is a weighted version of Thm. 2.1 on page 316.

3.1 Valuation Monoids and Weighted Finite Automata

A *valuation monoid* is an algebraic structure $(D, +, \mathrm{Val}, \mathbf{0})$ consisting of a commutative monoid $(D, +, \mathbf{0})$ and a *valuation function* $\mathrm{Val}\colon D^+ \to D$ such that $\mathrm{Val}(d) = d$ for all $d \in D$, and $\mathrm{Val}(d_1, \ldots, d_n) = \mathbf{0}$ whenever $d_i = \mathbf{0}$ for some $i \in \{1, \ldots, n\}$.

Example 3.1. The following structures are valuation monoids:

(a) $(\mathbb{Q} \cup \{-\infty\}, \max, \mathrm{avg}, -\infty)$ and $(\mathbb{Q} \cup \{\infty\}, \min, \mathrm{avg}, \infty)$ where

$$\mathrm{avg}(d_1, \ldots, d_n) = \frac{d_1 + \cdots + d_n}{n} \quad ,$$

(b) $(K, +, \prod, \mathbf{0})$ where $(K, +, \cdot, \mathbf{0}, \mathbf{1})$ is a semiring and $\prod(d_1, \ldots, d_n) = d_1 \cdots d_n$,

(c) $(\mathcal{L}, \vee, \inf, \bot)$ where $(\mathcal{L}, \vee, \wedge, \bot, \top)$ is a (non-distributive) bounded lattice and $\inf(d_1, \ldots, d_n) = d_1 \wedge \cdots \wedge d_n$.

The valuation monoid in (a) allows us to model the average consumption of some resource, an aspect which cannot be captured by the semiring weighted framework. For more examples we refer the reader to [10].

Before introducing weighted asynchronous cellular automata, we recall the weighted automaton model of Droste & Meinecke [10]. A *weighted finite automaton* over an alphabet Σ and a valuation monoid D (*wFA* for short) is a tuple $\mathcal{M} = (Q, I, T, F, \gamma)$ consisting of an FA (Q, I, T, F) over Σ and a *transition weight function* $\gamma\colon T \to D$. (Successful) runs of $\mathcal{M}$ are defined as for the underlying FA. The *weight* of a run $\sigma = \tau_1 \ldots \tau_n \in T^+$ is

$$\gamma(\sigma) = \mathrm{Val}\big(\gamma(\tau_1), \ldots, \gamma(\tau_n)\big)$$

and the *behavior* of $\mathcal{M}$ is the mapping $\|\mathcal{M}\|\colon \Sigma^+ \to D$ defined by

$$\|\mathcal{M}\|(w) = \sum \{\, \gamma(\sigma) \mid \sigma \text{ is a successful run of } \mathcal{M} \text{ on } w \,\} \quad .$$

3.2 Weighted Asynchronous Cellular Automata

In the definition of behavior above, Val is used to combine the transition weights of a single run, whereas $+$ is used to collect the weights of all different successful runs on one trace. Since there is no natural execution order of the transitions of an ACA, we require another property of valuation monoids:

Definition 3.2. *A valuation monoid* $(D, +, \mathrm{Val}, \mathbf{0})$ *is* order independent *if*

$$\mathrm{Val}(d_1, \ldots, d_n) = \mathrm{Val}(d_{\pi(1)}, \ldots, d_{\pi(n)})$$

holds true for all $(d_1, \ldots, d_n) \in D^+$ *and permutations* π *of* $\{1, \ldots, n\}$.

Example 3.3. The valuation monoids in Example 3.1 (a) and (c) are order independent, whereas this is only the case for (b) iff K is commutative.

For the rest of this section we fix a distributed alphabet Σ and an order independent valuation monoid $(D, +, \mathrm{Val}, \mathbf{0})$.

Definition 3.4. *A* weighted asynchronous cellular automaton *over Σ and D (*wACA *for short) is a tuple $\mathcal{A} = ((Q_\ell)_{\ell\in\mathcal{L}}, I, (T_\ell)_{\ell\in\mathcal{L}}, F, (\gamma_\ell)_{\ell\in\mathcal{L}})$ consisting of an ACA $((Q_\ell)_{\ell\in\mathcal{L}}, I, (T_\ell)_{\ell\in\mathcal{L}}, F)$ over Σ and* transition weight functions *$\gamma_\ell \colon T_\ell \to D$.*

(Successful) runs of a wACA are defined as for the underlying ACA. The *weight* of a run $\rho = (\iota, r)$ on a trace $t = (V, \sqsubseteq, \lambda) \in \mathbb{T}(\Sigma)$ is

$$\gamma(\rho) = \mathrm{Val}\Big(\big(\gamma_{\mathrm{loc}_t(x)}(r(x))\big)_{x\in V}\Big) \ .$$

This definition does not depend on the order in which the elements of V are enumerated, since D is order independent.

Definition 3.5. *Let $\mathcal{A}$ be a wACA over Σ and D. The* behavior *of $\mathcal{A}$ is the function $\|\mathcal{A}\| \colon \mathbb{T}(\Sigma) \to D$ defined as*

$$\|\mathcal{A}\|(t) = \sum \{\, \gamma(\rho) \mid \rho \text{ is a successful run of } \mathcal{A} \text{ on } t \,\} \ .$$

This definition suggests that maps $\mathbb{T}(\Sigma) \to D$ are subject to our interest. Thus, they get the concise name *trace series* from now on. In particular, we are interested in those trace series that can occur as the behavior of some wACA. Similarly, mappings $\Sigma^+ \to D$ are called *word series*.

3.3 The $\mathcal{I}$-diamond Property and Its Relationship to wACAs

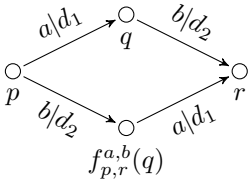
In order to use wFAs in the context of trace series we are interested in those automata $\mathcal{M}$ with a *trace closed* behavior, i.e., for all $u, v \in \Sigma^+$ with $u \sim_{\mathcal{I}} v$ we have $\|\mathcal{M}\|(u) = \|\mathcal{M}\|(v)$. In general, it is undecidable whether a given wFA has this property, cf. Prop. 3.1 in [17]. However, the following definition gives a syntactical property of wFAs which is sufficient for a trace closed behavior. For some wFA $\mathcal{M} = (Q, I, T, F, \gamma)$ over Σ and all $a, b \in \Sigma$ and $p, r \in Q$ we put

$$Q^{a,b}_{p,r} = \{\, q \in Q \mid (p, a, q) \in T \text{ and } (q, b, r) \in T \,\} \ .$$

Definition 3.6. *A wFA $\mathcal{M} = (Q, I, T, F, \gamma)$ has the* $\mathcal{I}$-diamond property *if for all $p, r \in Q$ and $(a, b) \in \mathcal{I}$ there is a bijective map $f = f^{a,b}_{p,r} \colon Q^{a,b}_{p,r} \to Q^{b,a}_{p,r}$ such that for any $q \in Q^{a,b}_{p,r}$ we have*

$$\gamma(p, a, q) = \gamma(f(q), a, r) \quad \text{and} \quad \gamma(q, b, r) = \gamma(p, b, f(q)) \ .$$

The idea behind is depicted in the figure on the right. Consider some states $p, r \in Q$ and a pair of independent letters $(a, b) \in \mathcal{I}$. For any path from p to r (via some $q \in Q$) labeled with ab and weighted by d_1 and d_2 there must be path between p and r (via $f^{a,b}_{p,r}(q)$) which is labeled with ba and has the same weights, but in the opposite order. Moreover, this correspondence between



pathes has to be one-to-one. Intuitively, in this situation the execution order of the independent actions a and b does not matter. This condition is similar to that in a semiring weighted setting [14], where the transition matrices of independent letters are required to commute. But since valuation monoids provide less algebraic properties than semirings, we need the existence of the bijections $f_{p,r}^{a,b}$ in order to prove the following lemma:

Lemma 3.7. *If a wFA $\mathcal{M}$ over Σ has the $\mathcal{I}$-diamond property, then its behavior $\|\mathcal{M}\|$ is trace closed.*

Proof. Due to the definition of $\sim_{\mathcal{I}}$ it suffices to show that $\|\mathcal{M}\|(w) = \|\mathcal{M}\|(w')$ holds true for all $w = uabv$ and $w' = ubav$ with $u, v \in \Sigma^*$ and $(a, b) \in \mathcal{I}$. Consider a successful run σ of $\mathcal{M}$ on w and let $\tau = (p, a, q)$ and $\tau' = (q, b, r)$ be the transitions within this run which belong to the distinguished letters a and b in w. If we replace τ and τ' in σ by (p, b, q') and (q', a, r), where $q' = f_{p,r}^{a,b}(q)$, the result is a successful run σ' on w'. Due to the order independence of D, this run satisfies $\gamma(\sigma) = \gamma(\sigma')$. Since $f_{p,r}^{a,b}$ is a bijection, this construction yields a weight preserving one-to-one correspondence between the successful runs of $\mathcal{M}$ on u and those on v. □

In order to formulate our first new result, we need to assign to each trace series $S\colon \mathbb{T}(\Sigma) \to D$ a word series $\mathrm{trc}^{-1}(S)\colon \Sigma^+ \to D$ defined by

$$\mathrm{trc}^{-1}(S)(w) = S(\mathrm{trc}(w)) \ .$$

Theorem 3.8. *Let $S\colon \mathbb{T}(\Sigma) \to D$ be a trace series. The following are equivalent:*

(1) S is the behavior of some wACA over Σ,
(2) $\mathrm{trc}^{-1}(S)$ is the behavior of some wFA over Σ with the $\mathcal{I}$-diamond property,
(3) $\mathrm{trc}^{-1}(S)$ is the behavior of some wFA over Σ.

The implication *(2) ⇒ (3)* is trivial and *(1) ⇒ (2)* is shown below. Proving *(3) ⇒ (1)* needs some preperation and is done afterwards.

Proof (Sketch for (1) ⇒ (2)). Let $\mathcal{A} = ((Q_\ell)_{\ell\in\mathcal{L}}, I, (T_\ell)_{\ell\in\mathcal{L}}, F, (\gamma_\ell)_{\ell\in\mathcal{L}})$ be a wACA such that $\|\mathcal{A}\| = S$. We construct a wFA $\mathcal{M} = (Q, I, T, F, \gamma)$ as follows: its state space is $Q = \prod_{\ell\in\mathcal{L}} Q_\ell$, the transition relation T is the set of all tuples $(p, a, q) \in Q \times \Sigma \times Q$ such that, with $\ell = \mathrm{loc}(a)$, we have $((p_m)_{m\in\mathcal{D}(\ell)}, a, q_\ell) \in T_\ell$ and $p_m = q_m$ for all $m \neq \ell$, and the weight $\gamma(p, a, q)$ of such a transition is given as $\gamma_\ell((p_m)_{m\in\mathcal{D}(\ell)}, a, q_\ell)$. To verify that $\mathcal{M}$ has the $\mathcal{I}$-diamond property, for $p, r \in Q$ and $(a, b) \in \mathcal{I}$ we can choose $f_{p,r}^{a,b}(q) = q'$ with $q'_{\mathrm{loc}(b)} = r_{\mathrm{loc}(b)}$ and $q'_\ell = p_\ell$ for $\ell \neq \mathrm{loc}(b)$.

In order to show $\|\mathcal{M}\| = \mathrm{trc}^{-1}(S)$, consider a word $w = a_1 \dots a_n \in \Sigma^+$, the trace $t = \mathrm{trc}(w) \in \mathbb{T}(\Sigma)$, and a successful run $\sigma = (q_0, a_1, q_1) \dots (q_{n-1}, a_n, q_n) \in T^n$ of $\mathcal{M}$ on w. For each $i = 1, \dots, n$ we put $r(i) = \big(((q_{i-1})_m)_{m\in\mathcal{D}(\ell)}, a_i, (q_i)_\ell\big)$, where $\ell = \mathrm{loc}(a)$. Then, $\rho = (q_0, r)$ is a successful run of $\mathcal{A}$ on t with $\gamma(\sigma) = \gamma(\rho)$. Moreover, this constrution yields a bijection between the successful runs of $\mathcal{M}$ on w and those of $\mathcal{A}$ on t. We can conclude $\|\mathcal{M}\| = \mathrm{trc}^{-1}(\|\mathcal{A}\|) = \mathrm{trc}^{-1}(S)$. □

In order to prove the remaining implication we need the concept of lexicographic normal forms, which was already used by Kuske [19] for a similar purpose. Therefore, we fix a linear order $\preceq$ on $\mathcal{L}$ and denote the induced lexicographic order on $\mathcal{L}^+$ also by $\preceq$. To each $w = a_1 \dots a_n \in \Sigma^+$ we assign the sequence $\mathrm{loc}(w) = \mathrm{loc}(a_1) \dots \mathrm{loc}(a_n) \in \mathcal{L}^+$. A word $w \in \Sigma^+$ is in *lexicographic normal form* if for all $u \in \Sigma^+$ with $w \sim_{\mathcal{I}} u$ we have $\mathrm{loc}(w) \preceq \mathrm{loc}(u)$. For every trace $t \in \mathbb{T}(\Sigma)$ there is exactly one word $\mathrm{lnf}(t) \in \mathrm{trc}^{-1}(t)$ which is in lexicographic normal form.

The main idea behind the missing proof is as follows: from a wFA $\mathcal{M}$ with behavior $\mathrm{trc}^{-1}(S)$ we construct a wACA which simulates $\mathcal{M}$ on the lexicographic normal form of each given trace.

Proof (Sketch for Thm. 3.8 (3) $\Rightarrow$ (1)). Let $\mathcal{M} = (Q, I', T, F', \gamma)$ be a wFA such that $\|\mathcal{M}\| = \mathrm{trc}^{-1}(S)$. We consider the distributed alphabet $\Gamma = (\Gamma_\ell)_{\ell \in \mathcal{L}}$ with $\Gamma_\ell = \{ (p, a, q) \in T \mid a \in \Sigma_\ell \}$ and let $L \subseteq \Gamma^+$ be the set of all successful runs of $\mathcal{M}$ on words in lexicographic normal form. Using the same arguments as in the proof of Prop. 4.6 in [19], there exists a deterministic ACA $\mathcal{A} = ((Q_\ell)_{\ell \in \mathcal{L}}, I, (T_\ell)_{\ell \in \mathcal{L}}, F)$ over Γ recognizing $\mathrm{trc}(L) \subseteq \mathbb{T}(\Gamma)$. We extend $\mathcal{A}$ to a wACA $\mathcal{A}' = (\mathcal{A}, (\gamma_\ell)_{\ell \in \mathcal{L}})$ by putting $\gamma_\ell((p_m)_{m \in \mathcal{D}(\ell)}, \tau, q) = \gamma(\tau)$. For $u = (U, \sqsubseteq, \mu) \in \mathrm{trc}(L)$ we obtain

$$\|\mathcal{A}'\|(u) = \mathrm{Val}\Big(\big(\gamma(\mu(x))\big)_{x \in U} \Big) \tag{1}$$

and $\|\mathcal{A}'\|(u) = \mathbf{0}$ for all other $u \in \mathbb{T}(\Gamma)$.

Now, we extend the projection $\pi \colon \Gamma \to \Sigma, (p, a, q) \mapsto a$ to the trace level by putting $\pi(u) = (U, \sqsubseteq, \pi \circ \mu)$ for any $u = (U, \sqsubseteq, \mu) \in \mathbb{T}(\Gamma)$. Since $\pi(\Gamma_\ell) \subseteq \Sigma_\ell$ for every $\ell \in \mathcal{L}$, this defines a map $\pi \colon \mathbb{T}(\Gamma) \to \mathbb{T}(\Sigma)$. Applying the same construction as in the proof of Prop. 4.10 in [19] to $\mathcal{A}'$ yields a wACA $\mathcal{B}$ over Σ such that $\|\mathcal{B}\|(t) = \sum_{u \in \pi^{-1}(t)} \|\mathcal{A}'\|(u)$ for any $t \in \mathbb{T}(\Sigma)$. In combination with (1), we obtain

$$\|\mathcal{B}\|(t) = \sum_{(U, \sqsubseteq, \mu) \in \pi^{-1}(t) \cap \mathrm{trc}(L)} \mathrm{Val}\Big(\big(\gamma(\mu(x))\big)_{x \in U} \Big) \ .$$

Since $\pi^{-1}(t) \cap \mathrm{trc}(L)$ encodes exactly the successful runs of $\mathcal{M}$ on $\mathrm{lnf}(t)$, the claim follows. □

4 Weighted MSO Logics

The goal of this section is to characterize the class of trace series which can occur as the behavior of some wACA by means of weighted MSO logics. Since valuation monoids neither provide an operation for interpreting conjunction nor a natural candidate for the truth value "true", we need to extend them to product valuation monoids [10].

4.1 Product Valuation Monoids

A *product valuation monoid* (*pv-monoid*) is a structure $(D, +, \mathrm{Val}, \diamond, \mathbf{0}, \mathbf{1})$ consisting of a valuation monoid $(D, +, \mathrm{Val}, \mathbf{0})$, a constant $\mathbf{1} \in D$, and a binary operation $\diamond$ on D such that $\mathbf{0} \diamond d = d \diamond \mathbf{0} = \mathbf{0}$ and $\mathbf{1} \diamond d = d \diamond \mathbf{1} = d$ for all $d \in D$, and $\mathrm{Val}(\mathbf{1}, \ldots, \mathbf{1}) = \mathbf{1}$. A pv-monoid is *order independent* if the underlying valuation monoid has this property. Notice that $\diamond$ and Val restricted to $\{\mathbf{0}, \mathbf{1}\}$ model classical boolean conjunction and universal quantification.

An order independent pv-monoid D is *regular* if for any $d \in D$ the constant word series $\Sigma^+ \to D, w \mapsto d$ is the behavior of some wFA, or equivalently, the constant trace series $\mathbb{T}(\Sigma) \to D, t \mapsto d$ is the behavior of some wACA. This property does not depend on the choice of the distributed alphabet Σ.

Example 4.1 (Continues Example 3.1). The following are regular pv-monoids:

(a) $(\mathbb{Q} \cup \{-\infty\}, \max, \mathrm{avg}, +, -\infty, 0)$ and $(\mathbb{Q} \cup \{\infty\}, \min, \mathrm{avg}, +, \infty, 0)$,
(b) $(K, +, \prod, \cdot, \mathbf{0}, \mathbf{1})$ for any commutative semiring $(K, +, \cdot, \mathbf{0}, \mathbf{1})$,
(c) $(\mathcal{L}, \vee, \inf, \wedge, \bot, \top)$ for any bounded lattice $(\mathcal{L}, \vee, \wedge, \bot, \top)$.

4.2 Weighted MSO Logics for Traces

For the rest of this section we fix a distributed alphabet Σ and an order independent product valuation monoid $(D, +, \mathrm{Val}, \diamond, \mathbf{0}, \mathbf{1})$. Again, we provide two disjoint infinite sets $\mathcal{V}_0$ and $\mathcal{V}_1$ of elementary and set variables, respectively. The syntax of *weighted MSO logics* over Σ and D (*wMSO logics* for short) is given by the grammar[1]

$$\begin{aligned}
\beta &::= x \leq y \mid \lambda(x) = a \mid x \in X \mid \neg\beta \mid \beta \wedge \beta \mid \forall x\, \beta \mid \forall X\, \beta \ , \\
\gamma &::= d \mid \beta \mid \gamma \vee \gamma \mid \gamma \wedge \gamma \ , \\
\varphi &::= \gamma \mid \varphi \vee \varphi \mid \beta \wedge \varphi \mid \varphi \wedge \beta \mid \exists x\, \varphi \mid \exists X\, \varphi \mid \forall x\, \gamma \ ,
\end{aligned}$$

where $d \in D$, $a \in \Sigma$, $x, y \in \mathcal{V}_0$, and $X \in \mathcal{V}_1$. The logics consists of three kinds of formulas: *boolean formulas* β, *almost boolean formulas* γ, and *wMSO-formulas* φ. Notice that conjunctions $\varphi \wedge \varphi$ and universal quantifications $\forall x\, \varphi$ are not part of the syntax. A *wMSO-sentence* is a wMSO-formula without free variables.

For a trace $t = (V, \sqsubseteq, \lambda)$ a *t-assignment* is a mapping $\alpha \colon \mathcal{V}_0 \cup \mathcal{V}_1 \to V \cup 2^V$ such that $\alpha(\mathcal{V}_0) \subseteq V$ and $\alpha(\mathcal{V}_1) \subseteq 2^V$. For $x \in \mathcal{V}_0$ and $v \in V$ we define a t-assignment $\alpha[x/v]$ by $\alpha[x/v](x) = v$ and $\alpha[x/v](y) = \alpha(y)$ for all $y \neq x$. Similarly, we define $\alpha[X/U]$ for $X \in \mathcal{V}_0$ and $U \subseteq V$.

The *semantics* $[\![\varphi]\!]$ of a wMSO-formula φ assigns an element of D to each pair (t, α) consisting of a trace $t = (V, \sqsubseteq, \lambda) \in \mathbb{T}(\Sigma)$ and a t-assignment α. This semantics $[\![\varphi]\!]$ is defined inductively on the structure of φ as follows: $[\![d]\!](t, \alpha) = d$

[1] Actually, this grammar describes the $\forall$- and strongly $\wedge$-restricted fragment of the weighted MSO logics from [10], but for simplicity of notation we use the term *weighted MSO logics* here. More on this can be found in the conclusions.

and

$$[\![x \le y]\!](t,\alpha) = \begin{cases} \mathbf{1} & \text{if } \alpha(x) \sqsubseteq \alpha(y) \\ \mathbf{0} & \text{otherwise} \end{cases} \qquad [\![\lambda(x) = a]\!](t,\alpha) = \begin{cases} \mathbf{1} & \text{if } \lambda(\alpha(x)) = a \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$[\![x \in X]\!](t,\alpha) = \begin{cases} \mathbf{1} & \text{if } \alpha(x) \in \alpha(X) \\ \mathbf{0} & \text{otherwise} \end{cases} \qquad [\![\neg\beta]\!](t,\alpha) = \begin{cases} \mathbf{1} & \text{if } [\![\beta]\!](t,\alpha) = \mathbf{0} \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$[\![\varphi \vee \psi]\!](t,\alpha) = [\![\varphi]\!](t,\alpha) + [\![\psi]\!](t,\alpha) \qquad [\![\varphi \wedge \psi]\!](t,\alpha) = [\![\varphi]\!](t,\alpha) \diamond [\![\psi]\!](t,\alpha)$$

$$[\![\exists x\, \varphi]\!](t,\alpha) = \sum_{v \in V} [\![\varphi]\!](t, \alpha[x/v]) \qquad [\![\forall x\, \varphi]\!](t,\alpha) = \mathrm{Val}\Big(\big([\![\varphi]\!](t, \alpha[x/v]) \big)_{v \in V} \Big)$$

$$[\![\exists X\, \varphi]\!](t,\alpha) = \sum_{U \subseteq V} [\![\varphi]\!](t, \alpha[X/U]) \quad [\![\forall X\, \varphi]\!](t,\alpha) = \mathrm{Val}\Big(\big([\![\varphi]\!](t, \alpha[X/U]) \big)_{U \subseteq V} \Big)$$

Since D is order independent, we do not need to specify in which order the elements of V and 2^V are enumerated in the definitions for universal quantification.

Remark 4.2. Every boolean formula β is an MSO-formula in the sense of Section 2.3. Hence, for any trace $t \in \mathbb{T}(\Sigma)$ and t-assigment α, there is an obvious meaning of $(t,\alpha) \models \beta$. By induction on β we can show that $[\![\beta]\!](t,\alpha) = \mathbf{1}$ if $(t,\alpha) \models \beta$ and $[\![\beta]\!](t,\alpha) = \mathbf{0}$ otherwise.

Notice that the boolean formulas include neither disjunction nor existential quantification. Even worse, for two boolean formulas β_1 and β_2 the value of $[\![\beta_1 \vee \beta_2]\!](t,\alpha)$ does not need to be $\mathbf{0}$ or $\mathbf{1}$. However, the following abbrevations provide the semantics we expect from such logical connectives:

$$\beta_1 \mathbin{\underline{\vee}} \beta_2 := \neg(\neg\beta_1 \wedge \neg\beta_2) \quad \text{and} \quad \underline{\exists} x\, \beta := \neg\forall x(\neg\beta) \ .$$

Finally, from the definition of the semantics one sees that $[\![\varphi]\!](t,\alpha)$ only depends on α for those variables which are free in φ. In particular, if φ is a sentence, then $[\![\varphi]\!](t,\alpha)$ does not depend on α at all, justifying the following definition:

Definition 4.3. *Let φ be a wMSO-sentence. The* trace semantics *of φ is the trace series $[\![\varphi]\!]_{\mathrm{t}} \colon \mathbb{T}(\Sigma) \to D$ defined by*

$$[\![\varphi]\!]_{\mathrm{t}}(t) = [\![\varphi]\!](t,\alpha) \ ,$$

where α is an arbitrary t-assigment.

The following theorem is the main result of this paper:

Theorem 4.4. *Let D be a regular order independent product valuation monoid and $S \colon \mathbb{T}(\Sigma) \to D$ a trace series. Then S is the behavior of some wACA if and only if S is the trace semantics of some wMSO-sentence.*

We prove this theorem using a technique introduced by Ebinger & Muscholl [13]. The main idea is to reduce to the corresponding characterization for word series by a translation of formulas. First, notice that the syntax of wMSO logics does not depend on the architecture $(\mathcal{L}, \mathcal{D})$ and recall that we can consider any word $w \in \Sigma^+$ as a trace for $|\mathcal{L}| = 1$. Thus, every wMSO-sentence φ also induces a word series $[\![\varphi]\!]_{\mathrm{w}} \colon \Sigma^+ \to D, w \mapsto [\![\varphi]\!](w, \alpha)$, for some arbitrary w-assignment α, which is called the *word semantics* of φ. This semantics coincides with that of Droste & Meinecke [10].

Theorem 4.5 (Droste & Meinecke [10]). *Let D be a regular product valuation monoid and $S \colon \Sigma^+ \to D$ a word series. Then S is the behavior of some wFA if and only if S is the word semantics of some wMSO-sentence.*

4.3 The Relationship between wMSO Logics for Traces and Words

This section is devoted to the proof of the following theorem which, in combination with Thms. 3.8 and 4.5, implies Thm. 4.4:

Theorem 4.6. *Let D be an order independent product valuation monoid and $S \colon \mathbb{T}(\Sigma) \to D$ a trace series. Then S is the trace semantics of some wMSO-sentence if and only if $\mathrm{trc}^{-1}(S)$ is the word semantics of some (other) wMSO-sentence.*

Proof (Sketch). First, assume $S = [\![\varphi]\!]_{\mathrm{t}}$ for some wMSO-sentence φ. The key idea is to construct a boolean first order formula $\beta(x, y)$ such that for any $w \in \Sigma^+$ and all w-assigments α we have $(w, \alpha) \models \beta(x, y)$ iff $(\mathrm{trc}(w), \alpha) \models x \leq y$. By replacing every subformula of the shape $x \leq y$ in φ by $\beta(x, y)$ we obtain a wMSO-sentence $\tilde{\varphi}$ with $\mathrm{trc}^{-1}(S) = [\![\tilde{\varphi}]\!]_{\mathrm{w}}$.

Therefore, we consider $w \in \Sigma^+$ and $\mathrm{trc}(w) = (V, \sqsubseteq, \lambda)$. By definition, for $i, j \in V$ we have $i \sqsubseteq j$ precisely if j is reachable from i in the graph (V, E) with $(i, j) \in E$ iff $i \leq j$ and $\big(\mathrm{loc} \circ \lambda(i), \mathrm{loc} \circ \lambda(j)\big) \in \mathcal{D}$. If the latter holds true, then there is also a path where all nodes, except maybe the first and the last one, are assigned to different locations. Moreover, since E is reflexive, we obtain $i \sqsubseteq j$ iff there is a path from i to j with exactly $|\mathcal{L}|$ edges. Obviously, E is definable in w by a boolean first order formula which does not depend on w. Thus, using $\underline{\exists}$ we can construct the desired formula $\beta(x, y)$.

For the converse direction, we consider once more a linear order $\preceq$ on $\mathcal{L}$ and derive the notion of a lexicographic normal form as in Section 3.3. We are interested in a boolean first order formula $\beta(x, y)$ which satisfies $(\mathrm{trc}(w), \alpha) \models \beta(x, y)$ iff $(w, \alpha) \models x \leq y$ for any $w \in \Sigma^+$ in lexicographic normal form and every w-assignment α. Replacing all occurrences of $x \leq y$ by $\beta(x, y)$ in a wMSO-sentence φ satisfying $[\![\varphi]\!]_{\mathrm{w}} = \mathrm{trc}^{-1}(S)$ yields a wMSO-sentence $\bar{\varphi}$ with $[\![\bar{\varphi}]\!]_{\mathrm{t}} = S$.

The key idea how to construct such a formula $\beta(x, y)$ is described in the proof of Thm. 6.1 ii) in [5]. Adopting another idea from [9] to our setting, we inductively define for each $n = 1, \ldots, |\mathcal{L}|$ a formula $\beta_n(x, y)$ as follows: $\beta_1(x, y) = x \leq y$ and

$$\beta_n(x, y) = x \leq y \ \underline{\vee} \ \underline{\exists} z \left(\mathrm{loc}(x) \prec \mathrm{loc}(z) \wedge \neg \beta_{n-1}(z, x) \wedge z \leq y\right) \ ,$$

where $\mathrm{loc}(x) \prec \mathrm{loc}(z)$ abbreviates $\bigvee_{a,b\in\Sigma,\mathrm{loc}(a)\prec\mathrm{loc}(b)} \big(\lambda(x) = a \wedge \lambda(z) = b\big)$. Then, $\beta_{|\mathcal{L}|}(x, y)$ is the desired formula. □

5 Discussion

Depending on the properties of the pv-monoid, Droste & Meinecke [10] characterized the expressive powers of wFAs by one of three fragments of their wMSO logics. The logics in this paper corresponds to their $\forall$- and strongly $\wedge$-restricted formulas. The other two fragments are the $\forall$- and $\wedge$-restricted formulas and the $\forall$- and commutatively $\wedge$-restricted formulas, which are both less restrictive concerning the conjunction of wMSO-formulas. The former fragment characterizes the expressiveness of wFAs if the pv-monoid has a property called left-distributivity, whereas the latter gives a characterization in case of cc-valuation semirings. Since both fragments are closed under the substitutions made in the proof of Thm. 4.6, these two results can be extended to the trace level, yielding an analogue of Thm. 4.4, without any further difficulties, cf. [16].

Like many other results, which were lifted in a similar way, this showed that traces form a robust generalization of words for modeling concurrency. Furthermore, in the unweighted situation many results for infinite words were extended to infinite traces. However, it is still not clear whether similar extensions are possible in a weighted setting, neither for (complete) semirings nor ω-valuation monoids. Since there is no suitable (lexicographic) normal form for infinite traces, new techniques will be necessary for proving analogues of Thms. 3.8 and 4.6, as this was already the case in the boolean setting, cf. [13].

Finally, message sequence charts (MSCs) provide a more generic framework for modeling distributed systems. Finite automata for MSCs and MSO logics over such structures were proven to be equally expressive, in an unweighted [15] as well as in a semiring weighted setting [2]. This naturally raises the question whether we can generalize these results to a situation with weights from a valuation monoid.

References

1. 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)
2. 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)
3. Büchi, J.R.: Weak second-order arithmetic and finite automata. Zeitschr. f. math. Logik und Grundlagen d. Math. 6, 66–92 (1960)
4. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 385–400. Springer, Heidelberg (2008)
5. Diekert, V., Métivier, Y.: Partial commutation and traces. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 457–533. Springer, Heidelberg (1997)

6. Diekert, V., Rozenberg, G. (eds.): The Book of Traces. World Scienific Publishing, Singapore (1995)
7. Droste, M., Gastin, P.: Weighted automata and weighted logics. In: [8], ch. 5, pp. 175–211
8. Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata. Springer, Heidelberg (2009)
9. Droste, M., Kuske, D.: Languages and logical definability in concurrency monoids. In: Kleine Büning, H. (ed.) CSL 1995. LNCS, vol. 1092, pp. 233–251. Springer, Heidelberg (1996)
10. Droste, M., Meinecke, I.: Describing average- and longtime-behavior by weighted MSO logics. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 537–548. Springer, Heidelberg (2010)
11. Droste, M., Rahonis, G.: Weighted automata and weighted logics with discounting. Theoretical Computer Science 410, 3481–3494 (2009)
12. Droste, M., Vogler, H.: Kleene and Büchi theorems for weighted automata and multi-valued logics over arbitrary bounded lattices. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 160–172. Springer, Heidelberg (2010)
13. Ebinger, W., Muscholl, A.: Logical definability on infinite traces. Theoretical Computer Science 154, 67–84 (1996)
14. Fichtner, I., Kuske, D., Meinecke, I.: Traces, series-parallel posets, and pictures: A weighted study. In: [8], ch. 10, pp. 397–441
15. Genest, B., Kuske, D., Muscholl, A.: On communicating automata with bounded channels. Fundamenta Informaticae 80, 147–167 (2007)
16. Huschenbett, M.: Models for quantitative distributed systems and multi-valued logics. Master's thesis, Universität Leipzig (2010)
17. Huschenbett, M.: A Kleene-Schützenberger theorem for trace series over bounded lattices (2011) (submitted journal version)
18. Kupferman, O., Lustig, Y.: Lattice automata. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 199–213. Springer, Heidelberg (2007)
19. Kuske, D.: Weighted asynchronous cellular automata. Theoretical Computer Science 374, 127–148 (2007)
20. 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)
21. Schützenberger, M.P.: On the definition of a family of automata. Information and Control 4, 245–270 (1961)
22. Thomas, W.: On logical definability of trace languages. In: ASMICS-Workshop "Free Partially Commutative Monoids". Rep. TUM-I9002, TU München, pp. 172–182 (1990)
23. Zielonka, W.: Asynchronous automata. In: [6], ch. 7, pp. 205–247

A Local Greibach Normal Form for Hyperedge Replacement Grammars

Christina Jansen, Jonathan Heinen, Joost-Pieter Katoen, and Thomas Noll

Software Modeling and Verification Group,
RWTH Aachen University, Germany
http://moves.rwth-aachen.de/

Abstract. Heap-based data structures play an important role in modern programming concepts. However standard verification algorithms cannot cope with infinite state spaces as induced by these structures. A common approach to solve this problem is to apply abstraction techniques. Hyperedge replacement grammars provide a promising technique for heap abstraction as their production rules can be used to partially abstract and concretise heap structures. To support the required concretisations, we introduce a normal form for hyperedge replacement grammars as a generalisation of the Greibach Normal Form for string grammars and the adapted construction.

1 Introduction

The verification of programs that use pointers to implement dynamic data structures is a highly challenging and important task, as memory leaks or dereferencing null pointers can cause great damage especially when software reliability is at stake. As objects can be created at runtime, dynamic data structures induce a possibly infinite state space and therefore cannot be handled by standard verification algorithms. Abstraction techniques such as shape analysis [18] that yield finite representations for these data structures are a common way to address this problem. Other popular techniques are based on separation logic [10,14] (duality with hyperedge replacement grammars is observed in [4]) and regular tree automata [2].

Our approach is to verify pointer-manipulating programs using *hyperedge replacement grammars* (HRGs) [7]. Dynamic memory allocation and destructive updates are transcribed on hypergraphs representing heaps. Production rules of the HRG reflect employed data structures. Terminal edges model variables and pointers, whereas nonterminal edges represent abstract parts of a heap. Thus, hypergraphs are heap configurations that are partially concrete and partially abstract, such that heap fragments relevant for the current program state are concrete while a finite heap representation is achieved. Concretisation of abstract heap fragments is obtained by classical forward grammar rule application, abstraction by backward application. This use of HRGs has been first proposed by us in [15]; tool support and the successful verification of the Deutsch-Schorr-Waite tree traversal algorithm have been reported in [8]. Graph grammars for

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 323–335, 2011.

heap verification have also been advocated in, e.g. [18,12,11,1,13]. Primarily this yields a rather intuitive and easy-to-grasp heap modelling approach, where no abstract program semantics is needed. In particular, it avoids a (often tedious) formal proof how this relates to a concrete semantics, see e.g. [3]. Pointer statements such as assignments and object creation are realised on concrete subgraphs only. Thus if pointer assignments "move" program variables too close to abstract graph fragments, a local concretisation is carried out. To enable this, our heap abstraction HRGs are required to be in a specific form that is akin to the well-known Greibach normal form (GNF) for string grammars.

In [15] we proposed to use the GNF introduced in [6] for this purpose, restricting manageable data structures to ones where each object is referenced by a bounded number of objects. This paper defines a normal form for HRGs as a generalisation of the original GNF for string grammars. Compared to [6], it allows us to model data structures without restrictions to referencing and in general results in grammars with less and smaller production rules. Furthermore our normal form allows to adapt the well-known GNF transformation algorithm for string to graph grammars. We present the adapted construction and its correctness in Section 3. In Section 2, the above concepts are formalised, we consider a notion of typing for HRGs, and provide all relevant theoretical results. The full version of this paper[1] contains the omitted proofs.

2 Preliminaries

Given a set S, $S^\star$ is the set of all finite sequences (strings) over S including the empty sequence ε. For $s \in S^\star$, the length of s is denoted by $|s|$, the set of all elements of s is written as $[s]$, and by $s(i)$ we refer to the i-th element of s. Given a tuple $t = (A, B, C, \dots)$ we write A_t, B_t etc. for the components if their names are clear from the context. Function $f \restriction S$ is the restriction of f to S. Function $f : A \to B$ is lifted to sets $f : 2^A \to 2^B$ and to sequences $f : A^\star \to B^\star$ by point-wise application. We denote the identity function on a set S by id_S.

2.1 Heaps and Hypergraphs

The principal idea behind our *Juggrnaut* framework [8,15] is to represent (abstract) heaps as hypergraphs.

Definition 1 (Hypergraph). *Let Σ be a finite ranked alphabet where $rk : \Sigma \to \mathbb{N}$ assigns to each symbol $a \in \Sigma$ its rank $rk(a)$. A* (labelled) hypergraph *over Σ is a tuple $H = (V, E, att, lab, ext)$ where V is a set of vertices and E a set of hyperedges, $att : E \to V^\star$ maps each hyperedge to a sequence of attached vertices, $lab : E \to \Sigma$ is a hyperedge-labelling function, and $ext \in V^\star$ a (possibly empty) sequence of pairwise distinct external vertices.*

[1] Technical Report AIB-2011-04 available from `http://aib.informatik.rwth-aachen.de`

For $e \in E$, we require $|att(e)| = rk(lab(e))$ and let $rk(e) = rk(lab(e))$. The set of all hypergraphs over Σ is denoted by HG_Σ.

Hypergraphs are graphs with edges as proper objects which are not restricted to connect exactly two vertices. Two hypergraphs are *isomorphic* if they are identical modulo renaming of vertices and hyperedges. We will not distinguish between isomorphic hypergraphs.

To set up an intuitive heap representation by hypergraphs we consider finite ranked alphabets $\Sigma = Var_\Sigma \uplus Sel_\Sigma$, where Var_Σ is a set of variables, each of rank one and Sel_Σ a set of *selectors* each of rank two. We model heaps as hypergraphs over Σ. Objects are represented by vertices, and pointer variables and selectors by edges connected to the corresponding object(s) where selector edges are understood as pointers from the first attached object to the second one. To represent abstract parts of the heap, we use nonterminal edges i.e. with labels from an additional set of *nonterminals* N of arbitrary rank (and we let $\Sigma_N = \Sigma \cup N$). The connections between hyperedges and vertices are called *tentacles*.

Example 1. A typical implementation of a doubly-linked list consists of a sequence of list elements connected by next and previous pointers and an additional list object containing pointers to the *head* and *tail* of the list. We consider an extended implementation where each list element features an additional pointer to the corresponding list object. Fig. 1 depicts a hypergraph representation of a doubly-linked list. The three circles are vertices representing objects on the heap. Tentacles are labeled with their ordinal number. For the sake of readability, selectors (*head* and *tail*) are depicted as directed edges. A variable named *list* referencing the list object is represented as an edge of rank one. The L-labeled box represents a nonterminal edge of rank three indicating an abstracted doubly-linked list between the first and second attached vertex, where each abstracted list element has a pointer to the list object. In Section 2.2 we will see how abstract structures are defined.

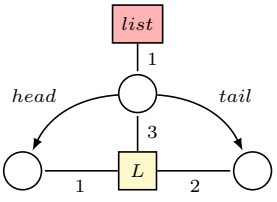


Fig. 1. Heap as hypergraph

Note that not every hypergraph represents a feasible heap: it is necessary that each variable and each a-selector (for every $a \in Sel_\Sigma$) refers to at most one object. Therefore we introduce *heap configurations* as restricted hypergraphs:

Definition 2 (Heap Configuration). $H \in HG_{\Sigma_N}$ *is a* heap configuration *if:*

1. $\forall a \in Sel_\Sigma, v \in V_H : |\{e \in E_H \mid att(e)(1) = v, lab(e) = a\}| \leq 1$, *and*
2. $\forall a \in Var_\Sigma : |\{e \in E_H \mid lab(e) = a\}| \leq 1$

We denote the set of all heap configurations over Σ_N by HC_{Σ_N}. If a heap configuration contains nonterminals it is abstract, *otherwise* concrete.

2.2 Data Structures and Hyperedge Replacement Grammars

As pointed out earlier, both abstraction and concretisation are transformation steps on the hypergraph representation of the heap. We use *hyperedge replacement grammars* for this purpose, implementing abstraction and concretisation as backward and forward application of replacement rules respectively.

Definition 3 (Hyperedge Replacement Grammar). *A* hyperedge replacement grammar *(HRG) over an alphabet Σ_N is a set of production rules of the form $X \to H$, with $X \in N$ and $H \in HG_{\Sigma_N}$ where $|ext_H| = rk(X)$. We denote the set of hyperedge replacement grammars over Σ_N by HRG_{Σ_N}.*

Example 2. Fig. 2 specifies an HRG for doubly-linked lists. n, p stand for *next* and *previous* while l is the pointer to the corresponding list object shared by all elements. *head* and *tail* (cf. Fig. 1) do not occur as they are not abstracted.

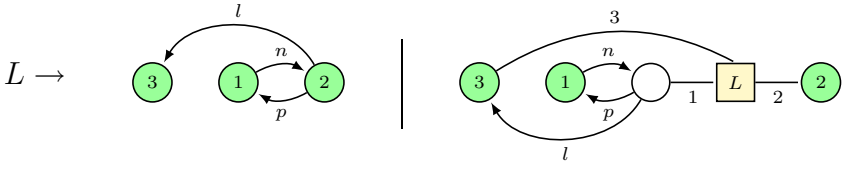


Fig. 2. A grammar for doubly-linked lists

The HRG derivation steps are defined through hyperedge replacement.

Definition 4 (Hyperedge Replacement). *Let $H, K \in HG_{\Sigma_N}, e \in E_H$ a nonterminal edge with $rk(e) = |ext_K|$. W.l.o.g. we assume that $V_H \cap V_K = E_H \cap E_K = \emptyset$ (otherwise the components in K have to be renamed). The* substitution of e by K, *$H[K/e] = J \in HG_{\Sigma_N}$, is defined by:*

$$V_J = V_H \cup (V_K \setminus [ext_K]) \qquad E_J = (E_H \setminus \{e\}) \cup E_K$$
$$lab_J = (lab_H \restriction (E_H \setminus \{e\})) \cup lab_K \qquad ext_J = ext_H$$
$$att_J = att_H \restriction (E_H \setminus \{e\}) \cup mod \circ att_K$$

with $mod = \mathrm{id}_{V_J} \cup \{[ext_K(1) \mapsto att_H(e)(1), \ldots, ext_K(rk(e)) \mapsto att_H(e)(rk(e))\}$.

Example 3. Reconsider the hypergraph H of Fig. 1 as well as the second rule in Fig. 2, denoted by $L \to K$. In H we replace the nonterminal edge e labelled with L by K, which yields $H[K/e]$. This is possible since $rk(L) = |ext_K| = 3$. Replacing the L-edge we merge external node $ext(1)$ with the node connected to the first tentacle, $ext(2)$ with the second and so on. The resulting graph is shown in Fig. 3.

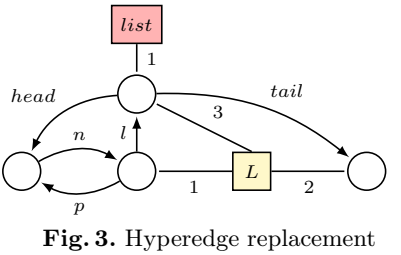


Fig. 3. Hyperedge replacement

Definition 5 (HRG Derivation). *Let* $G \in HRG_{\Sigma_N}$, $H, H' \in HG_{\Sigma_N}$, $p = X \to K \in G$ *and* $e \in E_H$ *with* $lab(e) = X$. H derives H' by p *iff* H' *is isomorphic to* $H[K/e]$. $H \overset{e,p}{\Longrightarrow} H'$ *refers to this derivation. Let* $H \overset{G}{\Longrightarrow} H'$ *if* $H \overset{e,p}{\Longrightarrow} H'$ *for some* $e \in E_H, p \in G$. *If* G *is clear from the context* $\Rightarrow^\star$ *denotes the reflexive-transitive closure.*

The definition of HRGs does not include a particular starting graph. Instead, it is introduced as a parameter in the definition of the generated language.

Definition 6 (Language of a HRG). *Let* $G \in HRG_{\Sigma_N}$ *and* $H \in HG_{\Sigma_N}$. $L_G(H) = \{K \in HG_\Sigma \mid H \Rightarrow^\star K\}$ *is the* language *generated from* H *using* G.

We write $L(H)$ instead of $L_G(H)$ if G is clear from the context. To define the language of a nonterminal we introduce the notion of a *handle* which is a hypergraph consisting of a single hyperedge attached to external vertices only.

Definition 7 (Handle). *Given* $X \in N$ *with* $rk(X) = n$, *an* X-handle *is the hypergraph* $X^\bullet = (\{v_1, \ldots, v_n\}, \{e\}, [e \mapsto v_1 \ldots v_n], [e \mapsto X], v_1 \ldots v_n) \in HG_\Sigma$.

Thus $L(X^\bullet)$ is the language induced by nonterminal X. For $H \in HC_{\Sigma_N}$, $L(H)$ denotes the set of corresponding concrete heap configurations. Note that it is not guaranteed that $L(H) \subseteq HC_\Sigma$, i.e., $L(H)$ can contain invalid heaps.

Definition 8 (Data Structure Grammar). $G \in HRG_{\Sigma_N}$ *is called a* data structure grammar *(DSG) over* Σ_N *if* $\forall X \in N : L(X) \subseteq HC_{Sel_\Sigma}$. *We denote the set of all data structure grammars over* Σ_N *by* DSG_{Σ_N}.

Theorem 1. *It is decidable whether a given HRG is a DSG.*

Later we will see that DSGs are still too permissive for describing heap abstraction and concretisation. In Section 2.4, we will therefore refine this definition to so-called *heap abstraction grammars*.

2.3 Execution of Program Statements

The overall goal of our framework is to reduce the large or even infinite program state spaces induced by dynamic data structures. To this aim, heap configurations are partially abstracted by backward application of replacement rules. As long as pointer manipulations are applied to concrete parts of the heap, they can be realised one-to-one. In order to avoid the need of defining an abstract semantics we avoid manipulations on abstract parts by applying local concretisation steps before. As pointers are not dereferenced backwards, restricting the dereferencing depth to one reduces the (potentially) affected parts of the heap to those nodes that are directly reachable from variable nodes by *outgoing edges*.

Definition 9 (Outgoing Edges). *Let* $H \in HC_\Sigma$, $v \in V_H$. *The set of* outgoing edges *at vertex* v *in* H *is defined as:* $out(v) = \{e \in E_H \mid att(e)(1) = v\}$.

In abstract heap configurations, variable vertices can have abstracted outgoing edges derivable at connected nonterminal tentacles.

Definition 10 (Tentacle). *Let $X \in N$, $i \in [1, rk(X)]$, the pair (X, i) is a tentacle. (X, i) is a* reduction tentacle *if, for all $H \in L(X^\bullet)$, $out(ext_H(i)) = \emptyset$.*

Example 4. Reconsider the grammar of Fig. 2. $(L, 3)$ is a reduction tentacle, as no outgoing terminal edges are derivable at external vertex 3.

A heap configuration is *inadmissible* if variable nodes are connected to non-reduction tentacles.

Definition 11 (Admissibility). *For $H \in HC_{\Sigma_N}$, $e \in E_H$, and $i \in \mathbb{N}$, the pair (e, i) is called a* violation point *if $(lab(e), i)$ is not a reduction tentacle and $\exists e' \in E_H : lab(e') \in Var_\Sigma \wedge att(e')(1) = att(e)(1)$. H is called* admissible *if it contains no violation point, and* inadmissible *otherwise.*

Heap manipulations may introduce violation points. This inadmissibility can be resolved by *concretisation*, that is, by considering all possible replacements of the corresponding edge. Notice that concretisation generally entails nondeterminism, viz. one successor state for each applicable replacement rule.

Example 5. On the left side of Fig. 4, an inadmissible heap configuration is depicted. While its *list* object is only connected to concrete edges and reduction tentacle $(L, 3)$, there is a violation point at the shaded $(L, 1)$-tentacle. Concretisation by applying both production rules of the grammar given in Fig. 2 results in the two admissible configurations on the right.

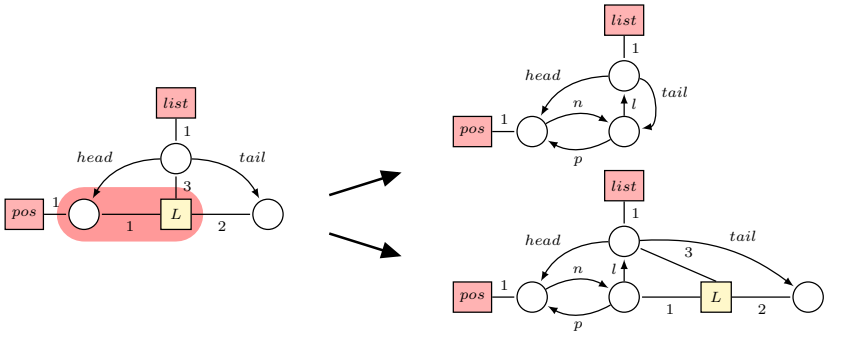


Fig. 4. Concretisation of inadmissible heap configurations

Theorem 2. *For $G \in HRG_{\Sigma_N}, H \in HC_{\Sigma_N}, e \in E_H$, $X \in N$ with $X = lab(e)$:*

$$L(H) = \bigcup_{\forall\, X \to K \in G} L(H[K/e])$$

This theorem follows directly from the confluence property of HRGs [16].

While concretisation is realised by standard, forward application of production rules, abstraction is handled by *backward* rule application. Thus we call $H \in HC_{\Sigma_N}$ an abstraction of $H' \in HC_{\Sigma_N}$ if $H \Rightarrow^\star H'$. Obviously $L(H') \subseteq L(H)$ and therefore abstraction leads to an over-approximation of the state space. The latter fact together with Theorem 2 yields the soundness of our heap abstraction approach. We apply the principle *"Abstract if possible – Concretise when necessary"* to obtain the best possible results in terms of the size of the resulting state space.

2.4 Heap Abstraction Grammars

As mentioned before, DSGs are not sufficient in our setting. Additional restrictions that ensure termination and correctness of the abstraction technique are listed and discussed in detail below.

Definition 12 (Heap Abstraction Grammar). $G \in DSG_{\Sigma_N}$ *is a* heap abstraction grammar (HAG) *over* Σ_N *if:*

(1)	G *is productive*	$\forall X \in N : L(X^\bullet) \neq \emptyset$
(2)	G *is increasing*	$\forall X \to H \in G : \lvert E_H \rvert \leq 1 \Rightarrow H \in HG_\Sigma$
(3)	G *is typed*	*see below*
(4)	G *is locally concretisable*	*see below*

We denote the set of all heap abstraction grammars over Σ_N *by* HAG_{Σ_N}.

Productivity (1) is a well-known notion from string grammars, ensuring that each abstract configuration represents at least one concrete configuration. A rule is *increasing* if its right-hand side is terminal or "bigger" than the corresponding handle. *Increasing grammars* (2) guarantee termination of abstraction, as applying rules backwards reduces the size of the heap representation. We call a grammar *typed* (3) if every concrete vertex has a well-defined type as induced by the set of outgoing edges.

Definition 13 (Typedness). $G \in DSG_{\Sigma_N}$ *is* typed *if* $\forall X \in N, i \in [1, rk(X)]$, $\exists type(X,i) \subseteq \Sigma : \forall H \in L(X^\bullet) : \ type(X,i) = out_H(ext_H(i))$.

As DSGs restrict the number of outgoing edges to a finite set of selectors, every untyped nonterminal can be replaced by a typed one for each derivable type.

Theorem 3. *It is decidable whether a HRG is typed. For any untyped DSG an equivalent typed DSG can be constructed.*

Local concretisability (4) ensures that admissibility of a heap configuration can be established within one concretisation step.

Example 6. Fig. 5(left) reconsiders the original heap configuration given in Fig. 4 with variable *pos* set to the tail of the list. This leads to an inadmissible configuration and two corresponding concretisations, cf. Fig. 5(right). While the first concretisation is an admissible configuration, the second one remains inadmissible. Successive concretisations would lead to further inadmissible configurations.

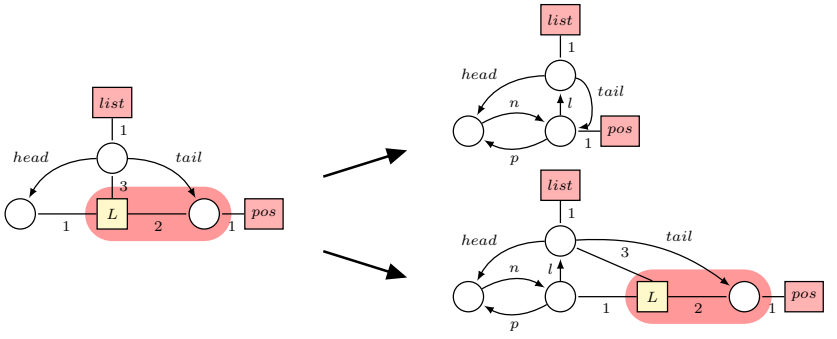


Fig. 5. Inadmissible concretisation

Ignoring the second rule yields termination but is unsound as Theorem 2 requires concretisations by every corresponding rule.

Let $G \in HRG_{\Sigma_N}$ with $p = X \to H \in G$. $(X, i) \to_p (Y, j)$ denotes that (X, i) can be replaced by (Y, j), i.e. $ext_H(i)$ is connected to a (Y, j)-tentacle.

Example 7. For p the second rule of Fig. 2 it holds that $(L, 3) \to_p (L, 3)$ and $(L, 3) \to_p (l, 2)$ denoting an incoming l-selector edge.

For simplicity we use G^X as the set of all rules $X \to H \in G$ and $\overline{G^X} = G \setminus G^X$.

Definition 14 (Local Concretisability). *$G \in HRG_{\Sigma_N}$ is locally concretisable if for all $X \in N$ there exist grammars $G_1^X, \cdots, G_{rk(X)}^X \subseteq G^X$ such that:*

1. $\forall i \in [1, rk(X)]$, $L_{G_i^X \cup \overline{G^X}}(X^\bullet) = L_G(X^\bullet)$
2. $\forall i \in [1, rk(X)]$, $a \in type(X, i), p \in G_i^X : (X, i) \to_p (a, 1)$

Theorem 4. *For each DSG an equivalent HAG can be constructed.*

Property (1) can be achieved easily by removing non-productive rules, typedness (3) by introducing new, typed nonterminals. The *Local Greibach Normal Form* presented in the next section ensures properties (2) and (4), cf. Theorem 6.

3 Local Greibach Normal Form

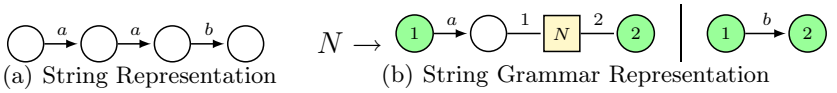
The Greibach Normal Form (GNF) for string grammars restricts production rules to the form $X \to aN_1 \dots N_k$ such that using left derivation only a word $w \in \Sigma^n N^\star$ is derived after n derivation steps. Thus terminal words are constructed from left to right extending a terminal prefix by one symbol each step.

Up to now the normal form given in [6] is the only notion of a GNF for graph grammars widely known and accepted. In contrast to [6], where graph derivation is from outside to inside, we consider a generalisation of GNF for strings where one-sided derivation is of interest.

Definition 15 (Local Greibach Normal Form). *$G \in DSG_{\Sigma_N}$ is in* local Greibach Normal Form (LGNF) *if for every non-reduction tentacle (X, i) there exists $G_i^X \subseteq G^X$ with:*

1. $L_{G_i^X \cup \overline{G^X}}(X^\bullet) = L_G(X^\bullet)$
2. $\forall p \in G_i^X \colon (X, i) \to_p (Y, j)$ *implies* $Y \in \Sigma$ *or* (Y, i) *is a reduction tentacle.*

Example 8. Strings can be uniquely represented by HGs containing chains of terminal edges only, production rules can be translated to HRGs analogously [7]. In Fig. 6, graph representations for word $w = aab$ and string grammar $X \to aX \mid b$ are given. As nonterminals are of rank two and $(X, 2)$ is a reduction tentacle for each nonterminal X exactly one G_i^X remains namely G_1^X containing the string GNF rules.



(a) String Representation (b) String Grammar Representation

Fig. 6. String Graphs

The LGNF for DSG G is established by merging corresponding sets G_i^X, constructed in four steps along the lines of the GNF construction for string grammars: Assume a total order on the non-reduction tentacles $T_1, \ldots, T_n$ over N. For increasing $i \in [1, n]$ (1) every rule p, such that $T_i \to_p T_j$ with $j < i$, is eliminated, then (2) local recursion is removed. In a next step (3) all rules are brought into LGNF using simple hyperedge replacements. Finally (4) rules for nonterminals introduced during the construction are transformed. In the following we guide through the four construction steps and define them in detail.

For each non-reduction (X, i)-tentacle we initialise the set $G_i^X = G^X$ and we define an ordering $T_1, \ldots, T_n$ on non-reduction tentacles.

Step 1: Elimination of rules. We first eliminate the rules $p = X \to H \in G_i^X$ with $(X, i) = T_k \to_p T_l$, $l < k$. Let $T_l = (Y, j)$, $e \in E_H$ with $lab(e) = Y$ and $att(e)(j) = ext(i)$. Then p is replaced by the set $\{X \to H[K/e] \mid Y \to K \in G_j^Y\}$. Theorem 2 states that this procedure does not change the language.

Lemma 1. *Let $G \in HRG_{\Sigma_N}$. For a grammar G' originating from G by eliminating a production rule, it holds that $L_G(H) = L_{G'}(H)$ for all $H \in HG_{\Sigma_N}$.*

Note that G_j^Y does not contain any rule p with $T_l \to_p T_m$, $m < l$, as they are removed before. Thus after finitely many steps all corresponding rules are eliminated.

Step 2: Elimination of local recursion. After step 1, rules p with $T_i \to_p T_i$ remain.

Definition 16 (Local Recursion). *Let $G \in \mathrm{HRG}_{\Sigma_N}$, $X \in N$, $i \in [1, rk(X)]$. G is* locally recursive *at (X, i) if there exists a rule p with $(X, i) \to_p (X, i)$.*

Let $G_r^X \subseteq G_i^X$ be the set of all rules locally recursive at (X, i). To remove local recursion in $p = X \to H \in G_i^X$ we introduce a new nonterminal B'_j, a recursive rule $B'_j \to J_n$ and an exit rule $B'_j \to J_t$. J_t corresponds to graph H, where edge e causing local recursion is removed. We also remove all external nodes singly connected to e ($V_R = \{v \in [ext_H] \mid \forall e' \in E_H : v \in [att_H(e')] \Rightarrow e = e'\}$). By removing border-edge e, its previously connected internal nodes move to the border and get external. Thus $V_{ext} = ([att_{H_j}(e)] \cup [ext_{H_j}]) \cap V_{J_t}$ is the set of arbitrary ordered external nodes.

J_n extends J_t by an additional edge e' labelled by B'_j. As this edge models the structure from the other side, it is connected to the remaining external nodes of H that will not be external any longer. Note that the rank of B'_j is already given by J_t and therefore introduced gaps in the external sequence are filled by new external nodes that are connected to edge e' ($\mathit{fill}(i) = ext_{J_t}(i)$ *if* $ext_{J_t}(i) \in att_H(e)$, *a new node otherwise*). To build up the same structure as (X, i) "from the other side" edge e' has to be plugged in correctly:

$$plug(g) = \begin{cases} ext_{H_j}(y) & \text{if } ext_{J_n}(g) \in V_{J_t} \\ ext_{J_t}(g) & \text{otherwise} \end{cases}, \text{ with } att_{H_j}(e)(y) = ext_{J_t}(g).$$

$B'_j \to J_t$ with:

$$\begin{aligned} V_{J_t} &= V_{H_j} \setminus V_R(e) \\ E_{J_t} &= E_{H_j} \setminus \{e\} \\ lab_{J_t} &= lab_{H_j} \restriction E_{J_t} \\ att_{J_t} &= att_{H_j} \restriction E_{J_t} \\ ext_{J_t} &\in V_{ext}^* \end{aligned}$$

$B'_j \to J_n$ with:

$$\begin{aligned} V_{J_n} &= V_{J_t} \cup [ext_{J_n}] \\ E_{J_n} &= E_{J_t} \cup \{e'\} \\ lab_{J_n} &= lab_{J_t} \cup \{e' \to B'_j\} \\ att_{J_n} &= att_{J_t} \cup \{e' \to plug\} \\ ext_{J_n} &= fill \end{aligned}$$

Newly introduced nonterminals are collected in a set N', $\Sigma' = \Sigma \cup N'$. For mirrored derivations, each terminal rule in G_i^X can be the initial one thus we add a copy, extended by an additional B'_j-edge, the G_i^X.

Lemma 2. *Let $G \in DSG_{\Sigma_N}$. For the grammar G_i^X over $\Sigma' = \Sigma \cup \{ B' \mid B'$ newly introduced nonterminal $\}$ originating from grammar G by eliminating the (X, i)-local recursion as described above, it holds that $L_G(H) = L_{G_i^X}(H)$ for all $H \in HG_{\Sigma_N}$.*

Example 9. The doubly-linked list HRG with production rules $L \to H \mid J$ given in Fig. 2 is locally recursive at $(L, 2)$. We introduce nonterminal B' and the rules $B' \to J_t \mid J_n$, cf. Fig. 7. The terminal right-hand side J_t corresponds to J with removed L-edge and attached external node $ext(2)$. J_n is a copy of J_t with an additional B'-edge and replaced external node $ext(1)$. Intuitively, local recursion is eliminated by introducing new production rules which allow "mirrored" derivations.

Step 3: Generation of Greibach rules. Starting at the highest order tentacle, for each G_i^X LGNF can be established by elimination of every non-reduction (Y,j)-tentacle connected to external node i. That is because (Y,j) is of higher order and thus already in LGNF.

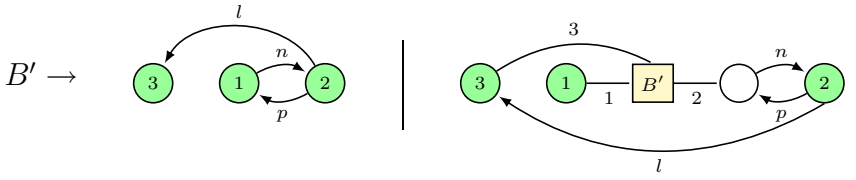


Fig. 7. Doubly-Linked Lists: $G_2^{B'}$

Step 4: Transforming new nonterminals to GNF . In the final step we apply steps one to three to the newly added nonterminals from step two. Obviously further nonterminals could be introduced. To avoid nontermination we merge nonterminals if the right-hand sides of the corresponding production rules are equal.

Theorem 5. *After finite many steps a nonterminal can be merged, thus the construction of LGNF terminates.*

Note that step 2 is nondeterministic as the order on external nodes can be chosen arbitrarily. Unnecessary steps introduced by unsuitable orders can be avoided by considering permutations of external nodes isomorphic. If we reach an isomorphic nonterminal after arbitrary many steps all nonterminals in between represent the same language and thus can be merged as long as the rank of the nonterminals permit this.

Example 10. Applying steps 1 to 3 to $G_1^{B'}$ results in a new nonterminal B'' isomorphic to L. Thus we can merge L with B'' and even B' as the latter occurred between the formers.

Theorem 6. *Any DSG can be transformed into an equivalent DSG in LGNF.*

Note that LGNF directly implies the local concretisable property of HAGs. It additionally ensures the increasingness property, as every production rule belongs to at least one G_i^X composed by rules with terminal edges at external node $ext(i)$.

Lemma 3. *Each DSG in LGNF is increasing.*

While we restrict the normalisable grammars to DSGs here the procedure can easily be lifted to arbitrary bounded HRGs.

4 Related Work

The idea of using HRGs for verifying heap manipulating programs was proposed in [8,15]. No technique for transforming a given HRG into a suitable grammar was provided though, instead using the GNF construction from [6] was proposed, allowing hypergraphs to be concretised from outer to inner. This generally results in more and larger rules compared to our LGNF approach, as LGNF generalises GNF, i.e. every resulting grammar from [6] is in LGNF. Considering the example grammar for binary trees with linked frontier [6], its GNF consists of 135

production rules whereas our local Greibach construction results in 36 rules. In number of nodes and edges our largest rule is half the size of the normalised example rule given in [6]. Note that [6] restricts the input grammars to bounded degree ones, i.e. those allowing only boundedly many references to each object, excluding for instance rooted grammars like the one given in Fig. 2. Adapting the construction to HAGs without additional restrictions leads to a complex construction with poor results.

A further GNF approach for HRGs can be found in [5]. The basic idea is to use the string GNF construction on a linearisation of the considered HRG. It is however not clear how to re-obtain the HRG from the resulting linearisation. Further normal form constructions addressing node replacement can be found in [17] and [9].

5 Conclusion

This paper presented the theoretical underpinnings of heap abstraction using hyperedge replacement grammars (HRGs). We showed that concretisation and abstraction are naturally obtained by forward and backward rule application respectively. The main contribution is a Greibach normal form (GNF) together with a procedure to transform an HRG into (local) GNF.

Future work will concentrate on advancing our prototypical tool [8], incremental LGNF construction, and on the automated synthesis of heap abstraction grammars from program executions.

References

1. Bakewell, A., Plump, D., Runciman, C.: Checking the shape safety of pointer manipulations. In: Berghammer, R., Möller, B., Struth, G. (eds.) RelMiCS 2003. LNCS, vol. 3051, pp. 48–61. Springer, Heidelberg (2004)
2. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract Regular Tree Model Checking. ENTCS 149, 37–48 (2006)
3. Distefano, D., Katoen, J.P., Rensink, A.: Safety and liveness in concurrent pointer programs. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2005. LNCS, vol. 4111, pp. 280–312. Springer, Heidelberg (2006)
4. Dodds, M.: From Separation Logic to Hyperedge Replacement and Back. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 484–486. Springer, Heidelberg (2008)
5. Dumitrescu, S.: Several Aspects of Context Freeness for Hyperedge Replacement Grammars. W. Trans. on Comp. 7, 1594–1604 (2008)
6. Engelfriet, J.: A Greibach Normal Form for Context-free Graph Grammars. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 138–149. Springer, Heidelberg (1992)
7. Habel, A.: Hyperedge Replacement: Grammars and Languages. Springer, New York (1992)
8. Heinen, J., Noll, T., Rieger, S.: Juggrnaut: Graph Grammar Abstraction for Unbounded Heap Structures. In: TTSS 2009 (2009) (to be published in ENTCS)

9. Klempien-Hinrichs, R.: Normal Forms for Context-Free Node-Rewriting Hypergraph Grammars. Math. Structures in Comp. Sci. 12, 135–148 (2002)
10. O'Hearn, P.W., Hongseok, Y., Reynolds, J.C.: Separation and Information Hiding. In: POPL 2004, vol. 39, pp. 268–280 (2004)
11. Rensink, A.: Canonical Graph Shapes. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 401–415. Springer, Heidelberg (2004)
12. Rensink, A.: Summary from the Outside In. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 486–488. Springer, Heidelberg (2004)
13. Rensink, A., Distefano, D.: Abstract Graph Transformation. In: SVV 2005, vol. 157, pp. 39–59 (2006)
14. Reynolds, J.C.: Separation Logic: A Logic for Shared Mutable Data Structures. In: LICS 2002, pp. 55–74 (2002)
15. Rieger, S., Noll, T.: Abstracting Complex Data Structures by Hyperedge Replacement. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 69–83. Springer, Heidelberg (2008)
16. Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1. World Scientific Publishing Co., Inc., River Edge (1997)
17. Rozenberg, G., Welzl, E.: Boundary NLC Graph Grammars-Basic Definitions, Normal Forms, and Complexity. Inf. Control 69, 136–167 (1986)
18. Sagiv, M., Reps, T., Wilhelm, R.: Parametric Shape Analysis via 3-Valued Logic. ACM Trans. Program. Lang. Syst. 24, 217–298 (2002)

Unique Small Subgraphs Are Not Easier to Find

Mirosław Kowaluk[1,⋆], Andrzej Lingas[2,⋆⋆], and Eva-Marta Lundell[2]

[1] Institute of Informatics, Warsaw University, Warsaw, Poland
kowaluk@mimuw.edu.pl
[2] Department of Computer Science, Lund University, 22100 Lund, Sweden
{Andrzej.Lingas,Eva-Marta.Lundell}@cs.lth.se

Abstract. Given a pattern graph H of fixed size, and a host graph G guaranteed to contain at most one occurrence of a subgraph isomorphic to H, we show that both the problem of finding such an occurrence (if any) as well as the decision version of the problem are as hard as in the general case when G may contain several occurrences of H.

1 Introduction

The problems of detecting subgraphs or induced subgraphs of a host graph that are isomorphic to a given pattern graph have been widely studied. They are important both in their own rights as well as subproblems for other problems in algorithmics and its applications, for instance in the area of bioinformatics [1,5] or automatic design [21].

The aforementioned problems are generally termed *subgraph isomorphism* and *induced subgraph isomorphism* problems, respectively. Their decision, finding, counting and even enumeration versions have been extensively investigated in the literature. In particular, the decision versions include as special cases such well-known NP-hard problems as the independent set, clique, Hamiltonian cycle or path problems [10]. For arbitrary graphs, they are known to admit polynomial-time solutions only in the case when the pattern graph is of fixed size.

For a given pattern graph H on k vertices and an arbitrary host graph on n vertices, the detection of an induced or non-necessarily induced subgraph in G that is isomorphic to H can be done in time $O(n^{\omega(\lfloor k/3\rfloor,\lceil (k-1)/3\rceil,\lceil k/3\rceil)})$, where $\omega(p,q,r)$ is the exponent of fast arithmetic matrix multiplication of an $n^p \times n^q$ matrix by an $n^q \times n^r$ matrix (cf. [7,11,13,16]).

The subgraph isomorphism and induced subgraph isomorphism for pattern graphs of fixed size are known to admit more efficient algorithmic solutions when restricted to special graph classes, e.g., sparse graphs [6,7] or in particular planar graphs [8].

Already a restriction of the pattern graph of fixed size to a special graph class, e.g., graphs of bounded treewidth, cycles, graphs having a relatively large independent set leads to faster algorithms (cf. [3,4,12,13,15,17,20]).

⋆ Research supported by the grant of the Polish Ministry of Science and Higher Education.

⋆⋆ Research supported in part by Swedish Research Council grant 621-2008-4649.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 336–341, 2011.

In this paper, we address the question of whether or not the guarantee that the host graph contains at most one occurrence of the pattern graph can yield more efficient solutions to the subgraph isomorphism problem with pattern graph of fixed size, than those in the general case where the number of occurrences of the pattern graph is unrestricted.

There are several known examples of combinatorial problems admitting more efficient algorithms under the assumption of solution uniqueness.

For instance, Gabow et al. [9] show that detecting if a given graph on n vertices and m edges has a unique perfect matching, and finding one if it exists, can be done in time $O(m \log^4 n)$. In a weighted setting, a variation of this problem is to decide whether a given perfect maximum-weight matching in a graph is unique. The latter problem has applications in computational biology, namely, in RNA structure prediction.

Next, unique lowest common ancestors in directed acyclic graphs can be found more efficiently than those non-necessarily unique [14]. On the other hand, it is well known that the SAT problem restricted to instances having at most one satisfying assignment is as hard as SAT [19].

We provide a negative answer to the addressed question. Namely, we show that the subgraph isomorphism problem with pattern graph of fixed size efficiently reduces to its restricted case where the host graph is guaranteed to have at most one occurrence of the pattern graph. Our reduction is randomized and it can be regarded as a generalization of the reduction of the problem of finding witnesses of Boolean matrix product to the problem of finding unique witnesses of Boolean matrix product [2,18]. It can be derandomized by using small probability spaces similarly as that in [2].

2 A Reduction to Instances with at Most One Occurrence

The fact that uniqueness of solutions does not help on the level of NP-hard problems is well known [19]. Here we show that analogously uniqueness does not help in finding and reporting small subgraphs.

Throughout the paper, *almost certainly* stands for *with probability at least* $1 - \frac{1}{n^\alpha}$ *for some constant* $\alpha \geq 1$. Furthermore, *an occurrence* of a pattern graph in a host graph denotes a subgraph of the host graph isomorphic to the pattern graph.

In Algorithm 1, we use the technique of gradually diluting the host graph interleaved with testing for unique occurrence of the pattern graph which can be seen as an extension of that for witnesses of Boolean matrix product [2].

Let F^i denote the current graph F after i iterations of the while loop in Algorithm 1. It follows that $F^0 = G$.

Remark 1. The probability that the number of occurrences of H in F^{i+1} is at most $3/4$ of the number of occurrences of H in F^i is at least $1/3$.

To see this, note that the expected number of occurrences of H in F^{i+1} is half the number of occurrences of H in F^i. The statement then follows from Markov inequality.

Note that this holds even if we assume only that every l deletions of edges are independent.

Algorithm 1

Input: A host graph G with n vertices and m edges and a pattern graph H with l edges

Output: An occurrence of H in G if there is any (almost certainly).

Set F to G and c to $16 \log n$;

while F has at least l edges **do**:

1. Delete each edge in F with probability $1-2^{-\frac{1}{l}}$ uniformly at random;
2. **for** $k = 1, ..., c$ **do**:
3. iterate $16k \log n$ times the following four steps:
   - Set K to F;
   - Delete each edge in K with probability $1 - k^{-\frac{1}{l}}$;
   - Run the hypothetical procedure for unique occurrence of H on K;
   - If an occurrence of H in K is found then output it and stop (alternatively in the decision version, if the existence of an occurrence of H in K is reported then report this and stop)

Output "no H"

We now formalize our argument in the following lemma.

Lemma 1. *Algorithm 1 outputs an occurrence of H (or reports it, respectively) if the input graph contains any with probability at least $\frac{1}{2}(1 - e^{-1})$. Let $U(n)$ be the running time of the hypothetical procedure for unique occurrence of H in a graph on n vertices. Algorithm 1 runs in time $\widetilde{O}(l(m + n + U(n)))$ almost certainly (the notation $\widetilde{O}(\ )$ suppresses polylogarithmic in n factors).*

Proof. After $3l \log_2 n$ iterations of the while block, a given edge remains with probability n^{-3}. Hence, the expected number of edges remaining after $3l \log_2 n$ iterations of the block is at most n^{-1} and it is not less than $l \geq 1$ with the probability of at most $n^{-1}l^{-1}$ by Markov's inequality. We conclude that the number of iterations is $O(l \log n)$ almost certainly. Since each iteration takes time $\widetilde{O}(m + n + U(n))$, the upper bound on the running time of Algorithm 1 follows.

Suppose that the input graph contains o occurrences of H. Suppose first that $o \leq c$. Then, during the first iteration of the while loop, for $k = o$, the expected number of occurrences of H in K after the deletion of edges with probability

$1 - k^{-\frac{1}{t}}$ is 1. It follows from Markov's inequality that the number of occurrences of H in K after such edge deletions is greater than 1 with probability at most $\frac{1}{2}$. Note that each set S of the deleted edges from K which leaves at most one occurrence of H contains a maximal subset S' of edges which deletes at most $o-1$ occurrences of H while any extension of the subset within S has not this property. Note also that $S \setminus S'$ is either empty when S leaves one occurrence of H or $S \setminus S'$ contains exactly one occurrence of H otherwise.

It follows that the conditional probability that each of say k non-necessarily consecutive iterations of the deletions of edges from K for $k = o$ which result in at most one occurrence of H would result in 0 occurrences of H is not greater than the probability that a single occurrence of H would disappear after each of the k iterations. In turn, the latter value is not greater than $(1 - \frac{1}{k})^k \leq e^{-1}$.

On the other hand, among the $16k \log n$ iterations of the deletions of edges from K for $k = o$, there is almost certainly a subsequence of k iterations each of which results in at most one occurrence of H. We conclude that at least one of the iterations in the subsequence ends with exactly one occurrence of H in K with probability at least $1 - e^{-1}$. Then, the hypothetical procedure would output it.

Suppose in turn $o > c$. Let j be the maximum i such that F^i contains more than c occurrences of H. Since the expected number of occurrences of H in F^{j+1} is not less than $c/2$, F^{j+1} contains at least one occurrence of H with probability at least $1/2$ (and at most c occurrences of H by the definition of j). It is sufficient now to plug in the argumentation from the first case to conclude that in the $j+1$th iteration of the while loop Algorithm 1 will output an occurrence of H with probability at least $\frac{1}{2}(1 - e^{-1})$. □

Now, we can state our main theorem.

Theorem 1. *Let H be a pattern graph with l vertices. If the problem of finding (respectively, detecting) an occurrence of H in a host graph on n vertices having at most one occurrence of H is solvable in time $U(n)$ then the problem of finding (or, respectively, detecting) an occurrence of H in a host graph on n vertices and m edges is almost certainly solvable in time $\widetilde{O}(l(U(n) + m + n))$.*

Proof. The reduction consists in iterating the method of Lemma 1, i.e., Algorithm 1 a logarithmic number of times. In this way, we can obtain an occurrence of H (or, respectively, report it) if any, almost certainly. □

Corollary 1. *Let H be a pattern graph on $O(1)$ vertices. If the problem of finding (respectively, detecting) an occurrence of H in a host graph on n vertices having at most one occurrence of H is solvable in time $U(n)$ then the problem of finding (or, respectively, detecting) an occurrence of H in a host graph on n vertices and m edges is almost certainly solvable in time $\widetilde{O}(U(n) + m + n)$.*

Alon and Naor derandomized a randomized algorithm for the so called witnesses of Boolean matrix multiplication by using c-wise ϵ-dependent probability spaces of polylogarithmic size for $c = O(\log \log n)$ [2]. The property of such spaces

is that for any subset of random variables of size $i \leq c$, the probability that the random variables attain a given configuration deviates from $\frac{1}{2^i}$ at most by ϵ. The known constructions of these spaces have size polynomial in c, $\frac{1}{\epsilon}$ and $\log n$. In similar fashion, we can derandomize the reduction, i.e., the iterations of Lemma 1, increasing the time performance solely by a polylogarithmic factor for $l = O(1)$.

3 Conclusions

We have shown that for a pattern graph H of a fixed size and an arbitrary host graph G, the problem of detecting or finding a subgraph isomorphic to H in G under the assumption that the number of occurrences of a subgraph isomorphic to H in G is at most one is basically as hard as in the general case.

As one of the referees observed our reduction cannot be directly extended to include induced subgraph isomorphism. Simply, an induced subgraph of a diluted graph is not necessarily an induced subgraph of the original graph.

Acknowledgments

The authors are grateful to referees for valuable comments.

References

1. Alon, N., Dao, P., Hajirasouliha, I., Hormozdiari, F., Sahinalp, S.: Biomolecular network motif counting and discovery by color coding. In: Proceedings 16th International Conference on Intelligent Systems for Molecular Biology (ISMB), Toronto, Canada, July 19-23, pp. 241–249 (2008)
2. Alon, N., Naor, M.: Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions. Algorithmica 16(4), 434–449 (1996)
3. Alon, N., Yuster, R., Zwick, U.: Color-coding. Journal of the ACM 42(4), 844–856 (1995)
4. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica 17(3), 209–223 (1997)
5. Bachman, P., Liu, Y.: Structure discovery in PPI networks using pattern-based network decomposition. Bioinformatics 25(14), 1814–1821 (2009)
6. Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM J. Comput. 14(1), 210–223 (1985)
7. Eisenbrand, F., Grandoni, F.: On the complexity of fixed parameter clique and dominating set. Theoretical Computer Science 326(1-3), 57–67 (2004)
8. Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. J. Graph Algorithms Appl. 3(3), 1–27 (1999)
9. Gabow, H., Kaplan, H., Tarjan, R.: Unique maximum matching algorithms. J. Algorithms 40(2), 159–183 (2001)
10. Garey, M., Johnson, D.: Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences. WH Freeman and Company, New York (2003)

11. Huang, X., Pan, V.: Fast rectangular matrix multiplications and applications. Journal of Complexity 14, 257–299 (1998)
12. Itai, A., Rodeh, M.: Finding a Minimum Circuit in a Graph. SIAM Journal on Computing 7, 413–423 (1978)
13. Kloks, T., Kratsch, D., Muller, H.: Finding and counting small induced subgraphs efficiently. Information Processing Letters 74(3), 115–121 (2000)
14. Kowaluk, M., Lingas, A.: Unique lowest common ancestors in dags are almost as easy as matrix multiplication. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 265–274. Springer, Heidelberg (2007)
15. Kowaluk, M., Lingas, A., Lundell, E.: Counting and detecting small subgraphs via equations and matrix multiplication. In: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1468–1476 (2011)
16. Nešetřil, J., Poljak, S.: On the complexity of the subgraph problem. Commentationes Mathematicae Universitatis Carolinae 26(2), 415–419 (1985)
17. Plehn, J., Voigt, B.: Finding minimally weighted subgraphs. In: Graph-Theoretic Concepts in Computer Science (WG), pp. 18–29 (1991)
18. Seidel, R.: On the all-pairs-shortest-path problem. In: Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing (STOC), pp. 745–749 (1992)
19. Valiant, V., et al.: NP is as easy as detecting unique solutions. Theoretical Computer Science 47, 85–93 (1986)
20. Vassilevska, V., Williams, R.: Finding, minimizing, and counting weighted subgraphs. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC), pp. 455–464 (2009)
21. Wolinski, C., Kuchcinski, K., Raffin, E.: Automatic design of application-specific reconfigurable processor extensions with UPaK synthesis kernel. ACM Transactions on Design Automation of Electronic Systems (TODAES) 15(1), 1–36 (2009)

Simplifying DPDA Using Supplementary Information*

Pavel Labath and Branislav Rovan

Department of Computer Science,
Faculty of Mathematics, Physics and Informatics,
Comenius University, Bratislava, Slovakia
{labath,rovan}@dcs.fmph.uniba.sk

Abstract. We study the effect of using supplementary information on the complexity of deterministic pushdown automata. This continues the study of assisted problem solving initiated in [Gaži, Rovan 2008]. We study deterministic PDA that can assume its input to be in a given regular advisory language. We first show that no suitable complexity measure for DPDA combining the number of states and stack symbols exists. Next we prove tight bounds on the state complexity of an infinite sequence of deterministic context-free languages and show that no supplementary regular information can decrease the state complexity of the DPDA recognizing them. We also find an infinite sequence of deterministic context-free languages for which a suitable supplementary regular information enables the construction of a significantly simpler DPDA.

1 Introduction

In this paper we continue to study the notion of usefulness of information initiated in [1]. We examine the effect of supplementary information on the complexity of recognizing languages. Given an automaton recognizing a certain language, we attempt to construct a simpler automaton recognizing the same language under the assumption that the input is not an arbitrary word from Σ^*, but it belongs to some *advisory* language L_A. We define the language L accepted by the automaton A with an advisory language L_A as follows:

$$L = L(A, L_A) = \{w \mid A \text{ accepts } w \wedge w \in L_A\} \ . \tag{1}$$

If L and L_A are recognized by automata A_L and A_A respectively, then it holds $L = L(A_L) \cap L(A_A)$. This problem is then equivalent to the problem of decomposing an automaton into two new automata such that the result of the computation of the original automaton can be determined from the results of the computations of the new automata. The decomposition is nontrivial if the new automata are simpler than the original automaton.

There are similar approaches studied in the literature (advice functions, promise problems). We insist on having the simpler automaton and the "adviser"

* This research was supported in part by the grant VEGA 1/0726/09.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 342–353, 2011.

to solve together the *original* problem (possibly viewing the adviser to work in parallel) and therefore stressing also the lower complexity of the adviser.

Decompositions of certain types of automata have been studied in the past. Results for the sequential machines can be found in [2][3], decompositions of deterministic finite automata have been studied in [1]. We turn our attention to deterministic pushdown automata (DPDA) with regular supplementary information. In order to study the usefulness of additional information we first need to introduce a measure of complexity for DPDA and show some of its properties.

2 Deterministic Pushdown Automata

We shall use the standard definition of the deterministic pushdown automaton $A = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.[1] We shall use the empty stack accepting mode of DPDA thereby restricting our attention to prefix-free deterministic context-free languages (denoted by $\mathscr{L}_{eDPDA}$). To avoid listing all components of the tuple for an automaton every time, we shall adopt the following convention: if the automaton has an index or some other marker (e.g. A_1) then we use the components of the tuple with the same index or marker (K_1, δ_1, q_{01}, etc.). We shall write the bottom of the stack as the leftmost symbol of a word.

3 Complexity of Deterministic Context-Free Languages

In order to be able to tell if some information simplifies an automaton, we must be able to compare the complexity of automata. Defining the complexity of finite automata is simple: a natural measure of complexity is the number of states of the automaton. In the case of DPDA, we have *at least* two parameters: the number of states and the number of stack symbols. And if we have two automata, one having more states and the other more stack symbols, it is not immediately clear which one should be considered "simpler". Measures of complexity (the number of states, stack symbols, total size of PDA description, etc.) have been considered already (see e.g. [4]). In this section we shall show that a measure of complexity that combines the two parameters and possesses some natural properties does not exist. This justifies the choice of a simpler and more coarse measure — the number of states — used in the rest of the paper.

Let $\preceq$ and $\prec$ denote the standard partial order on pairs of natural numbers (i.e., $(a, b) \preceq (a', b') \overset{\text{def}}{\iff} a \leq a' \wedge b \leq b'$).

Definition 1. *Let A and A' be DPDA. A is simpler than A' if $(|K|, |\Gamma|) \prec (|K'|, |\Gamma'|)$. A is not more complex than A' if $(|K|, |\Gamma|) \preceq (|K'|, |\Gamma'|)$.*

This definition handles the simple case: if the first automaton has less states *and* stack symbols than the second one, then it is natural to say that the first

[1] Formal definition along with all the omitted proofs can be found in the full version of this text at `http://www.st.fmph.uniba.sk/~labath2/master/`

automaton is less complex. This will be our reference definition, to which we shall compare other complexity functions — any reasonable function should preserve the ordering induced by $\prec$.

3.1 Measuring Complexity of Automata

Although we could use $\prec$ to compare the complexity of DPDA, it is preferable to define complexity in a way that induces a total order on the set of automata. A desirable property of such a combined measure is the following: Any two equivalent automata which are minimal in the sense of $\prec$ have the same complexity using the combined measure. We shall now illustrate these ideas using several examples and show that a combined measure of complexity having this property does not exist.

It is easy to see that the following three automata all accept (by empty stack) the language $L = \{a^n b^n \mid n \in \mathbb{N}^+\}$. We have shown that they are minimal in the sense of $\prec$.

$K_1 = \{q_0, q_1\}$	$K_2 = \{q_0, q_1, q_2\}$	$K_3 = \{q_0\}$
$\Gamma_1 = \{Z_0, a\}$	$\Gamma_2 = \{Z_0\}$	$\Gamma_3 = \{Z_0, a, b\}$
$\delta_1(q_0, a, Z_0) = (q_0, a)$	$\delta_2(q_0, a, Z_0) = (q_1, Z_0)$	$\delta_3(q_0, a, Z_0) = (q_0, a)$
$\delta_1(q_0, a, a) = (q_0, aa)$	$\delta_2(q_1, a, Z_0) = (q_1, Z_0 Z_0)$	$\delta_3(q_0, a, a) = (q_0, ba)$
$\delta_1(q_0, b, a) = (q_1, \varepsilon)$	$\delta_2(q_1, b, Z_0) = (q_2, \varepsilon)$	$\delta_3(q_0, b, a) = (q_0, \varepsilon)$
$\delta_1(q_1, b, a) = (q_1, \varepsilon)$	$\delta_2(q_2, b, Z_0) = (q_2, \varepsilon)$	$\delta_3(q_0, b, b) = (q_1, \varepsilon)$

Theorem 1. *The automata A_1, A_2 a A_3 are minimal automata accepting L (with respect to definition 1).*

Proof. A smaller automaton must have parameters $(1, 2)$, $(2, 1)$ or $(1, 1)$. We have shown through a detailed analysis of possible values for the transition function that no such automaton can accept L.

Note 1. Although no smaller automaton can accept L, the same does not hold for a very similar language: $L' = \{a^{n-1} b^n \mid n \in \mathbb{N}^+\}$. L' is accepted by the following automata:

$K_4 = \{q_0\}$	$K_5 = \{q_0, q_1\}$
$\Gamma_4 = \{Z_0, b\}$	$\Gamma_5 = \{Z_0\}$
$\delta_4(q_0, a, Z_0) = (q_0, bZ_0)$	$\delta_5(q_0, a, Z_0) = (q_0, Z_0 Z_0)$
$\delta_4(q_0, b, Z_0) = (q_0, \varepsilon)$	$\delta_5(q_0, b, Z_0) = (q_1, \varepsilon)$
$\delta_4(q_0, b, b) = (q_0, \varepsilon)$	$\delta_5(q_1, b, Z_0) = (q_1, \varepsilon)$

This example shows how delicate is the complexity of languages — even a small change in the language can enable us to recognize it with a simpler automaton.

Since A_1, A_2 and A_3 are minimal automata, it would be desirable to assign them the same complexity. Looking at these automata, one might conclude that the function "sum of the number of states and stack symbols" is a good complexity measure. However, it is easy to find an example where the product of the two numbers would fit better. We shall now show that a combined measure assigning the same complexity to A_1, A_2 and A_3 does not exist.

Theorem 2. *There is no function* $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ *such that for each pair of automata* A *and* A' *accepting the same language* L *it holds:*

1. $(|K|, |\Gamma|) \prec (|K'|, |\Gamma'|) \Longrightarrow f(|K|, |\Gamma|) < f(|K'|, |\Gamma'|)$[2]
2. *If* A *and* A' *are minimal automata for* L *then* $f(|K|, |\Gamma|) = f(|K'|, |\Gamma'|)$

Proof. By contradiction. Let us assume such a function f does exist. Let us examine the following (regular) language

$$L_{\mathrm{mod2}} = \{wc \mid w \in \{a, b\}^* \wedge \#_a(w) \equiv \#_b(w) \equiv 0 \pmod 2\} \ . \tag{2}$$

Consider a minimal automaton for L_{mod2}. If we fix the number of stack symbols to 1 then it must have at least 4 states corresponding to the 4 combinations of remainders. Clearly, 4 states suffice because one can easily construct a 4-state automaton accepting L_{mod2}. If we allow the automaton to use 2 states then 2 stack symbols are sufficient. The latter automaton can count the number of symbols a (modulo 2) in the state and the number of symbols b on the stack, while the former can do it all in the state. When the automata read c they empty the stack if the remainders are 0.

Hence, it must hold $f(4, 1) = f(2, 2)$. From our study of the language L we know that A_1 and A_2 are its minimal automata and therefore $f(2, 2) = f(3, 1)$. But this contradicts the first requirement which states that $f(4, 1) > f(3, 1)$. □

3.2 State Complexity

In view of Theorem 2 we shall use a simpler (and coarser) measure of complexity — the number of states. Note that this is not a trivial measure, since there is no "one state normal form" like we have for nondeterministic PDA.

Definition 2. *The state complexity of DPDA* A *is the number of its states and we shall denote it by* $\mathrm{STATE}(A)$. *A is simpler (resp. not more complex) than DPDA* A' *if it holds* $|K| < |K'|$ *(resp.* $|K| \leq |K'|$*). The complexity of* $L \in \mathscr{L}_{\mathrm{eDPDA}}$ *is the complexity of the simplest automaton accepting* L *and is denoted by* $\mathrm{STATE}(L)$.

Note 2. State complexity induces a total order on the set of automata and it is compatible with $\prec$. Furthermore, due to well-orderedness of natural numbers, the state complexity of languages is well defined.

In Sect. 4, we examine the effect of supplementary information on the complexity of automata and languages. To see if some information simplifies a language we need a tight bound on the complexity of some languages. Here we focus our attention on the following sequence of languages:

$$L_1 = L = \{a^n b^n \mid n \in \mathbb{N}^+\} \tag{3}$$

$$L_{i+1} = \{a^n w b^n \mid n \in \mathbb{N}^+ \wedge w \in {L_i}^*\} \ . \tag{4}$$

[2] We require the strict inequality to disallow trivial functions assigning the same value to all inputs.

The words in L_i contain correctly parenthesized symbols a and b. The difference between the languages is in the level of nesting. For example, the word $aaaaaaabbabbbaabbbbb$ belongs to $L_3 - L_2$ because

$$aaaaaaabbabbbaabbbbb = a^3 \underbrace{a^2 \overbrace{a^2b^2}^{\in L_1} \overbrace{ab}^{\in L_1} b^2}_{\in L_2 - L_1} \underbrace{a^2b^2}_{\in L_2} b^3 \in L_3 - L_2 \ . \tag{5}$$

These words can also be represented graphically. Two such examples can be found in Fig. 1 (page 352).

Our intention is to prove that the complexity of language L_n is exactly n. To prove the upper bound, we shall construct an automaton having n states and $n+2$ stack symbols. After that, we shall prove that there is no automaton accepting L_n with a smaller number of states.

Theorem 3. $\mathrm{STATE}(L_n) = n$ *for each* $n \in \mathbb{N}^+$.

Proof. We first show that $\mathrm{STATE}(L_n) \leq n$. We construct an automaton $A_n = (\{q_1, \ldots, q_n\}, \{a, b\}, \{Z_0, Z_1, \ldots, Z_{n-1}, a, b\}, \delta_n, q_1, Z_0, \emptyset)$ for each n:
$\delta_n(q_1, a, Z_0) = (q_1, a)$; $\delta_n(q_1, a, a) = (q_1, ba)$; $\delta_n(q_1, b, a) = (q_1, \varepsilon)$;
$\delta_n(q_i, b, b) = (q_i, \varepsilon)$ for $i = 1 \ldots n$;
$\delta_n(q_i, b, Z_j) = (q_{max(i,j)+1}, \varepsilon)$, $\delta_n(q_i, a, Z_j) = (q_1, Z_{max(i,j)}a)$ for $i, j = 1 \ldots n-1$.
$\delta_n(q_i, a, b) = (q_1, Z_i a)$ for $i = 1 \ldots n-1$;
The automaton operates similarly to A_2 from Subsection 3.1 — always preserving the invariant "number of open parentheses" = "number of symbols on the stack". Additionally, it must check the nesting level. It does that in two places. It stores the symbols Z_i on the stack to indicate the place where a nest ends. The state of the automaton indicates the nesting level of the currently processed nest.

We now show (by contradiction) that $\mathrm{STATE}(L_n) \geq n$. Let us assume that an automaton A such that $|K_A| = n-1$ does exist. Let $z = |\Gamma_A|$. The computation of A on the word $c_1c_2 \cdots c_l$ ($c_i \in \{a, \varepsilon\}$) is

$$(p_0, c_1c_2 \cdots c_l, z_0) \vdash (p_1, c_2 \cdots c_l, s_1z_1) \vdash \cdots \vdash (p_l, \varepsilon, s_lz_l) \ . \tag{6}$$

For each $i \in \mathbb{N}^+$, a^i is a prefix of a word in L_n. Hence, A must not halt while reading a^i. According to the pigeonhole principle, for $l > z \cdot n$ some pair (state, stack symbol) must occur twice. After that, the automaton will operate in "cycles". The pairs must not repeat themselves because then the automaton could not know how many symbols it has read. Therefore, the stack will gradually grow, but it will contain a repeating block. Formally, after reading a^i for a sufficiently large i, the pair will be $(s_p\gamma^k s_s^{(i)}, q^{(i)})$ (with s_p fixed, k gradually increasing and $s_s^{(i)}$ and $q^{(i)}$ periodically repeating).

Let us fix $s_s = s_s^{(i)}$ for some i. For $k = 1, 2, \ldots$ let n_k be a number such that after reading a^{n_k} the stack of A will be $s_p\gamma^k s_s$. Let us define a sequence of words w_i by $w_1 = ab$ and $w_{i+1} = aw_ibab$. We shall use these words to control the nesting level of the words containing them. E.g., $aw_2b = a\ aw_1b\ ab\ b = a\ aabb\ ab\ b$ is in L_2 because it contains two words from L_1 ($aabb$ and ab) surrounded by a and b.

Let us consider the behavior of the automaton while reading $u_i = a^{n_k} w_i b^{n_k} \in L_n$ for $i = 1 \dots n$. The stack cannot shrink significantly while A is reading the middle part since it would "forget" the number of symbols read. A must empty the stack after having read b^{n_k}. Therefore, at some point it will reach the configuration $(q_{x_i}, b^{l_i}, s_p \gamma^{k-1})$ for some l_i, x_i. A has only $n-1$ states $\rightsquigarrow$ $q_{x_i} = q_{x_j}$ for some $i, j \in \{1, 2, \dots, n\}; i < j$. Let

$$v_i = a^{n_k} w_i b^{n_k - l_i} (abb)^{n-i} b^{l_i - (n-i)} \in L_n \tag{7}$$

$$v_j = a^{n_k} w_j b^{n_k - l_j} (abb)^{n-j} b^{l_j - (n-j)} \in L_n \ . \tag{8}$$

These words have the same prefixes as u_i and u_j. While reading them, A will reach configurations $(q_{x_i}, (abb)^{n-i} b^{l_i-(n-i)}, s_p \gamma^{k-1})$ and $(q_{x_j}, (abb)^{n-j} b^{l_j-(n-j)}, s_p \gamma^{k-1})$. But $q_{x_i} = q_{x_j}$ $\rightsquigarrow$ the stack-state pairs are the same and A cannot tell these configurations apart. We can swap the suffixes and the corresponding parts of the computation and we shall obtain an accepting computation on the word $a^{n_k} w_j b^{n_k - l_j} (abb)^{n-i} b^{l_i-(n-i)}$, which does not belong to L_n, because it contains more than n nests $\rightsquigarrow$ contradiction with $N(A) = L_n$. □

The automaton A_n needed n states because it had to store the information about the nesting level even when the stack was shrinking. It can be shown that this is the only case when an automaton must store information in its state. States that are not reachable from stack-shrinking rules can be replaced by stack symbols. Given an automaton A, we can construct an equivalent automaton A' which will have K_ε as the set of states, where

$$K_\varepsilon = \{p \in K \mid \exists q \in K \ \ \exists x \in \Sigma \cup \{\varepsilon\} \ \ \exists Z \in \Gamma \quad \delta(q, x, Z) = (p, \varepsilon)\} \ . \tag{9}$$

The idea is that the new automaton can store the state on its stack (in the top symbol, which will be a pair consisting of the state and the original stack symbol). This approach works when the stack does not shrink. When the stack does shrink, we must store the information in the state, and this means the automaton needs $|K_\varepsilon|$ states. It is important to realize that this optimization does not have to produce a minimal automaton for the given language. A different automaton, operating in a completely different way, may exist and it may have a smaller number of states.

Introducing the state complexity point of view to DPDA can lead to improvements in some "standard" constructions. For example, the exponential increase of the number of states for the complement construction is not necessary and it can be shown that a linear increase suffices. In fact, the construction of the "automaton reads the whole input" normal form (which is the key part of the complement construction) can be done by adding only one state.

4 Usefulness of Supplementary Information

In this section we study how (and if) can regular supplementary information help a deterministic pushdown automaton. In the first part we show examples

when the information enables us to construct a substantially simpler automaton. We conclude the section by showing that no regular information can help the minimal automata for L_n (i.e., the languages are not decomposable, they cannot be written as the intersection of a regular language and a deterministic context-free language accepted by a simpler automaton).

4.1 Decomposable Languages

We can study the effect of supplementary information from two points of view. The first possibility is to take a specific automaton A and then try to find an advisory language which would enable us to construct a simpler automaton A'. The new automaton must recognize the same language but it will have help in the form of a regular advisory language.

Let us study the following sequence of automata $A_1, A_2, \ldots$

$$\begin{aligned}
A_n &= (\{q_0, \ldots, q_{n-1}\}, \{a, b\}, \{Z_0, Z_1, a\}, \delta_n, q_0, Z_0, \emptyset) \\
\delta_n(q_0, a, Z_0) &= (q_{(1 \bmod n)}, Z_1) \\
\delta_n(q_i, a, x) &= (q_{(i+1) \bmod n}, xa) \qquad i \in \{0, \ldots, n-1\} \quad x \in \{Z_1, a\} \\
\delta_n(q_i, b, a) &= (q_i, \varepsilon) \qquad\qquad\quad i \in \{0, \ldots, n-1\} \\
\delta_n(q_0, b, Z_1) &= (q_0, \varepsilon)
\end{aligned}$$

The automaton A_n recognizes the language

$$D_{(\bmod n)} = \{awb \mid w \in D_1 \wedge \#_a(awb) \equiv 0 \pmod n\} \ , \tag{10}$$

where D_1 is the *Dyck* language consisting of balanced pairs of parentheses (symbols a and b in our case). The automaton has complexity n. However, it is easy to see that if we choose

$$L_{(\bmod n)} = \{w \in \{a, b\}^+ \mid \#_a(w) \equiv 0 \pmod n\} \tag{11}$$

as the advisory language we can accept $D_{(\bmod n)}$ with only one state: when we know the input has the correct number of symbols a it is sufficient to check the balancing. The automaton $A' = A_1$ can do that with one state, therefore it holds

$$D_{(\bmod n)} = L(A_1, L_{(\bmod n)}) = L_{A_1} \cap L_{(\bmod n)} \tag{12}$$

The problem with this approach is that, in general, we are not able to tell whether the simplification of the automaton is a consequence of the help provided by the advisory language. It is possible that there is an automaton A_m recognizing the same language which is simpler than our automaton A_n. The automaton can even be simpler than A'.

Therefore, it is better to study the effect of supplementary information from the language point of view. We can identify a language with its minimal automaton. If we now show that the new automaton is simpler than a *minimal* automaton it is clear this was caused by the presence of the advisory language and that the information contained within is *in this particular case*[3] useful. Then

[3] The same information may be totally useless for some other language/automaton.

the most complicated part is to prove that an automaton is a minimal automaton for the given language. In the case of $L_{(\mathrm{mod}\ n)}$, A_n is indeed a minimal automaton. The proof is similar to that of Theorem 3 and is omitted due to lack of space.

Theorem 4. *$STATE(D_{(\mathrm{mod}\ n)}) = n$ for each $n \in \mathbb{N}^+$ (i.e., the automaton A_n is a minimal automaton for the language $D_{(\mathrm{mod}\ n)}$).*

This shows that the information whether a word belongs to $L_{(\mathrm{mod}\ n)}$ *is* relevant when testing whether a word belongs to $D_{(\mathrm{mod}\ n)}$ and allows us to reduce the number of states from n to 1. Hence, we can say that the decomposition of the language $D_{(\mathrm{mod}\ n)}$ to languages $L_{(\mathrm{mod}\ n)}$ and D_1 is nontrivial, according to the following definition.

Definition 3. *Let $L, L_1 \in \mathscr{L}_{eDPDA}$. Let L_2 be a regular language and let $L = L_1 \cap L_2$. Let A and A_1 be automata recognizing languages L and L_1 respectively and let A be a minimal automaton recognizing L (using the "number of states" criterion). The language L is **nontrivially** decomposable to languages L_1 and L_2 if the automaton A_1 has (strictly) less states than A.*

This definition does not limit the complexity of the finite automaton recognizing L_2. That means the automaton may possibly have significantly more states than the original automaton A. Despite that, we shall consider this to be a nontrivial decomposition since the finite automaton (even when it has more states) is a "simpler" kind of a device than the deterministic pushdown automaton. We shall show examples of languages which are not decomposable even when we allow the finite automaton to have an arbitrary number of states.

4.2 Non-decomposable Languages

In Sect. 3 we have defined the sequence of languages $\{L_n\}_{n=1}^{\infty}$ and proved that the complexity of the nth language is exactly n. We would like to know if some regular information would enable us to simplify the automaton recognizing L_n. At first, one might think it is possible — we have the limitation on the nesting level, which is a constant (for a fixed n) and one might want to construct a finite automaton checking that. This would leave the DPDA with the task of checking the balancing of the symbols, which we know can be done using just one state.

However, this is not true. Namely, the nesting level depends too much on the exact number of symbols in a word. For example, the word

$$a^{3m}b^m a^m b^m a^m b^{3m} = a^m a^m \overbrace{a^m b^m}^{\in L_1} \overbrace{a^m b^m}^{\in L_1} \overbrace{a^m b^m}^{\in L_1} b^m b^m \tag{13}$$

belongs in L_2, but the word

$$a^{3m}b^m a^m b^{2m} a^m b^{2m} = a^m \underbrace{a^m \overbrace{a^m b^m}^{\in L_1} \overbrace{a^m b^m}^{\in L_1} b^m}_{\in L_2 - L_1} \underbrace{a^m b^m}_{\in L_2} b^m \tag{14}$$

is in $L_3 - L_2$. In this subsection we shall prove that the languages L_n cannot be decomposed. The formal proof consists of two parts.

1. We shall find an infinite set of words (let us call it L_{bad}) which must be accepted by an automaton accepting a superset of L_n with less than n states. This part is similar to the proof of Theorem 3, where we find a single such word. However, here we must analyze the behavior of the automaton in more detail, because we need an infinite (and non-regular) set, so we can use it in the second step.
2. We shall show that there is no DFA that accepts all words from L_n while rejecting every word from L_{bad}. We shall use the non-regularity of L_{bad} to show that the finite automaton cannot distinguish "good" words from L_n and "bad" words from L_{bad}.

To improve readability, we have split the first step into several lemmas. The first one handles the simplest case — the language L_2.[4] Due to the lack of space, we shall omit the proof for the more complex languages. Instead, we shall only briefly describe the differences.

Lemma 1. *Let A be a single-state DPDA such that $L_2 \subseteq N(A)$. There exists $m \in \mathbb{N}^+$ such that for each $j, l \in \mathbb{N}^+$ there exist $i, k, t \in \mathbb{N}_0 : k > 2j$ such that A accepts*

$$a^i b^t\ a^l b^l\ b^{jm}\ ab\ b^{(k-j)m} b^{i-t-km}\ . \tag{15}$$

Before we can prove this theorem we need to introduce some definitions.

Definition 4. *Let A be a DPDA with a single state q_0. Partial functions $c_A : \Gamma^* \to \mathbb{N}_0$, $B_A : \Gamma^* \to \{b\}^*$ are defined as follows:*
Let $s \in \Gamma^*$. Let there exist $m \in \mathbb{N}_0$ such that[5]

$$(q_0, b^m, s) \vdash_A^* (q_0, \varepsilon, \varepsilon)\ . \tag{16}$$

Then $c_A(s) = m$, $B_A(s) = b^m$. If there is no such m the functions are undefined. If it is clear which automaton we have in mind, we omit the index A.

We shall use these functions in the piecewise construction of the input for the automaton. We shall give the automaton a part of the input, look at the contents of the stack after the automaton has read it and based on that information we shall give the automaton the next part of the input word. If we know that the input can be completed to a word from the accepted language with a number of symbols b then the function c_A returns that number. E.g., if the word w' has that property and the stack after reading it is $s_1 s_2$ then the stack after reading $w' b^{c(s_2)} = w' B(s_2)$ is s_1. Additionally, the functions have the following properties:

$$c(s_2 s_1) = c(s_1) + c(s_2)\ , \tag{17}$$

$$B(s_2 s_1) = B(s_1) \cdot B(s_2)\ . \tag{18}$$

Now we can proceed with the proof.

[4] One need not consider L_1 since its minimal automaton already has only one state.
[5] If such an m exists, it is unique, because A is deterministic.

Proof (of Lemma 7). The stack of A after reading sufficiently long a^i will be $s_p\gamma^k s_s^{(i)}$. As in the proof of Theorem 3 we shall choose a fixed s_s and vary i.

After reading $u = a^iB(\gamma s_s)$ the stack will be $s_p\gamma^{k-1}$. What will the stack look like after reading $u\,a^lb^l$, $l \in \mathbb{N}^+$? While processing a^lb^l the stack will never drop below $s_p\gamma^{k-2}$. We shall show this by contradiction. Let the stack after reading uu_0 be $s_p\gamma^{k-2}$, where u_0 is some prefix of a^lb^l. Then it holds $\#_a(u_0) \geq \#_b(u_0)$ and A will accept the word $uu_0B(s_p\gamma^{k-2})$, but not $uu_0b^{\#_a(uu_0)-\#_b(uu_0)} \in L_2$, because

$$\#_a(uu_0) - \#_b(uu_0) \geq i - \#_b(u) = c(s_p\gamma^k s_s) - c(\gamma s_s) > c(s_p\gamma^{k-2}) \ . \tag{19}$$

Therefore, the stack after reading $u\,a^lb^l$ can be written as $s_p\gamma^{k-2}\gamma'$. The exact value of γ' may depend on l, but for each l it must hold $c(\gamma') = c(\gamma)$ because $u\,a^lb^l$ has the same number of "open brackets"[6] as u.

For $j = 2 \ldots k-1$ after reading

$$v_j = u\ a^lb^l\ B(\gamma^{j-2}\gamma') = a^iB(\gamma s_s)\ a^lb^l\ B(\gamma^{j-2}\gamma') = a^iB(\gamma s_s)\ a^lb^l\ B(\gamma^j - 1) \tag{20}$$

the stack will be $s_p\gamma^{k-j}$. However, the automaton will have the same stack after reading $w_j = a^iB(\gamma^j s_s)$ as well. v_ja is not a prefix of a word from L_2, but w_ja is. Since A cannot tell the words apart, we can say that the stack after reading v_ja and w_ja will be $s_p\gamma^{k-1-j}\gamma''$, where

$$c(\gamma'') = c(\gamma) + 1 \ . \tag{21}$$

Since A accepts $w_jaB(s_p\gamma^{k-1-j}\gamma'') \in L_2$, it must also accept

$$v_jaB(s_p\gamma^{k-1-j}\gamma'') = a^iB(\gamma s_s)\ a^lb^l\ B(\gamma^j)aB(s_p\gamma^{k-1-j}\gamma'') \ . \tag{22}$$

Using (21) and (18) we can rewrite the word as

$$a^iB(\gamma s_s)\ a^lb^l\ B(\gamma)^j\ ab\ B(\gamma)^{k-j}B(s_p) \ . \tag{23}$$

We can chose i arbitrarily large and so for each $j \in \mathbb{N}^+$ we can find an i such that $k = k(i) > 2j$. When we replace $c(\gamma s_s)$ with t and $c(\gamma)$ with m we get

$$a^ib^t\ a^lb^l\ b^{jm}\ ab\ b^{(k-j)m}B(s_p) \ . \tag{24}$$

From the definition of c it follows that

$$i = c(s_p\gamma^k s_s) = c(s_p) + kc(\gamma) + c(s_s) = c(s_p) + km + t \ . \tag{25}$$

Hence, it holds $c(s_p) = i - t - km$ and so the word meets all the requirements of the lemma. □

The next step is to prove a similar lemma for the language L_3. A minimal automaton for this language has 3 states, thus a simpler automaton may have

[6] $\#_a(u\,a^lb^l) - \#_b(u\,a^lb^l) = \#_a(u) - \#_b(u)$.

more than one state. This means that we cannot use some of the arguments from the previous proof. The biggest difference is that we have to take the state of the automaton into account when constructing the words using the c_A and B_A functions and we look for words that produce the same stack, we need to make sure the automaton is in the same state as well.

After that, we can prove the lemma for the general case. The difference from the previous one is that we must use induction for the piecewise construction of words from $L_{n+1} - L_n$ which have the desired form.

Lemma 2. *Let $n \in \mathbb{N}^+$. Let A be a DPDA such that $|K_A| = n - 1$ and $L_n \subseteq N(A)$. Then there exists $m \in \mathbb{N}^+$ such that for all sufficiently large $j, l \in \mathbb{N}^+$ there exist $i, k, t, x_1, \ldots x_{n-1} \in \mathbb{N}_0 : k > 2j$ such that A accepts*

$$a^i b^t \; a^l b^l \; b^{jm} \; abb^{x_1} \; \cdots \; abb^{x_{n-1}} \; b^{(k-j)m} b^{i-t-km-\sum_{s=1}^{n-1} x_s} \tag{26}$$

This lemma finds words which are accepted by a simpler automaton accepting all the words from L_n. We then prove that there is no finite automaton that can reject all these bad words while accepting all the good words from L_n. From this it follows that the intersection of the languages accepted by these two automata cannot be L_n and that L_n (and its minimal automaton) is non-decomposable.

Theorem 5. *The languages L_n for $n \in \mathbb{N}^+$ are not (non-trivially) decomposable.*

Proof. By contradiction. Let A be a DPDA and A' an FSA such that $N(A) \cap L(A') = L_n$. Then it holds $L_n \subseteq N(A)$ and according to the previous lemmas there exists $m \in \mathbb{N}^+$ such that for each (sufficiently large) $j, l \in \mathbb{N}^+$ there exist $i, k, t, x_1, \ldots x_{n-1} \in \mathbb{N}_0 : k > 2j$ such that A accepts

$$w_{j,l} = a^i b^t \; a^l b^l \; b^{jm} \; abb^{x_1} \; \cdots \; abb^{x_{n-1}} \; b^{(k-j)m} b^{i-t-km-\sum_{s=1}^{n-1} x_s} \; . \tag{27}$$

$w_{j,l}$ (the word for $n = 3$ is depicted on Fig. 1 left) is accepted by A but it does not belong in L_n and so A' must not accept $w_{j,l}$ for any j and l.

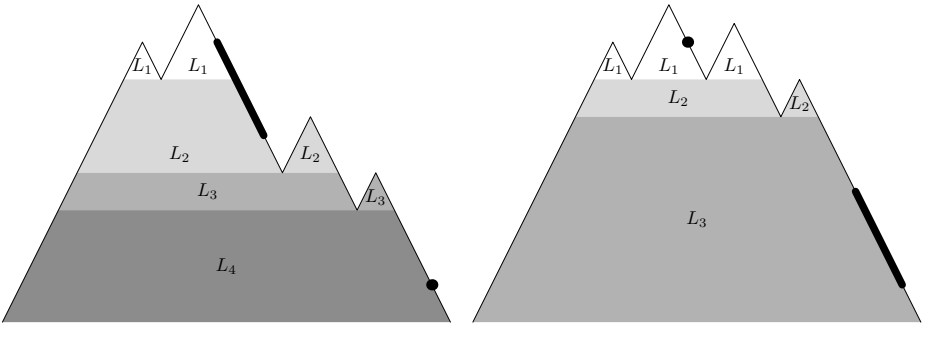


Fig. 1. Word $w_{j,t}$ before and after "pumping"

Let $l = j = |K_{A'}|!$. How will A' behave when processing $w_{j,l}$? The length of the part $b^l b^{jm}$ is $|K_{A'}|! \cdot (m+1)$. That is more than the number of states of the automaton and so the automaton will cycle. The length of the cycle evenly divides $|K_{A'}|! \cdot m$ so we can surely remove that number of symbols b and the automaton will not "notice" that — it will reject the word

$$a^i b^t \; a^{|K_{A'}|!} b^{|K_{A'}|!} \; abb^{x_1} \; \cdots \; abb^{x_{n-1}} \; b^{k-|K_{A'}|!} b^{i-t-km-\sum_{s=1}^{n-1} x_s} \tag{28}$$

as well. Now let us focus on the part $b^{(k-|K_{A'}|!)m}$. Its length is at least $|K_{A'}|!$ (since $k > 2j$) and so we can add $|K_{A'}|! \cdot m$ symbols b. That means the automaton will also reject the word

$$\begin{aligned} w' &= a^i b^t \; a^{|K_{A'}|!} b^{|K_{A'}|!} \; abb^{x_1} \cdots abb^{x_{n-1}} \; b^{km} b^{i-t-km-\sum_{s=1}^{n-1} x_s} \\ &= \underbrace{a^{i-t-\sum_{s=1}^{n-1} x_s} \; a^{x_{n-1}} \cdots \underbrace{a^{x_1} \overbrace{a^t b^t}^{\in L_1} \overbrace{a^{|K_{A'}|!} b^{|K_{A'}|!}}^{\in L_1} abb^{x_1}}_{\in L_2} \cdots \overbrace{ab}^{\in L_1} b^{x_{n-1}} b^{i-t-\sum_{s=1}^{n-1} x_s}}_{\in L_n} \end{aligned} \tag{29}$$

w' (Fig. 1 right) is not in $L(A')$, hence it cannot be in the intersection of $L(A')$ and $N(A)$. That contradicts the assumption $N(A) \cap L(A') = L_n$. □

5 Conclusion

There are several ways to extend this research. One can study the (closure) properties of the class of decomposable and non-decomposable languages. Also, one could study other definitions of the complexity of pushdown automata: non-decomposable languages according to the "number of states" measure could become decomposable if one used a finer measure. Another option is to consider modifications of the DPDA (e.g., accepting by state, a stack with a fixed bottom-of-stack symbol) or a more powerful model of computation.

References

1. Gaži, P., Rovan, B.: Assisted problem solving and decompositions of finite automata. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 292–303. Springer, Heidelberg (2008)
2. Gécseg, F.: Products of Automata. Springer, Berlin (1986)
3. Gécseg, F., Peák, I.: Algebraic Theory of Automata. Akadémiai Kiadó, Budapest (1972)
4. Valiant, L.G.: Regularity and related problems for deterministic pushdown automata. J. ACM 22(1), 1–10 (1975)

Normalization of Sequential Top-Down Tree-to-Word Transducers

Grégoire Laurence[1,2], Aurélien Lemay[1,2], Joachim Niehren[1,3], Sławek Staworko[1,2], and Marc Tommasi[1,2]

[1] Mostrare project, INRIA & LIFL (CNRS UMR8022)
[2] University of Lille, France
[3] INRIA, Lille, France

Abstract. We study normalization of deterministic sequential top-down tree-to-word transducers (STWs), that capture the class of deterministic top-down nested-word to word transducers. We identify the subclass of *earliest* STWs (eSTWs) that yield unique normal forms when minimized. The main result of this paper is an effective normalization procedure for STWs. It consists of two stages: we first convert a given STW to an equivalent eSTW, and then, we minimize the eSTW.

1 Introduction

The classical problems on transducers are equivalence, minimization, learning, type checking, and functionality [2,13,14,6]. Except for the latter two questions, one usually studies deterministic transducers because non-determinism quickly leads to fundamental limitations. For instance, equivalence of non-deterministic string transducers is known to be undecidable [8]. We thus follow the tradition to study classes of deterministic transducers. The problems of equivalence, minimization, and learning are often solved using unique normal representation of transformations definable with a transducer from a given class [9,7,5,11]. Normalization i.e., constructing the normal form of a given transducer, has been studied independently for various classes, including string transducers [4,3], top-down tree transducers [5], and bottom-up tree transducers [7].

In this paper, we study the normalization problem for the class of deterministic sequential top-down tree-to-word transducers (STWs). STWs are finite state machines that traverse the input tree in top-down fashion and at every node produce words obtained by the concatenation of constant words and the results from processing the child nodes. The main motivation to study this model is because tree-to-word transformations are better suited to model general XML transformations as opposed to tree-to-tree transducers [5,11,14]. This follows from the observation that general purpose XML transformation languages, like XSLT, allow to define transformations from XML documents to arbitrary, not necessarily structured, formats. Also, STWs capture a large subclass of deterministic nested-word to word transducers (DN2W), which have recently been the object of an enlivened interest [6,15,16].

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 354–365, 2011.

Expressiveness of STWs suffers from two limitations: 1) every node is visited exactly once, and 2) the nodes are visited in the fix left-to-right preorder traversal of the input tree. Consequently, STWs cannot express transformations that reorder the nodes of the input tree or make multiple copies of a part of the input document. STWs remain, however, very powerful and are capable of: concatenation in the output, producing arbitrary context-free languages, deleting inner nodes, and verifying that the input tree belongs to the domain even when deleting parts of it. These features are often missing in tree-to-tree transducers, and for instance, make STWs incomparable with the class of top-down tree-to-tree transducers [5,11].

Normal forms of transducers are typically obtained in two steps: output normalization followed by machine minimization. A natural way of output normalization is (re)arranging output words among the transitions rules so that the output is produced as soon as possible when reading the input, and thus transducers producing output in this fashion are called *earliest*. Our method subscribes to this approach but we note that it is a challenging direction that is not always feasible in the context of tree transformations. For instance, it fails for bottom-up tree-to-tree transducers, where *ad-hoc* solutions need to be employed [7].

We propose a natural normal form for STWs because on being earliest for STWs and define the corresponding class of *earliest* STWs (eSTWs) using easy to verify structural requirements. We present an effective procedure to convert an STW to an equivalent eSTW. This process is very challenging and requires novel tools on word languages. We point out that while this procedure works in time polynomial in the size of the output eSTW, we only know a doubly-exponential upper-bound and a single-exponential lower bound of the size of the output eSTW. This high complexity springs from the fact that the output language of an STW may be an arbitrary context-free language. We also show that minimization of earliest STWs is in PTIME thanks to a fundamental property: two equivalent eSTWs have rules of the same form and allow bisimulation. General STWs are unlikely to enjoy a similar property because their minimization is NP-complete.

Overall, we obtain an effective normalization procedure for STWs. Our results also offer an important step towards a better understanding of the same problem for dN2Ws because STWs capture a large class of top-down dN2Ws modulo the standard first-child next-sibling encoding and the conversion from one model to another can be done efficiently [16]. It is a significant result because there exist arguments suggesting that arbitrary dN2Ws are unlikely to have natural normal forms [1].

Organization. In Section 2 we present basic notions and introduce STWs and eSTWs. Section 3 introduces important tools on word languages and presents an STW to eSTW conversion algorithm. In Section 4 we deal with minimization of STWs and eSTWs. Section 5 summarizes our work and outlines future directions. Because of space restrictions we omit the proofs, which can be found in the full version at `http://hal.inria.fr/inria-00566291/en/`.

2 Sequential Top-Down Tree-to-Word Transducers

A *ranked alphabet* is a finite set of ranked symbols $\Sigma = \bigcup_{k\geq 0} \Sigma^{(k)}$ where $\Sigma^{(k)}$ is the set of k-ary symbols. We assume that every symbol has a unique arity i.e., $\Sigma^{(i)} \cap \Sigma^{(j)} = \varnothing$ for $i \neq j$. We use $f, g, \ldots$ to range over symbols of non negative arity and $a, b, \ldots$ to range over *constants* i.e., symbols of arity 0. We write $f^{(k)}$ to indicate that $f \in \Sigma^{(k)}$ if the arity of f is not known from the context. A *tree* is a ranked ordered term over Σ. We use $t, t_0, t_1, \ldots$ to range over trees. For instance, $t_0 = f(a, g(b))$ is a tree over $\Sigma = \{f^{(2)}, g^{(1)}, a^{(0)}, b^{(0)}\}$.

For a finite set Δ of symbols by Δ^* we denote the free monoid on Δ. We write $u \cdot v$ for the concatenation of two words u and v and ε for the empty word. We use $a, b, \ldots$ to range over Δ and $u, v, w, \ldots$ to range over Δ^*. For a word w by $|w|$ we denote its length. Given a word $u = u_p \cdot u_f \cdot u_s$, u_p is a *prefix* of u, u_f a *factor* of u, and u_s a *suffix* of u. The *longest common prefix* of a nonempty set of words W, denoted $lcp(W)$, is the longest word u that is a prefix of every word in W. Analogously, we define the *longest common suffix* $lcs(W)$.

Definition 1. *A* deterministic sequential top-down tree-to-word transducer (STW) *is a tuple* $M = (\Sigma, \Delta, Q, init, \delta)$*, where* Σ *is a ranked alphabet of* input trees*,* Δ *is a finite alphabet of* output words*,* Q *is a finite set of states,* $init \in \Delta^* \cdot Q \cdot \Delta^*$ *is the* initial rule*,* δ *is a partial transition function from* $Q \times \Sigma$ *to* $(\Delta \cup Q)^*$ *such that if* $\delta(q, f^{(k)})$ *is defined, then it has exactly* k *occurrences of elements from* Q*. By* STWs *we denote the class of deterministic sequential top-down tree-to-word transducers.*

In the sequel, if $u_0 \cdot q_0 \cdot u_1$ is the initial rule, then we call q_0 the initial state. Also, we often view δ as a set of *transition rules* i.e., a subset of $Q \times \Sigma \times (\Delta \cup Q)^*$, which allows us to quantify over δ. The *size* of the STW M is the number of its states and the lengths of its rules, including the lengths of words used in the rules. The semantics of the STW M is defined with the help of auxiliary partial functions T_q (for $q \in Q$), recursively defined on the structure of trees as follows:

$$T_q(f(t_1, \ldots, t_k)) = \begin{cases} u_0 \cdot T_{q_1}(t_1) \cdot u_1 \cdot \ldots \cdot T_{q_k}(t_k) \cdot u_k, \\ \qquad\qquad \text{if } \delta(q, f) = u_0 \cdot q_1 \cdot u_1 \ldots \cdot q_k \cdot u_k, \\ \text{undefined}, \qquad \text{if } \delta(q, f) \text{ is undefined.} \end{cases}$$

The *transformation* T_M defined by M is a partial function mapping trees over Σ to words over Δ defined by $T_M(t) = u_0 \cdot T_{q_0}(t) \cdot u_1$, where $init = u_0 \cdot q_0 \cdot u_1$. Two transducers are *equivalent* iff they define the same transformation.

Example 1. We fix the input alphabet $\Sigma = \{f^{(2)}, g^{(1)}, a^{(0)}\}$ and the output alphabet $\Delta = \{a, b, c\}$. The STW M_1 has the initial rule q_0 and the following transition rules:

$$\delta(q_0, f) = q_1 \cdot ac \cdot q_1, \qquad \delta(q_1, g) = q_1 \cdot abc, \qquad \delta(q_1, a) = \varepsilon.$$

It defines the transformation $T_{M_1}(f(g^m(a), g^n(a))) = (abc)^m ac(abc)^n$, where $m, n \geq 0$, and T_{M_1} is undefined on all other input trees. The STW M_2 has the initial rule p_0 and these transition rules:

$$\delta(p_0, f) = p_1 \cdot p_3 \cdot ab, \quad \delta(p_1, g) = a \cdot p_2, \quad \delta(p_2, g) = ab \cdot p_3, \quad \delta(p_3, g) = p_3,$$
$$\delta(p_0, a) = ba, \quad \delta(p_1, a) = \varepsilon, \quad \delta(p_2, a) = \varepsilon, \quad \delta(p_3, a) = \varepsilon.$$

Now, $T_{M_2}(a) = ba$ and for $n \geq 0$, the result of $T_{M_2}(f(g^m(a), g^n(a))$ is ab for $m = 0$, aab for $m = 1$, and $aabab$ for $m \geq 2$; T_{M_2} is undefined for all other input trees. Note that p_3 is a *deleting* state: it does not produce any output but allows to check that the input tree belongs to the domain of the transducer. □

In the sequel, to simplify notation we assume every state belongs to exactly one transducer, and so T_q above is defined in unambiguous manner. We consider only trimmed STWs i.e., transducers where all states define a nonempty transformation and are accessible from the initial rule. Also, by dom_q we denote the set $dom(T_q)$, the domain of T_q i.e., the set of trees on which T_q is defined, and by L_q the range of T_q i.e., the set of words returned by T_q. For instance, $dom_{q_0} = \{f(g^m(a), g^n(a) \mid m, n \geq 0\}$ and $L_{q_0} = (abc)^* ac(abc)^*$. We observe that dom_q is a regular tree language and L_q is a context-free word language (CFL).

Next, we introduce the notion of being *earliest* that allows us to identify normal forms of transformations definable with STWs. It is a challenging task because the notion of being earliest needs to be carefully crafted so that every transducer can be made earliest. Take, for instance, the transformation *turn* that takes a tree over $\Sigma = \{a^{(1)}, b^{(1)}, \bot^{(0)}\}$ and returns the sequence of its labels in the reverse order e.g., $turn(a(b(b(\bot)))) = bba$. It is definable with a simple STW.

$$\delta(q_{turn}, a) = q_{turn} \cdot a, \qquad \delta(q_{turn}, b) = q_{turn} \cdot b, \qquad \delta(q_{turn}, \bot) = \varepsilon.$$

One way to view the transformation is a preorder traversal of the input tree that produces one output word upon entering the node and another word prior to leaving the node. When analyzing *turn* from this perspective, the earliest moment to produce any output is when the control reaches $\bot$, and in fact, the whole output can be produced at that point because all labels have been seen. This requires storing the label sequence in memory, which is beyond the capabilities of a finite state machine, and thus, *turn* cannot be captured with a transducer satisfying this notion of being earliest.

We propose a notion of being *earliest* that is also based on preorder traversal but with the difference that both output words are specified on entering the node and the output of a node is constructed right before leaving the node. Intuitively, we wish to *push up* all possible factors in the rules. Clearly, the STW above satisfies the condition. We remark that in some cases the output words in the rule can be placed in several positions, e.g. the rule $\delta(q_1, g) = q_1 \cdot abc$ in M_1 (Examples 1) can be replaced by $\delta(q_1, g) = abc \cdot q_1$ without changing the semantics of M_1. Consequently, we need an additional requirement that resolves this ambiguity: intuitively, we wish to *push left* the words in a rule as much as possible.

Definition 2. *An* STW $M = (\Sigma, \Delta, Q, init, \delta)$ *is* earliest (eSTW) *iff the following two conditions are satisfied:*

(E$_1$) $lcp(L_q) = \varepsilon$ *and* $lcs(L_q) = \varepsilon$ *for every state* q,
(E$_2$) $lcp(L_{q_0} \cdot u_1) = \varepsilon$ *for the initial rule* $u_0 \cdot q_0 \cdot u_1$ *and for every transition* $\delta(q, f) = u_0 \cdot q_1 \cdot \ldots \cdot q_k \cdot u_k$ *and* $1 \leq i \leq k$ *we have* $lcp(L_{q_i} \cdot u_i \cdot \ldots \cdot L_{q_k} \cdot u_k) = \varepsilon$.

Intuitively, the condition **(E$_1$)** ensures that no factor can be pushed up in the traversal and **(E$_2$)** ensures that no factor can be pushed left. We note that **(E$_1$)** and **(E$_2$)** can be efficiently checked in an STW because we need only to check that the results of *lcp* and *lcs* are ε. The main contribution of this paper follows.

Theorem 1. *For every* STW *there exists a unique minimal equivalent* eSTW.

The proof consists of an effective procedure that works in two stages: In the first stage we normalize the outputs, i.e. from the input STW we construct an equivalent eSTW, and in the second stage we minimize the obtained eSTW. The first stage is, however, quite complex as illustrated in the following example.

Example 2 (contd. Example 1). M_1 is not earliest because **(E$_1$)** is not satisfied at q_0: every word of $L_{q_0} = (abc)^* ac (abc)^*$ begins with a i.e., $lcp(L_{q_0}) = a$, and ends with c i.e., $lcs(L_{q_0}) = c$. Consequently, we need to *push up* these two symbols to the new initial rule $a \cdot q_0' \cdot c$, but we also need to retract them from the rule $\delta(q_0, f) = q_1 \cdot ac \cdot q_1$ producing a new state q_0' and new rules for this state. Essentially, we need to push the symbol a to the left through the first occurrence of q_1 and push the symbol c to the right through the second occurrence of q_1. Pushing symbols through states produces again new states with rules obtained by reorganizing the output words. Finally, we obtain

$$\delta'(q_0', f) = q_1' \cdot q_1'', \quad \delta'(q_1', g) = bca \cdot q_1', \quad \delta'(q_1'', g) = cab \cdot q_1'', \quad \delta'(q_1', a) = \delta'(q_1'', a) = \varepsilon.$$

M_2 is not earliest because **(E$_2$)** is not satisfied by $\delta(p_0, f) = p_1 \cdot p_3 \cdot ab$: every word produced by this rule starts with a. First, we push the word ab through the state p_3, and then we push the symbol a through the state p_1. Pushing through p_3 is easy because it is a deleting state and the rules do not change. Pushing through p_1 requires a recursive push through the states of the rules of p_1 and this process affects the rules of p_2. Finally, we obtain an eSTW with the initial rule p_0' and the transition rules

$$\begin{array}{llll} \delta'(p_0', f) = a \cdot p_1' \cdot b \cdot p_3', & \delta'(p_1', g) = a \cdot p_2', & \delta'(p_2', g) = ba \cdot p_3', & \delta'(p_3', g) = p_3', \\ \delta'(p_0', a) = ba, & \delta'(p_1', a) = \varepsilon, & \delta'(p_2', a) = \varepsilon, & \delta'(p_3', a) = \varepsilon. \quad \square \end{array}$$

3 Output Normalization

The first phase of normalization of an STW consists of constructing an equivalent eSTW, which involves changing the placement of the factors in the rules of the transducer and deals mainly with the output. Consequently, we begin with several notions and constructions inspired by the conditions **(E$_1$)** and **(E$_2$)** but set in a simpler setting of word languages. We consider only nonempty languages because in trimmed STWs the ranges of the states are always nonempty.

3.1 Reducing Languages

Enforcement of **(E$_1$)** corresponds to what we call constructing the *reduced decomposition* of a language. A nonempty language L is *reduced* iff $lcp(L) = \varepsilon$ and $lcs(L) = \varepsilon$. Note that the assumption that we work with a nonempty language is essential here. Now, take a nonempty language L, that is not necessarily reduced. We decompose it into its *reduced core* $Core(L)$ and two words $Left(L)$ and $Right(L)$ such that $Core(L)$ is reduced and

$$L = Left(L) \cdot Core(L) \cdot Right(L). \tag{1}$$

We observe that different decompositions are possible. For instance, $L = \{a, aba\}$ has two decompositions $L = a \cdot \{\varepsilon, ba\} \cdot \varepsilon$ and $L = \varepsilon \cdot \{\varepsilon, ab\} \cdot a$. We resolve the ambiguity by choosing the former decomposition because it is consistent with **(E$_1$)** and **(E$_2$)** which indicate to *push to the left*. Formally, $Left(L) = lcp(L)$ and $Right(L) = lcs(L')$, where $L = Left(L) \cdot L'$. The reduced core $Core(L)$ is obtained from (1). As an example, the reduced decomposition of $L_{q_0} = (abc)^*ac(abc)^*$ from Example 1 is $Left(L_{q_0}) = a$, $Right(L_{q_0}) = c$, and $Core(L_{q_0}) = (bca)^*(cba)^*$.

3.2 Pushing Words through Languages

In this subsection, we work with *nonempty* and *reduced* languages only. Condition **(E$_2$)** introduces the problem that we call pushing words through languages. To illustrate it, suppose we have a language $L = \{\varepsilon, a, aa, aaab\}$ and a word $w = aab$, which together give $L \cdot w = \{aab, aaab, aaaab, aaabaab\}$. The goal is to find the longest prefix v of w such that $L \cdot w = v \cdot L' \cdot u$, where $w = v \cdot u$ and L' is some derivative of L. Intuitively speaking, we wish to push (a part of) the word w forward i.e., from right to left, through the language L. In the example above, the solution is $v = aa$, $L' = \{\varepsilon, a, aa, abaa\}$, and $u = b$ (note that L' is different from L). In this section, we show that this process is always feasible and for CFLs it is constructive.

The result of pushing a word w through a language L will consist of three words: $push(L, w)$ the longest part of w that can be pushed through L, $rest(L, w)$ the part that cannot be pushed through, and $offset(L, w)$ a special word that allows to identify the corresponding derivative of L. There are three classes of languages that need to be considered, which we present next together with an outline of how the pushing is done.

The first class contains only the *trivial* language $L = \{\varepsilon\}$ e.g., the range of the state p_3 of M_2 in Example 1. This language allows every word to be pushed through and it never changes in the process. For instance, if $w_0 = ab$, then $push(L_{p_3}, w_0) = ab$, $rest(L_{p_3}, w_0) = \varepsilon$, and $offset(L_{p_3}, w_0) = \varepsilon$.

The second class consists of non-trivial periodic languages, essentially languages contained in the Kleene closure of some period word. An example is $L_{q_1} = (abc)^* = \{\varepsilon, abc, abcabc, \ldots\}$ whose period is abc. Periodic languages allow to push multiplicities of the period and then some prefix of the period e.g., if we take $w_1 = abcabcaba$, then $push(L_{q_1}, w_1) = abcabcab$ and $rest(L_{q_1}, w_1) = a$. The offset here is the corresponding prefix of the period: $offset(L_{q_1}, w_1) = ab$.

The third class contains all remaining languages i.e., non-trivial non-periodic languages. Interestingly, we show that for a language in this class there exists a word that is the longest word that can be pushed fully through the language, and furthermore, every other word that can be pushed through is a prefix of this word. For instance, for $L_{p_1} = \{\varepsilon, a, aab\}$ from Example 1, aa is the longest word that can be pushed through. If we take $w_2 = ab$, then we get $push(L_{p_1}, w_2) = a$ and $rest(L_{p_1}, w_2) = b$. Here, the offset is the prefix of aa that has been already pushed through: $offset(L_{p_1}, w_2) = a$. Note that this class also contains the languages that do not permit any pushing through e.g., $L_{p_0} = \{ba, ab, aab\}$ does not allow pushing through because it contains two words that start with a different symbol.

We now define formally the pushing process. First, for $L \subseteq \Delta^*$ we define the set of words that can be pushed fully through L:

$$Shovel(L) = \{w \in \Delta^* \mid w \text{ is a common prefix of } L \cdot w\}.$$

For instance, $Shovel(L_{p_1}) = \{\varepsilon, a, aa\}$ and $Shovel(L_{q_0}) = (abc)^* \cdot \{\varepsilon, a, ab\}$. We note that $Shovel(\{\varepsilon\}) = \Delta^*$ and $Shovel(L)$ always contains at least one element ε because L is assumed to be nonempty. Also, as we prove in appendix, $Shovel(L)$ is prefix-closed and totally ordered by the prefix relation.

Next, we define periodic languages (cf. [12]). A language $L \subseteq \Delta^*$ is *periodic* iff there exists a nonempty word $v \in \Delta^*$, called a *period* of L, such that $L \subseteq v^*$. A word w is *primitive* if there is no v and $n \geq 0$ such that $w = v^n$. Recall from [12] that every non-trivial periodic language L has a unique primitive period, which we denote $Period(L)$. For instance, the language $\{\varepsilon, abab, abababab\}$ is periodic and its primitive period is ab; $abab$ is also its period but not primitive. In the sequel, by $Prefix(w)$ we denote the set of prefixes of the word w.

Proposition 1. *Given a reduced and non-trivial language L, $Shovel(L)$ is infinite iff L is periodic. Furthermore, if L is periodic then $Shovel(L) = Period(L)^* \cdot Prefix(Period(L))$.*

This result and the observations beforehand lead to three relevant cases in the characterisation of $Shovel(L)$ for a language L.

0^o $L = \{\varepsilon\}$ (trivial language), and then $Shovel(L) = \Delta^*$,
1^o L is periodic, $L \neq \{\varepsilon\}$, and then $Shovel(L) = Period(L)^* \cdot Prefix(Period(L))$.
2^o L is non-periodic, and $Shovel(L) = Prefix(v)$ for some $v \in Shovel(L)$.

Now, suppose we wish to *push* a word $w \in \Delta^*$ *through* a language $L \subseteq \Delta^*$ and let $s = \max_{\leq_{prefix}}(Prefix(w) \cap Shovel(L))$ and $w = s \cdot r$. We define $push(L, w)$, $rest(L, w)$, and $offset(L, w)$ depending on the class L belongs to:

0^o $L = \{\varepsilon\}$: $push(L, w) = w$, $rest(L, w) = \varepsilon$, and $offset(L, w) = \varepsilon$.
1^o L is non-trivial and periodic: $s = Period(L)^k \cdot o$ for some (maximal) proper prefix o of $Period(L)$, and we assign $push(L, w) = s$, $rest(L, w) = r$, and $offset(L, w) = o$.
2^o L is non-periodic: $push(L, w) = s$, $rest(L, w) = r$, and $offset(L, w) = s$.

Offsets play a central role in the output normalization procedure, which is feasible thanks to the following result.

Proposition 2. *The set $\{offset(L, w) \mid w \in \Delta^*\}$ is finite for any reduced L.*

3.3 Pushing Words Backwards

Until now, we have considered the problem of pushing a word through a language from right to left. However, in Example 1 if we consider the second occurrence of q_1 in the rule $\delta(q_0, f) = q_1 \cdot ac \cdot q_1$, we realize that pushing words in the opposite direction needs to be investigated as well. These two processes are dual but before showing in what way, we present a natural extension of the free monoid Δ^* to a pregroup (or groupoid) $\mathbb{G}_\Delta$. It allows to handle pushing in two directions in a unified manner and simplifies the output normalization algorithm.

A *pregroup of words over* Δ is the set $\mathbb{G}_\Delta = \Delta^* \cup \{w^{-1} \mid w \in \Delta^+\}$, where w^{-1} is a term representing the inverse of an nonempty word w. This set comes with two operators, a unary inverse operator: $(w)^{-1} = w^{-1}$, $\varepsilon^{-1} = \varepsilon$, and $(w^{-1})^{-1} = w$ for $w \in \Delta^*$, and a partial extension of the standard concatenation that satisfies the following equations (complete definition in appendix): $w^{-1} \cdot w = \varepsilon$ and $w \cdot w^{-1} = \varepsilon$ for $w \in \Delta^*$, and $v^{-1} \cdot u^{-1} = (uv)^{-1}$ for $u, v \in \Delta^*$. We note that some expressions need to be evaluated diligently e.g., $ab \cdot (cb)^{-1} \cdot cd = ab \cdot b^{-1} \cdot c^{-1} \cdot cd = ad$, while some are undefined e.g., $ab \cdot a^{-1}$. In the sequel, we use $w, u, v, \ldots$ to range over Δ^* only and $z, z_1, \ldots$ to range over elements of $\mathbb{G}_\Delta$.

Now, we come back to pushing a word w backwards through L, which consists of finding $u \cdot v = w$ and L' such that $w \cdot L = u \cdot L' \cdot v$. We view this process as pushing the inverse w^{-1} through L i.e., we wish to find $u \cdot v = w$ such that $L \cdot w^{-1} = v^{-1} \cdot L' \cdot u^{-1}$ because then $L \cdot v^{-1} = v^{-1} \cdot L'$, and consequently, $w \cdot L = (u \cdot v) \cdot (v^{-1} \cdot L' \cdot v) = u \cdot L' \cdot v$.

But to define pushing backwards more properly we use another perspective based on the standard reverse operation of a word e.g., $(abc)^{\mathrm{rev}} = cba$. Namely, pushing w backwards through L is essentially pushing w^{rev} through L^{rev} because $(w \cdot L)^{\mathrm{rev}} = L^{\mathrm{rev}} \cdot w^{\mathrm{rev}}$ and if $L^{\mathrm{rev}} \cdot w^{\mathrm{rev}} = v_0 \cdot L_0 \cdot u_0$, then $w \cdot L = u_0^{\mathrm{rev}} \cdot L_0^{\mathrm{rev}} \cdot v_0^{\mathrm{rev}}$. Thus $\mathit{push}(L, w^{-1}) = (\mathit{push}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1}$, $\mathit{rest}(L, w^{-1}) = (\mathit{rest}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1}$, and $\mathit{offset}(L, w^{-1}) = (\mathit{offset}(L^{\mathrm{rev}}, w^{\mathrm{rev}})^{\mathrm{rev}})^{-1}$.

Now, the main condition of pushing words through languages is: for every L and $z \in \mathbb{G}_\Delta$ we have $L \cdot z = \mathit{push}(L, z) \cdot (\mathit{offset}(L, z)^{-1} \cdot L \cdot \mathit{offset}(L, z)) \cdot \mathit{rest}(L, z)$. Because the output normalization procedure works on STWs and not languages, to prove its correctness we need a stronger statement treating independently every word of the language.

Proposition 3. *Given a reduced and nonempty language* $L \subseteq \Delta^*$ *and* $z \in \mathbb{G}_\Delta$, *for any word* $u \in L$

$$u \cdot z = \mathit{push}(L, z) \cdot (\mathit{offset}(L, z)^{-1} \cdot u \cdot \mathit{offset}(L, z)) \cdot \mathit{rest}(L, z).$$

3.4 Output Normalization Algorithm

We fix an STW $M = (\Sigma, \Delta, Q, \mathit{init}, \delta)$ and introduce the following macros:

$$\hat{L}_q = \mathit{Core}(L_q), \quad \mathit{Left}(q) = \mathit{Left}(L_q), \quad \mathit{Right}(q) = \mathit{Right}(L_q),$$
$$\mathit{push}(q, z) = \mathit{push}(\hat{L}_q, z), \quad \mathit{offset}(q, z) = \mathit{offset}(\hat{L}_q, z), \quad \mathit{rest}(q, z) = \mathit{rest}(\hat{L}_q, z).$$

Also, let $\mathit{Offsets}(q) = \{\mathit{offset}(q, z) \mid z \in \mathbb{G}_\Delta\}$ and note that by Proposition 2 it is finite. The constructed STW $M' = (\Sigma, \Delta, Q', \mathit{init}', \delta')$ has the following states

$$Q' = \{\langle q, w\rangle \mid q \in Q,\ w \in \mathit{Offsets}(q)\}.$$

Our construction ensures that $T_M = T_{M'}$ and for every $q \in Q$, every $z \in \mathit{Offsets}(q)$, and every $t \in \mathit{dom}_q$

$$T_{\langle q,z\rangle}(t) = z^{-1} \cdot \mathit{Left}(q)^{-1} \cdot T_q(t) \cdot \mathit{Right}(q)^{-1} \cdot z$$

If $\mathit{init} = u_0 \cdot q_0 \cdot u_1$, then $\mathit{init}' = u_0' \cdot q_0' \cdot u_1'$, where u_0', u_1', and q_0' are calculated as follows:

1: $v := \mathit{Right}(q_0) \cdot u_1$
2: $q_0' := \langle q_0, \mathit{offset}(q_0, v)\rangle$
3: $u_0' := u_0 \cdot \mathit{Left}(q_0) \cdot \mathit{push}(q_0, v)$
4: $u_1' := \mathit{rest}(q_0, v)$

For a transition rule $\delta(p, f) = u_0 \cdot p_1 \cdot u_1 \cdot \ldots \cdot u_{k-1} \cdot p_k \cdot u_k$ and any $z \in \mathit{Offsets}(p)$ we introduce a rule $\delta'(\langle p, z\rangle, f) = u_0' \cdot p_1' \cdot u_1' \cdot \ldots \cdot u_{k-1}' \cdot p_k' \cdot u_k'$, where $u_0', \ldots, u_k'$ and $p_1', \ldots p_k'$ are calculated as follows:

1: $z_k := \mathit{Right}(p_k) \cdot u_k \cdot \mathit{Right}(p)^{-1} \cdot z$
2: **for** $i := k, \ldots, 1$ **do**
3: $\quad u_i' := \mathit{rest}(p_i, z_i)$
4: $\quad p_i' := \langle p_i, \mathit{offset}(p_i, z_i)\rangle$
5: $\quad z_{i-1} := \mathit{Right}(p_{i-1}) \cdot u_{i-1} \cdot \mathit{Left}(p_i) \cdot \mathit{push}(p_i, z_i)$
6: $u_0' := z^{-1} \cdot \mathit{Left}(p)^{-1} \cdot z_0$

where (for convenience of the presentation) we let $\mathit{Right}(p_0) = \varepsilon$. We remark that not all states in Q' are reachable from the initial rule and in fact the conversion procedure can identify the reachable states *on the fly*. This observation is the basis of a conversion algorithm that is polynomial in the size of the output.

Example 3. We normalize the STW M_1 from Example 1. The initial rule q_0 becomes $a \cdot \langle q_0, \varepsilon\rangle \cdot c$ with $\mathit{Left}(q_0) = a$ and $\mathit{Right}(q_0) = c$ being pushed up from q_0 but with nothing pushed through q_0. The construction of the state $\langle q_0, \varepsilon\rangle$ triggers the normalization algorithm for the rule $\delta(q_0, f) = q_1 \cdot ac \cdot q_1$ with $\mathit{Left}(q_0) = a$ and $\mathit{Right}(q_0) = c$ to be retracted from left and right side resp. (and nothing pushed through since $z = \varepsilon$). This process can be viewed as a taking the left hand side of the original rule with the inverses of retracted words $a^{-1} \cdot q_1 \cdot ac \cdot q_1 \cdot c^{-1}$ and pushing words forward as much as possible, which gives $a^{-1} \cdot q_1 \cdot ac \cdot c^{-1} \cdot \langle q_1, c^{-1}\rangle$ and then $a^{-1} \cdot a \cdot \langle q_1, a\rangle \cdot \langle q_1, c^{-1}\rangle$. This gives $\delta'(\langle q_0, \varepsilon\rangle, f) = \langle q_1, a\rangle \cdot \langle q_1, c^{-1}\rangle$. Note that while $\mathit{Offsets}(q_1) = \{(bc)^{-1}, c^{-1}, \varepsilon, a, ab\}$, only two states are constructed.

Next, we need to construct rules for the new state $\langle q_1, a\rangle$ with $z = a$ and $\mathit{Left}(q_1) = \mathit{Right}(q_1) = \varepsilon$. We start with the rule $\delta(q_1, a) = \varepsilon$ and to its left hand side we add a^{-1} at the beginning and a at its end: $a^{-1} \cdot \varepsilon \cdot a = \varepsilon$, which yields the rule $\delta'(\langle q_1, a\rangle, a) = \varepsilon$. Now, for the rule $\delta(q_1, g) = q_1 \cdot abc$ we obtain the expression $a^{-1} \cdot q_1 \cdot abca$. Recall that $L_{q_1} = (abc)^*$ is a periodic language, and so

$push(q_1, abca) = abca$, $rest(q_1, abca) = \varepsilon$, and $offset(q_1, abca) = a$. Consequently, we obtain the rule $\delta'(\langle q_1, a\rangle, g) = bca \cdot \langle q_1, a\rangle$. Here, it is essential to use the offsets to avoid introducing a redundant state $\langle q_1, abca\rangle$ and entering an infinite loop. Similarly, we obtain: $\delta'(\langle q_1, c^{-1}\rangle, g) = cab \cdot \langle q_1, c^{-1}\rangle$ and $\delta'(\langle q_1, c^{-1}\rangle, a) = \varepsilon$. □

Theorem 2. *For an* STW M *let* M' *be the* STW *obtained with the method described above. Then,* M' *is equivalent to* M *and satisfies* $\mathbf{(E_1)}$ *and* $\mathbf{(E_2)}$*. Furthermore,* M' *can be constructed in time polynomial in the size* M'*, which is at most doubly-exponential in the size of* M*.*

Because of space limitations, the details on complexity have been omitted and can be found in the full version available online.

3.5 Exponential Lower Bound

First, we show that the size of a rule may increase exponentially.

Example 4. For $n \geq 0$ define an STW M_n over the input alphabet $\Sigma = \{f^{(2)}, a^{(0)}\}$ with the initial rule q_0, and these transition rules (with $0 \leq i < n$):

$$\delta(q_i, f) = q_{i+1} \cdot q_{i+1}, \qquad \delta(q_n, a) = a.$$

The transformation defined by M_n maps a perfect binary tree of height n to a string a^{2^n}. M_n is not earliest. To make it earliest we need to replace the initial rule by $a^{2^n} \cdot q_0(x_0)$ and the last transition rule by $\delta(q_n, a) = \varepsilon$. □

The next example shows that also the number of states may become exponential.

Example 5. For $n \geq 0$ and take the STW N_n with $\Sigma = \{g_1^{(1)}, g_0^{(1)}, a_1^{(0)}, a_0^{(0)}\}$, the initial rule q_0, and these transition rules (with $0 \leq i < n$):

$$\delta(q_i, g_0) = q_{i+1}, \qquad \delta(q_n, a_0) = \varepsilon,$$
$$\delta(q_i, g_1) = q_{i+1} \cdot a^{2^i}, \qquad \delta(q_n, a_1) = a^{2^n} \cdot \#.$$

While the size of this transducer is exponential in n, one can easily compress the exponential factors 2^{2^i} and obtain an STW of size linear in n (cf. Example 4). M_n satisfies $\mathbf{(E_1)}$ but it violates $\mathbf{(E_2)}$, and defines the following transformation.

$$T_{N_n} = \{(g_{b_0}(g_{b_1}(\ldots g_{b_{n-1}}(a_0)\ldots)), a^{\mathbf{b}}) \mid \mathbf{b} = (b_{n-1}, \ldots, b_0)_2\} \cup$$
$$\{(g_{b_0}(g_{b_1}(\ldots g_{b_{n-1}}(a_1)\ldots)), a^{2^n} \cdot \# \cdot a^{\mathbf{b}}) \mid \mathbf{b} = (b_{n-1}, \ldots, b_0)_2\},$$

where $(b_{n-1}, \ldots, b_0)_2 = \sum_i b_i * 2^i$. The normalized version N'_n has the initial rule $\langle q_0, \varepsilon\rangle$ and these transition rules:

$$\delta'(\langle q_i, a^j\rangle, g_0) = \langle q_{i+1}, a^j\rangle, \qquad \delta'(\langle q_n, a^k\rangle, a_0) = \varepsilon,$$
$$\delta'(\langle q_i, a^j\rangle, g_1) = a^{2^i} \cdot \langle q_{i+1}, a^{j+2^i}\rangle, \qquad \delta'(\langle q_n, a^k\rangle, a_1) = a^{2^n - k} \# a^k,$$

where $0 \leq i < n$, $0 \leq j < 2^i$, and $0 \leq k < 2^n$. We also remark that N'_n is the minimal eSTW that recognises T_{N_n}. □

4 Minimization

In this section we investigate the problem of minimizing the size of a transducer. Minimization of eSTWs is simple and relies on testing the equivalence of eSTWs known to be in PTIME [16]. For an eSTW M the minimization procedure constructs a binary equivalence relation $\equiv_M$ on states such that $q \equiv_M q'$ iff $T_q = T_{q'}$. The result of minimization is the quotient transducer $M/_{\equiv_M}$ obtained by choosing in every equivalence class C of $\equiv_M$ exactly one representative state $q \in C$, and then replacing in rules of M every state of C by q.

To show that the obtained eSTW is minimal among all eSTWs defining the same transformation, we use an auxiliary result stating that all eSTWs defining the same transformation use rules with the same distribution of the output words and allow bisimulation.

A *labeled path* is a word over $\bigcup_{k>0} \Sigma^{(k)} \times \{1, \ldots, k\}$, which identifies a node in a tree together with the labels of its ancestors: ε is the root node and if a node π is labeled with f, then $\pi \cdot (f, i)$ is its i-th child. By $paths(t)$ we denote the set of labeled paths of a tree t. For instance, for $t_0 = f(a, g(b))$ we get $paths(t_0) = \{\varepsilon, (f,1), (f,2), (f,2)\cdot(g,1)\}$. We extend the transition function δ to identify the state reached at a path π: $\delta(q, \varepsilon) = q$ and $\delta(q, \pi \cdot (f, i)) = q_i$, where $\delta(q, \pi) = q'$ and $\delta(q', f) = u_0 \cdot q_1 \cdot u_1 \cdot \ldots \cdot q_k \cdot u_k$. Now, the lemma of interest.

Lemma 1. *Take two* eSTWs $M = (\Sigma, \Delta, Q, init, \delta)$ *and* $M' = (\Sigma, \Delta, Q', init', \delta')$ *defining the same transformation* $T = T_M = T_{M'}$ *and let* $init = u_0 \cdot q_0 \cdot u_1$ *and* $init' = u'_0 \cdot q'_0 \cdot u'_1$. *Then,* $u_0 = u'_0$ *and* $u_1 = u'_1$, *and for every* $\pi \in paths(dom(T))$, *we let* $q = \delta(q_0, \pi)$ *and* $q' = \delta'(q'_0, \pi)$, *and we have*

1. $T_q = T_{q'}$,
2. $\delta(q, f)$ *is defined if and only if* $\delta'(q', f)$ *is, for every* $f \in \Sigma$, *and*
3. *if* $\delta(q, f) = u_0 \cdot q_1 \cdot u_1 \cdot \ldots \cdot q_k \cdot u_k$ *and* $\delta'(q', f) = u'_0 \cdot q'_1 \cdot u'_1 \cdot \ldots \cdot q'_k \cdot u'_k$, *then* $u_i = u'_i$ *for* $0 \leq i \leq k$.

The proof is inductive and relies on properties **(E$_1$)** and **(E$_2$)**, and the determinism of the transducers. We show the correctness of our minimization algorithm by observing that it produces an eSTW whose size is smaller than the input one, and Lemma 1 essentially states that the result of minimization of two equivalent transducers is the same transducer (modulo state renaming). This argument also proves Theorem 1. We also point out that Lemma 1 (with $M = M'$) allows to devise a simpler and more efficient minimization algorithm along the lines of the standard DFA minimization algorithm [10].

Theorem 3. *Minimization of* eSTWs *is in* PTIME.

In STWs the output words may be arbitrarily distributed among the rules, which is the main pitfall of minimizing general STWs. This difficulty is unlikely to be overcome as suggested by the following result.

Theorem 4. *Minimization of* STWs *i.e., deciding whether for an* STW M *and* $k \geq 0$ *there exists an equivalent* STW M' *of size at most* k, *is* NP*-complete.*

5 Conclusions and Future Work

We have presented an effective normalization procedure for STWs, a subclass of top-down tree-to-word transducers closely related to a large subclass of nested-word to word transducers. One natural continuation of this work is find whether it can be extended to a Myhill-Nerode theorem for STWs, and then, to a polynomial learning algorithm. Also, the question of exact complexity of the normalization remains open. Finally, the model of STWs can be generalized to allow arbitrary non-sequential rules and multiple passes over the input tree.

References

1. Alur, R., Kumar, V., Madhusudan, P., Viswanathan, M.: Congruences for visibly pushdown languages. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1102–1114. Springer, Heidelberg (2005)
2. Berstel, J., Boasson, L.: Transductions and context-free languages. Teubner Studienbucher (1979)
3. Choffru, C.: Contribution à l'étude de quelques familles remarquables de fonctions rationnelles. PhD thesis, Université de Paris VII (1978)
4. Choffrut, C.: Minimizing subsequential transducers: a survey. Theoretical Computer Science 292(1), 131–143 (2003)
5. Engelfriet, J., Maneth, S., Seidl, H.: Deciding equivalence of top-down XML transformations in polynomial time. Journal of Computer and System Science 75(5), 271–286 (2009)
6. Filiot, E., Raskin, J.F., Reynier, P.A., Servais, F., Talbot, J.M.: Properties of visibly pushdown transducers. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 355–367. Springer, Heidelberg (2010)
7. Friese, S., Seidl, H., Maneth, S.: Minimization of deterministic bottom-up tree transducers. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 185–196. Springer, Heidelberg (2010)
8. Griffiths, T.V.: The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. Journal of the ACM 15(3), 409–413 (1968)
9. Gurari, E.M.: The equivalence problem for deterministic two-way sequential transducers is decidable. SIAM Journal on Computing 11(3), 448–452 (1982)
10. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison-Wesley, Reading (2001)
11. Lemay, A., Maneth, S., Niehren, J.: A learning algorithm for top-down xml transformations. In: ACM Symposium on Principles of Database Systems (PODS), pp. 285–296 (2010)
12. Lothaire, M. (ed.): Combinatorics on Words, 2nd edn. Cambridge Mathematical Library. Cambridge University Press, Cambridge (1997)
13. Maneth, S.: Models of Tree Translation. PhD thesis, Leiden University (2003)
14. Martens, W., Neven, F., Gyssens, M.: Typechecking top-down XML transformations: Fixed input or output schemas. Information and Computation 206(7), 806–827 (2008)
15. Raskin, J.-F., Servais, F.: Visibly pushdown transducers. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 386–397. Springer, Heidelberg (2008)
16. Staworko, S., Laurence, G., Lemay, A., Niehren, J.: Equivalence of deterministic nested word to word transducers. In: Kutyłowski, M., Charatonik, W., Gębala, M. (eds.) FCT 2009. LNCS, vol. 5699, pp. 310–322. Springer, Heidelberg (2009)

Planarity of Knots, Register Automata and LogSpace Computability

Alexei Lisitsa, Igor Potapov, and Rafiq Saleh

Department of Computer Science, University of Liverpool, Ashton Building, Ashton St, Liverpool L69 3BX, UK
{Lisitsa,Potapov,R.A.Saleh}@liverpool.ac.uk

Abstract. In this paper we investigate the complexity of planarity of knot diagrams encoded by Gauss words, both in terms of recognition by automata over infinite alphabets and in terms of classical logarithmic space complexity. As the main result, we show that recognition of planarity of unsigned Gauss words can be done in deterministic logarithmic space and by deterministic register automata. We also demonstrate generic results on the mutual simulations between logspace bounded classical computations (over finite alphabets) and register automata working over infinite alphabets.

1 Introduction

Knot theory is the area of mathematics that studies mathematical knots and links. A knot (a link) is an embedding of a circle (several circles) in 3-dimensional Euclidean space, $\mathbb{R}^3$, considered up to a smooth deformation of the ambient space. It is a well established and active area of research with strong connections to topology, algebra and combinatorics. Some major problems in the main focus of knot theory have algorithmic or computational nature: *equivalence problem* (how to recognise that two knots are equivalent), or *unknottedness problem* (how to recognise that a knot is a trivial one). Consideration of such problems led to fruitful interactions between knot theory and computer science. In particular, the questions of computational complexity of knot problems have been addressed in [9]. Examples of other interactions include works on formal language theory [11] and quantum computing [1,7,16].

One of the earliest questions of algorithmic nature related to knots was the question of characterisation of Gauss words [8]. With every knot one can associate a word, called a Gauss word, which is a sequence of labels for the crossings read off directly from the projection of the knot on a plane. Depending on whether the information on the orientation is present the word can be signed or unsigned. The simple property of any Gauss word is that every label (index) in it appears twice. The converse is not true, there are the words with every symbol appearing twice which do not correspond to any classical planar knot diagram. The question of characterisation of "true", or planar Gauss words was posed by Gauss himself [8] and was eventually resolved by Nagy in [20]. Since then there has been proposed many criteria and algorithms both for recognition of signed [3,13] and unsigned [17,22,24,25,12,4,5,6,24,18,26] Gauss words. The questions

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 366–377, 2011.

of computational complexity of the proposed algorithms were rarely explicitly addressed with notable exceptions being [13] where linear time algorithm for the signed case is proposed, and in [25] where a linear time complexity for unsigned case is established and compared with earlier quadratic bounds in [22].

In [15] we proposed to evaluate the complexity of problems of recognising knot properties in terms of the computational power of devices needed to recognise the properties. Following the proposal we demonstrated lower and upper bounds for recognisability of knot properties in terms of various automata models over *infinite* alphabets (see Section 2.2 below). The infinite alphabet appeared naturally due to the fact that the number of crossings in knots is unbounded. The main property addressed was *planarity* of signed Gauss words [13] (see Section 3). We have shown that planarity of signed Gauss words can be recognised by deterministic register automata working over infinite alphabets. It follows that signed planarity problem is in $\mathcal{L}$ (deterministic logspace).

In this paper we continue this line of research focussing mainly on the *unsigned case*. Our contribution is as follows:
1) We provide an analysis of Cairns-Elton algorithm for unsigned planarity and show that it is implementable by co-non-deterministic register automata.
2) We demonstrate generic results on the mutual simulations between logspace bounded classical computations (over finite alphabets) and register automata working over infinite alphabets. New characterisation of languages recognisable by register automata is more general than one proposed in [21].
3) We further show that Cairns-Elton algorithm is implementable in $\mathcal{SL}$ (symmetric logspace) and therefore in $\mathcal{L}$. It follows that planarity of unsigned Gauss words is recognisable by deterministic register automata, refuting the conjecture from [15].

2 Preliminaries

We use standard notations from complexity theory and assume that the reader is familiar with the complexity classes $\mathcal{L}$ (deterministic logspace), $\mathcal{SL}$ (symmetric logspace), $\mathcal{NL}$ (nondeterministic logspace), **co-**$\mathcal{NL}$ (co-non-deterministic logspace) and $SPACE(f(n))$ (space is bounded by a function $f(n)$).

2.1 Automata over an Infinite Alphabet

Register automata are finite state machines equipped with a finite number of memory cells called registers which may hold values from an infinite alphabet. It is one of the weakest models of automata over infinite alphabets. It was initially introduced in [10] and then studied further in [21]. The model we consider is computationally equivalent to the original version in [21], which is modified by adding two extra rules (see Definition 1) [15].

Let D be an infinite set called an *alphabet*. A word, or a string over D, or shortly, D-word or D-string is a finite sequence $d_1, \ldots, d_n$ where $d_i \in D$, $i = 1, \ldots, n$. A language over D (D-language) is a set of D-words. For a word w and a symbol d denote by $\mid w \mid_d$ the number of occurrences of d in w. As usual $\mid w \mid$ denotes the length of the word w.

Definition 1. *[15,21] A non-deterministic two-way k-register automaton over an infinite alphabet D is a tuple* (Q, q_0, F, τ_0, P) *where* Q *is a finite set of states,* $q_0 \in Q$ *is the initial state,* $F \subseteq Q$ *is the set of final states,* $\tau_0 : \{1,\ldots,k\} \to D \cup \{\triangleright, \triangleleft\}$ *is the initial register assignment and* P *is a finite set of transitions :*

1) $(i, q) \to (q', d)$ (If a current state is q and the observed symbol on the tape equals to a value in the register i then enter the state q' and move along the string according to the specified direction d where $i \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$)
2) $q \to (q', i, d)$ (If a current state is q and the observed symbol on the tape does not equal to any value held in registers then enter the state q', copy the current symbol to a specified register i and move along the string according to the specified direction d, where $i \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$).
3) $(i, q) \to (q', j, d)$ (If a current state is q and the observed symbol equals to a value in the register i then enter the state q', copy the current symbol to a register j and move along the string according to the specified direction d where $i, j \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$).
4) $q \to (q', d)$ (If a current state is q and the observed symbol does not equal to any value held in registers then enter the state q' and move along the string according to the specified direction d, where $q, q' \in Q$ and $d \in \{stay, left, right\}$).

Deterministic and co-non-deterministic register automata as well as the language accepted by automata are defined in the standard way. We denote by DRA, NRA, coNRA to be the classes of languages recognisable by deterministic, nondeterministic and co-non-deterministic two-way register automata, respectively.

Words and Data Words. In previous works on the computational models on infinite alphabets it has been acknowledged that in many situations it is natural to consider infinite alphabets as the subsets of $\Sigma \times \Delta$ where Σ is a *finite* set and Δ is an infinite set. Thus, the symbol here is an ordered pair (a, b). The words over such alphabets are called *data words* [2]. In particular Gauss words are introduced in Section 2.2 as natural instances of data words. In the definition of automata over data words, it is sensible to assume that when an automaton reads a symbol (a, b) it has a direct access to both components of the pair. For this purpose, the form of transition rules can be adapted to include one extra argument on the left-hand sides.

2.2 Knots

A knot is defined as a closed, non-self-intersecting curve that is embedded in $\mathbb{R}^3$. Knots can be described in many ways, including various discrete representations. A common method of representing a knot is a knot diagram that is a projection of the knot to a plane in a general position involving only double crossings.

At each crossing we indicate which branch is "over" and which is "under", which allows us to recreate the original knot. For oriented knots each crossing has a well-defined sign ('+' or '−'). To determine the sign of each crossing, we label the crossing with a + if we can rotate its under-strand clockwise to make

the arrows line up and a − if the rotation of the under-strand is counter-clockwise (see the picture in left-hand side of Figure 1).

A knot diagram can be encoded by a string of symbols O_i's (over) and U_i's (under) known as a *Gauss word*. The procedure of writing a Gauss word can be described as follows. Starting from a base point on a knot diagram, write down the labels of the crossings and their types of strand ordered according to the orientation of the knot. The oriented trefoil in Figure 1 is encoded by the *signed* Gauss word $O_1{}^+U_2{}^+O_3{}^+U_1{}^+O_2{}^+U_3{}^+$. Removing signs leads to the *unsigned* Gauss word. Indices of crossings can be arbitrarily permuted, so one knot diagram can be encoded by many Gauss words. A *shadow* Gauss word can be obtained from an unsigned Gauss word by removing the Us and Os from each label.

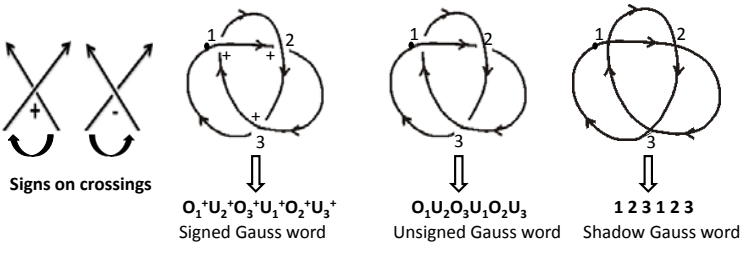


Fig. 1. Discrete representations of a knot

Definition 2. *An unsigned Gauss word w is a data word over the alphabet $\Sigma \times \mathbb{N}$ where $\Sigma = \{U, O\}$, such that for every $n \in \mathbb{N}$ either*

- $|w|_{(U,n)} = |w|_{(O,n)} = 0$, *or*
- $|w|_{(U,n)} = |w|_{(O,n)} = 1$.

Definition 3. *A signed Gauss word w is a data word over the alphabet $\Sigma \times \mathbb{N}$ where $\Sigma = \{U^+, O^+, U^-, O^-\}$, such that for every n either*

- $|w|_{(U^+,n)} = |w|_{(O^+,n)} = |w|_{(U^-,n)} = |w|_{(O^-,n)} = 0$, *or*
- $|w|_{(U^+,n)} = |w|_{(O^+,n)} = 1$ *and* $|w|_{(U^-,n)} = |w|_{(O^-,n)} = 0$, *or*
- $|w|_{(U^-,n)} = |w|_{(O^-,n)} = 1$ *and* $|w|_{(U^+,n)} = |w|_{(O^+,n)} = 0$.

Definition 4. *A shadow projection of a (signed/unsigned) Gauss word $w = (A_1, i_1), ..., (A_n, i_n)$ is a shadow Gauss word $sp(w) = i_1, ..., i_n$ where $A_i \in \Sigma$ and $i_j \in \mathbb{N}$.*

Definition 5. *A signing s is a mapping $s : \mathbb{N} \to \{+, -\}$.*

2.3 Planarity of Knot Diagrams Represented by Gauss Words

Every knot diagram can be represented by a Gauss word but not every Gauss word represents a knot diagram. The fact that every knot can be represented by a Gauss word directly follows from the constructive definition of a Gauss word and the second fact that not every Gauss word represents a classical knot in $\mathbb{R}^3$ is

illustrated in Figure 2. For example, any attempt to reconstruct a knot diagram from the Gauss word $O_1O_2U_1O_3U_2U_3$ will lead to new (virtual) crossings which are not present in the Gauss word (virtual crossings are marked on the diagram in Figure 2 with a small circle around the crossing). Such an observation was one of the motivations for introducing virtual knot theory [12]. A Gauss word which represents a classical knot diagram, that is a diagram embeddable into a plane without virtual crossings, is called classical or planar. The question about

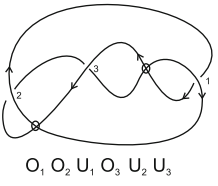


Fig. 2. Non-planar knot diagram

planarity of knot diagrams encoded by Gauss words is formulated separately for signed and unsigned cases. For Signed Gauss words the planarity question can be described as given a signed Gauss word w, does w represents a planar knot diagram? We will refer to this problem as SIGNED PLANARITY. For unsigned Gauss words, the question is stated in the following way: Given an unsigned Gauss word w, does there exist a choice of signings that can be assigned to w such that w represents a planar knot diagram? This problem will be referred to as UNSIGNED PLANARITY.

In [15], we showed that an algorithm for the decision of planarity of unsigned Gauss words proposed by Kauffman in [12] can be implemented by Linear Bounded Memory Automata over an infinite alphabet and conjectured that UNSIGNED PLANARITY is not recognisable by non-deterministic register automata. Here, we refute this conjecture and show that UNSIGNED PLANARITY can be recognised by deterministic register automata.

3 Main Results

In this section we present an upper bound for the planarity problem of unsigned Gauss words by first showing that it is recognisable by a co-non-deterministic register automata. Then we refine this result to show that the only part that requires non-determinism can be reduced to the search of cycles with simply checkable properties. We further show that the search is implementable in deterministic logspace. Finally, we show that deterministic logspace computations can be modelled by a deterministic register automata.

3.1 Cairns-Elton Algorithm

Cairns and Elton presented an algorithm in [4] which deals with UNSIGNED PLANARITY. We will first begin with definitions that will be used to describe the main steps of the algorithm.

Definition 6. *For a Gauss word w, denote by $\alpha_i(w)$ the number of symbols that occur in w in cyclic order between the symbols U_i and O_i, taken modulo 2.*

Definition 7. *For an unsigned Gauss word w and a signing s, a signed word w^s is obtained from w by replacing all symbols (U,i) with $(U^{s(i)}, i)$ and (O,i) with $(O^{s(i)}, i)$. Let S_i denote the subset of symbols that occur in w^s in cyclic order between, either the symbols U_i^+ and O_i^+, or O_i^- and U_i^-. Let $\bar{S}_i$ denote $\{U_i^{s(i)}, O_i^{s(i)}\} \cup S_i$ and S_i^{-1} denote the set S_i after swapping Us with Os, that is $S_i^{-1} = \{(U^{s(j)}, j) | (O^{s(j)}, j) \in S_i\} \cup \{(O^{s(j)}, j) | (U^{s(j)}, j) \in S_i\}$. Then $\beta_{ij}(w^s)$ is the number of elements in the intersection of $\bar{S}_i$ and S_j^{-1} taken modulo 2 (i.e. $\beta_{ij}(w^s) = |\bar{S}_i \cap S_j^{-1}| \pmod 2$).*

Notice that $\beta_{ij}(w^s)$ depends on signs $s(i)$ and $s(j)$ but not on $s(k)$ for $k \neq i, j$.

Definition 8. *Given an unsigned Gauss word w, the vertices of the interlacement graph $G(w)$ are labels in a shadow projection $sp(w)$ (natural numbers) and the edges of $G(w)$ are the pairs of labels (i, j) such that i and j are interlaced in $sp(w)$ (i occurs once between two occurrences of j and vice versa).*

Example 1. Given $w = U_1O_3U_4U_2O_1U_5O_2U_3O_5O_4$, the interlacement graph $G(w)$ of w is shown in figure 4. Let $i = 1$, $j = 2$, $s(1) = +$ and $s(2) = +$. Then $\alpha_1(w) = |\{O_3, U_4, U_2\}| = 3 \equiv 1 \pmod 2$, and $\beta_{12}(w^s) = |\bar{S}_1 \cap S_2^{-1}| = |\{U_1^{s(1)}, O_3^{s(3)}, U_4^{s(4)}, U_2^{s(2)}, O_1^{s(1)}\} \cap \{U_1^{s(1)}, O_5^{s(5)}\}| = |\{U_1^{s(1)}\}| \equiv 1 \pmod 2$.

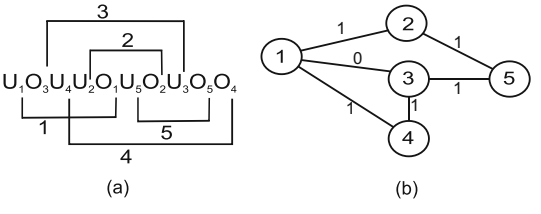


Fig. 3. Non-planar Gauss word w and its corresponding interlacement graph $G(w)$ with edges (i, j) labelled by $\beta_{ij}(w^s)$

For a signed word w^s and for each edge e_{ij} in $G(w)$, we assign the number $\beta_{ij}(w^s) \in \mathbb{Z}_2$. According to [4] that assignment defines a $\mathbb{Z}_2$ 1-cochain $B(w^s)$ and the property that the Cairns-Elton algorithm checks is whether this cochain is closed. For the purpose of this paper we need only characterisation of the closedness of $B(w^s)$ in terms of notions we have already introduced: $B(w^s)$ is closed if and only if for every closed path P in $G(w)$, the sum of the numbers $\beta_{ij}(w^s) \equiv 0 \pmod 2$ for each edge $(ij) \in P$. The closed path is in fact a simple cycle with no repeated vertices other than starting and ending vertices.
The Propositions 1 and 2 provide with the properties crucial for the efficient implementation of the Cairns-Elton algorithm. Also as an easy consequence of Proposition 1, we formaulate Lemma 1.

Proposition 1. *[4, page 139] $\beta_{ij}(w^s)$ does not depend on s whenever i and j do not interlace.*

Lemma 1. *Let $G(w)$ denote the interlacement graph of the Gauss word w and $N(v_i)$ denote the set of vertices connected to v_i, then $\beta_{ij}(w) = |N(v_i) \cap N(v_j)| \pmod 2$ whenever i and j do not interlace in w.*

Proposition 2. *[4, Lemma 1] The condition $B(w^s)$ to be closed depends on w but not on s.*

For *all positive* signing s, that is $s(i) = +$ for all $i \in \mathbb{N}$, we denote $B(w^s)$ by $B(w)$ and $\beta_{ij}(w^s)$ by $\beta_{ij}(w)$.

Cairns-Elton Algorithm: Given an unsigned Gauss word w, the algorithm proceeds by checking that
1. For all $i \in w$, $\alpha_i(w) = 0$.
2. For all $i, j \in w$, $\beta_{i,j}(w) = 0$ whenever i and j do not interlace.
3. $B(w)$ is closed.

If conditions 1,2 and 3 are satisfied then the algorithm returns "the word w is planar", otherwise if any of the conditions is not satisfied, the algorithm returns "the word w is non-planar".

3.2 Implementation of Cairns-Elton Algorithm

We will show in this subsection that the checking of first two conditions of the above algorithm is implementable by a deterministic register automata whereas the checking of the third condition is implementable by a co-non-deterministic register automata.

First, we refine the above description of the algorithm and present it in more details. Given an unsigned Gauss word w, the algorithm proceeds in three stages.

1. The input word is checked on whether the number of neighbours of v_i is odd for some v_i in $G(w)$. If "yes", the algorithm stops with the result "the input word is non-planar", otherwise the algorithm proceeds to the second stage.
2. The input word is checked on whether $\beta_{ij}(w)$ is odd for some pair of vertices (v_i, v_j) in $G(w)$ that are not connected by an edge. If "yes", the algorithm stops with the result "the input word is non-planar", otherwise the algorithm proceeds to the third stage.
3. For all positive signing s, the input word is checked on whether there exists a cycle in $G(w)$ such that the sum of $\beta_{ij}(w^s)$ assigned to its edges e_{ij} is odd. If "yes", the algorithm stops with the result "the input word is non-planar", otherwise the algorithm stops with the result "the input word is planar".

Theorem 1. *The language of non-planar unsigned Gauss words (UNSIGNED NONPLANARITY) can be recognised by a two-way non-deterministic register automaton.*

Proof. The proof is divided into two parts. In the first part we show that the first two conditions can be implemented by a deterministic register automata and in the second part we show the third condition can be implemented by a non-deterministic register automata. Let w be an unsigned Gauss word and $G(w)$ be the interlacement graph of w. Denote by $N(v_i)$ the set of all neighbours of a vertex $v_i \in G(w)$.

Part 1. For the first condition, the automaton will check the number $|N(v_i)|$ of neighbours of each vertex $v_i \in G(w)$. It will store in the register the first occurrence of the label i in w which corresponds to vertex v_i and store the parity of $|N(v_i)|$ in finite state control of the automaton. To check the parity of $|N(v_i)|$, the automaton goes through each symbol j in between the pair of the labels i and i^{-1} in w (where i^{-1} represents the second occurrence of i in w) and on the first occurrence of j it moves first to an odd state and then alternates between odd and even states for any further occurrences. If i^{-1} is reached and the current state is odd then it moves to an accepting state. Otherwise if, for all vertices $v_i \in G(w)$, the parity of $|N(v_i)|$ is even then it checks condition (2).

For the second condition, we use Lemma 1 where the automaton is required to check the number of common neighbours ($|N(v_i) \cap N(v_j)|$) for any pair of vertices (v_i, v_j) that are not connected by an edge in $G(w)$. To verify that a vertex v_i is not connected to any vertex v_j, the automaton stores in the registers the first occurrence of i and the first occurrence of each j and then it checks whether there is an even number (either 2 or 0) of occurrences of each j in between i and i^{-1}. If the number of occurrences of j in between i and i^{-1} is even then it stores the symbol k in the register (which occurs in between i and i^{-1}) and compares it with the symbols in between j and j^{-1}. The parity of $|N(v_i) \cap N(v_j)|$ is stored in finite state control of the automaton. If there is a match, it will move first to an odd state and then alternate between odd and even states for any further matches. If i^{-1} is reached and current state is odd, the automaton moves to an accepting state. Otherwise if, for all pairs of vertices $v_i, v_j \in G(w)$ that are not connected by an edge, the parity of $|N(v_i) \cap N(v_j)|$ is even then it checks condition 3.

Part 2. For checking Condition 3, we assume that s is all positive signing. First, we show how to check if two vertices v_i and v_j are connected by an edge and how to compute their $\beta_{ij}(w^s)$ value. Then we show how to sum up the $\beta_{ij}(w^s)$ values online during the traversal of a cycle. To verify that two vertices v_i and v_j are connected by an edge of $G(w)$, the automaton keeps a copy of i and j in the registers and checks that there is only one occurrence of j in between U_i and O_i. Now to calculate the value of $\beta_{ij}(w^s)$ for all positive signing s, the automaton moves its head to find the symbol U_j then compares the counterpart of each symbol k in between U_j and O_j (notice that all such counterparts form the set S_j^{-1}) with the symbols in the set $\bar{S}_i(U_i, \ldots, O_i)$. If there is a match it will move first to an odd state and then alternate between odd and even states for any further matches until O_j is reached. Finally to traverse a cycle in $G(w)$, the automaton non-deterministically chooses a vertex v_i and moves along chosen edge. During the traversal, it sums up the $\beta_{ij}(w^s)$ values of each visited edge by incrementing the counter by $1 \pmod 2$ only if the value of $\beta_{ij}(w^s)$ is odd, and continue updating the counter until the same vertex v_i is met for the second time. If v_i is met for the second time and the value of the counter is odd then the automaton moves to an accepting state. □

Corollary 1. *UNSIGNED PLANARITY can be recognised by two-way co-non-deterministic register automata.*

3.3 RA to LOGSPACE

We will show that if a language L over the infinite alphabet D is acceptable by two-way register automata then the encoding of L over a finite alphabet can be accepted by a Turing machine in log space memory. Let us first define the encoding of L over a finite alphabet as follows.

Let L be a language over the infinite alphabet D. For any symbol y of a word $w \in L$ we denote by $ord_w(y)$ (or $ord(y)$ if w is understood) the number of distinct symbols in w to the left of the first occurrence of y in w. So, for example, if $w = aacbca$ then $ord_w(a) = 0$, $ord_w(c) = 1$ and $ord_w(b) = 2$.

If ϕ is a mapping from natural numbers into their binary encoding, $\phi : \mathbb{N} \to \{0,1\}^*$, then for any word $w = w_1, ..., w_k \in L$ where $w_i \in D$ the mapping $\psi(w) : D^* \to \{0,1,\#\}^*$ is an encoding of a word w, where each binary encoding of w_i is separated by a special symbol $\#$:

$$\psi(w) = \#\phi(ord(w_1))\#\phi(ord(w_2))\#, ..., \#\phi(ord(w_k)).$$

Thus a language $L_{finite} = \{\psi(w) | w \in L\}$ is an encoding of L over the finite alphabet $\{0,1,\#\}$ and L_{binary} is a natural encoding of L_{finite} over $\{0,1\}$ alphabet, where $0 \to 00$, $1 \to 01$ and $\# \to 11$.

Proposition 3. *If L is recognisable by a finite register automata then L_{finite} is recognisable by a Turing machine in* log *space memory.*

Proof. Let L denote the language accepted by a finite register automaton A with r registers. Let us show that the language L_{finite} is recognisable by a Turing machine M that uses at most $O(\log n)$ space. Let $w = w_1, ..., w_n \in L$ and $|w| = n$, then w consists of no more than n different symbols, so the length of the binary encoding of w_i is no more than $\log n$, i.e. $|\phi(ord(w_i))| \leq \log n$. So M can mimic all the computations of A by keeping the value of the registers of A on its work tape. The finite state control in A will correspond to the finite state control in M (including non-determinism) and the content of r registers in A will be stored on the work tape in M, which require $r \log n$ cells and r is a constant. The only two operations of register automata are: to store a symbol in a register and to compare the register value with a symbol on a tape will not require any extra space apart from $r \log n$ cells. □

Corollary 2. *UNSIGNED PLANARITY is in* **co-**$\mathcal{NL}$ *and therefore is in* $\mathcal{NL}$.

Next we refine the complexity bounds of planarity recognition by 1) demonstrating general result on simulation of Logspace bounded computations on register automata and 2) showing that UNSIGNED PLANARITY is in $\mathcal{L}$.

3.4 UNSIGNED PLANARITY in $\mathcal{L}$

We have shown in subsection 3.2 that planarity of Gauss words is recognisable by co-nondeterministic register automata and therefore belongs to the complexity class **co-**$\mathcal{NL}$ ($= \mathcal{NL}$). Using simple arguments one can show that in terms of classical complexity classes the result can be refined further, that is UNSIGNED

PLANARITY belongs to $\mathcal{L}$ (deterministic logspace). Indeed, the only place when one needs nondeterminism in the proof of Theorem 1 is in PART 2 where the search for the cycles in the interlacement graph bearing odd sum of labels. It is routine to check that the rest of the algorithm can be easily implemented in $\mathcal{L}$. It follows then that UNSIGNED NONPLANARITY is logspace reducible to the problem from the following proposition and therefore is in $\mathcal{L}$.

Proposition 4. *The following problem is in $\mathcal{L}$ (deterministic logspace).*

GIVEN: *An undirected graph G with every edge labelled by* 0 *or* 1
QUESTION: *Is there any cycle C in G that the sum of labels of all edges in C is odd?*

Proof. The search for the required cycle can be be done nondeterministically in logspace by guessing next edge in the cycle and computing the parity of the sum of labels online. Since the graph is undirected, the search can be implemented in symmetric logspace [14]. By [23] $\mathcal{SL} = \mathcal{L}$. Since $\mathcal{L}$ is closed under complementation, it follows that UNSIGNED PLANARITY is in $\mathcal{L}$. □

3.5 LOGSPACE to DRA

For a word w we define its variability $v(w)$ as a number of distinct symbols in w. For a language L and an integer function $f(n)$ we say that variability of L is of the order $f(n)$ iff $min_{w\in L,|w|=n}v(w) \geq f(n)$, i.e. $f(n)$ is a lower bound of variabilities of words of length n in L.

Lemma 2. *Computations of a Turing Machine on a work tape T of size $c{\cdot}log(k)$ over a binary alphabet can be simulated by Register Automata model on an input string S, where $v(S)=k$.*

Proof. Assume that a head H is on a work tape T at the position $T(i)$. First all operations under the tape head such as rewriting, checking current symbol on a tape and head moves can be simulated by operations on two pushdown stacks, where first stack keeps the front part of T: $T(0)\dots T(i)$ and the second part of T, is stored in the second pushdown stack with $T(i{+}1)$ on the top [19]. Also it is easy to see that a binary word on a stack can be represented by an integer x, where empty stack corresponds to 1 value, push a 0 onto the top of a stack corresponds to $x \mapsto 2x$ and pushing a 1 corresponds to $x \mapsto 2x + 1$; checking the top symbol can be done by checking divisibility by 2 and popping of a 1 or 0 can be done by operations $x \mapsto (x - 1)/2$ or $x \mapsto (x - 1)/2$ respectively. Further we notice that the register automata on an input with k distinct symbols can simulate any finite number of counters of size k^c for any constant c with the following main operations: increment, decrement, multiplication by 2, division by 2 and zero testing. Storing a symbol x in a register represents a value $ord(x) + 1$ of a counter. The implementation of all required operations is straightforward albeit tedious. The operation of increment (decrement) by one can be implemented by moving forward to the first occurrence of the next (previous) distinct symbol on an input string [15]. Testing of a counter to be equal to 1 corresponds to checking whether a stored symbol appear as a first symbol of an input. Also it is well known that operation of multiplication (or division) by 2 and divisibility by

2 can be implemented by a finite number of extra counters with increment and decrement operations and zero testing (or testing to be equal to 1) [19]. Thus, all operations for integer representations of pushdown stacks can be implemented by a register automaton. In this case since the virtual counters can store a value of size k^c and simulate two pushdown stacks of size $log(k^c)$ we have that the length of a work tape T which is simulated by register automaton on an input with k distinct symbols is bounded by $log(k^c)$. □

Theorem 2. *Given a language L with a variability of the order $f(n)$, if a finite projection L_{binary} of L belongs to SPACE(log(f(n))) then L can be recognised by a register automata.*

Proof. The binary code of each symbol in $w \in L$ is of a logarithmic size from the number of distinct symbols in w. So for any word of size n over an infinite alphabet, the length of the binary code for each symbol is no more than $log(n)$. Since the number of distinct symbols in any word $w \in L$ is at least $f(n)$, thus by Lemma 2 we can simulate virtual tape over a binary alphabet of a length $c \cdot log(f(n))$ for any integer constant c. Therefore register automaton can take any symbol y from an infinite alphabet on an input string and convert it into a finite binary representation in the virtual work tape over a binary alphabet, by counting the number of previously appeared distinct symbols from the beginning of an input string and storing it on a virtual tape, i.e. converting y into a binary representation of $ord(y)$. Also by the same Lemma 2 we have that any extra memory of size $O(log(f(n))$ can be implemented by a finite number of registers on a language L with a variability of the order $f(n)$. So any computations over language L_{binary} that requires SPACE($log(f(n))$) can be implemented by a register automaton on a language L with a variability of the order $f(n)$. □

Corollary 3. *UNSIGNED PLANARITY is in DRA.*

The above corollary follows from Proposition 4 and Theorem 2.

4 Conclusion

In this paper we have investigated the complexity of planarity of knot diagrams represented by Gauss words. We have shown that planarity of unsigned Gauss words can be recognised in deterministic logarithmic space on classical computational models and by deterministic register automata over infinite alphabets. To demonstrate these results we have used generic mutual simulation between both computations models for the languages of bounded variability. Notice that unlike the case of signed Gauss words [15] we do not provide explicit deterministic LOGSPACE bounded decision procedure for planarity of unsigned Gauss words and rather refer to the general reduction of $\mathcal{SL}$ to $\mathcal{L}$ [23]. An explicit deterministic LOGSPACE algorithm for the later case as well as the comparison of its complexity with the algorithm(s) for signed case is a topic for further work.

References

1. Abramsky, S.: Temperley-Lieb Algebra: From Knot Theory to Logic and Computation via Quantum Mechanics. Mathematics of Quantum Computation and Quantum Technology (2007)

2. Björklund, H., Schwentick, T.: On Notions of Regularity for Data Languages. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 88–99. Springer, Heidelberg (2007)
3. Cairns, G., Elton, D.: The planarity problem for signed Gauss words. Journal of Knot Theory and its Ramifications 2(4), 359–367 (1993)
4. Cairns, G., Elton, D.: The Planarity Problem II. Journal of Knot Theory and its Ramifications 5, 137–144 (1996)
5. De Fraysseix, H., Ossona de Mendez, P.: On a characterization of Gauss codes. Discrete and Computational Geometry 22(2), 287–295 (1999)
6. Dehn, M.: Uber kombinatorische topologie. Acta Math. 67, 123–168 (1936)
7. Freivalds, R.: Knot Theory, Jones Polynomial and Quantum Computing. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 15–25. Springer, Heidelberg (2005)
8. Gauss, C.: Werke. Band 8. Teubner (1900)
9. Hass, J., Lagarias, J., Pippenger, N.: The computational complexity of knot and link problems. Journal of the ACM (JACM) 46(2), 185–211 (1999)
10. Kaminski, M., Francez, N.: Finite-memory automata. Theoretical Computer Science 134(2), 329–363 (1994)
11. Kari, J., Niemi, V.: Morphic Images of Gauss Codes. In: Developments in Language Theory, pp. 144–156 (1993)
12. Kauffman, L.: Virtual knot theory. Arxiv Preprint Math. GT/ 9811028 (1998)
13. Kurlin, V.: Gauss paragraphs of classical links and a characterization of virtual link groups. Mathematical Proceedings Cambridge Phil. Soc. 145, 129–140 (2008)
14. Lewis, H., Papadimitriou, C.: Symmertric space-bounded computation (extended abstract). In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 374–384. Springer, Heidelberg (1980)
15. Lisitsa, A., Potapov, I., Saleh, R.: Automata on Gauss Words. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 505–517. Springer, Heidelberg (2009)
16. Lomonaco Jr., S., Kauffman, L.: Topological Quantum Computing and the Jones Polynomial. Arxiv Preprint Quant-ph/ 0605004 (2006)
17. Lovász, L., Marx, M.: A forbidden substructure characterization of Gauss codes. Bull. Amer. Math. Soc. 82(1) (1976)
18. Manturov, V.: A proof of Vassiliev's conjecture on the planarity of singular links. Izvestiya: Mathematics 69(5), 1025–1033 (2005)
19. Minsky, M.: Computation: finite and infinite machines (1967)
20. Nagy, J.: Über ein topologisches Problem von Gauss. Mathematische Zeitschrift 26(1), 579–592 (1927)
21. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Transactions on Computational Logic 5(3), 403–435 (2004)
22. Read, R., Rosenstiehl, P.: On the Gauss crossing problem. Colloq. Math. Soc. Janos Bolyai. 18, 843–876 (1976)
23. Reingold, O.: Undirected connectivity in log-space. Journal of the ACM (JACM) 55(4), 17 (2008)
24. Rosenstiehl, P.: Solution algebrique du probleme de Gauss sur la permutation des points d'intersection d'une ou plusieurs courbes fermees du plan. CR Acad. Sci. Paris Ser. AB 283 (1976)
25. Rosenstiehl, P., Tarjan, R.: Gauss codes, planar Hamiltonian graphs, and stack-sortable permutations. Journal of Algorithms 5(3), 375–390 (1984)
26. Shtylla, B., Traldi, L., Zulli, L.: On the realization of double occurrence words. Discrete Mathematics 309(6), 1769–1773 (2009)

Tarski's Principle, Categorial Grammars and Learnability

Jacek Marciniec

Faculty of Mathematics and Computer Science,
Adam Mickiewicz University,
Poznań, Poland
jacek.marciniec@amu.edu.pl

Abstract. In this paper, functorial languages with the following characteristic are investigated: if two functor-argument structures occur in at last one common functorial context, then they are intersubstitutable on arguments' positions in all elements (sentences) of a language. We prove learnability of the class of all such languages (in the model of Gold). Since our class has infinite elasticity, we could not employ a widely used method of learnability proving. Instead, we adopted Buszkowski's discovery procedure, based on unification.

Keywords: categorial grammar, grammatical inference, learning, positive data, unification.

1 Introduction

Formal learning theory is an important branch of the theory of inductive inference. Its mathematical shape has been proposed by Solomonoff [17,18] and Gold [11]. Rudimentary Gold's results concerning identification of languages from positive data seemed to be of no perspectives at first. Yet, Angluin's [1,2,3] contribution made a breakthrough — non-trivial learnable classes were presented. Many examples of learnable classes of different sorts were described so far. The classes are most often defined through some restrictions, mainly numerical imposed on the shape of the grammars (or automata). Language theoretic properties of the languages under considerations are usually neglected.

Angluin's reversible languages [3], were described both ways — through automaton properties and language theoretic. The latter conforms with so-called Tarski's principle (see [19]) — expressions mutually substitutable in at least one sentential context are mutually substitutable in all sentential contexts. Analogous condition, this time concerning context-free languages, one can find in [10].

According to Buszkowski [5,8], Tarski's principle also holds for rigid functorial languages. Since it applies also to some non-rigid ones (as for example $\{(a,c)_1,(b,c)_1,(d,a)_2,(d,b)_2\}$), a natural question arises whether it implies learnability of functorial languages (recall that learnability of rigid languages was established by Kanazawa [12,13]). The question above is still open, but we

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 378–389, 2011.

can show that, when restricted to argument substructures only, Tarski's principle constitutes a sufficient condition for learnability.

The class of functorial languages we present in this paper is learnable despite infinite elasticity. The elements (languages) from the class have been characterized in a purely language theoretic way — each two structures, occurring in at least one common functorial context, are intersubstitutable in all argument positions. Following Kanazawa [12,13] we make substantial use of the method involving unification, introduced by Buszkowski [6], and further developed by Buszkowski and Penn [9]. Also we employ the properties of infinite and unifiable sets of types, established in [15,16].

2 Preliminaries

2.1 Functor-Argument Structures

The set $\mathsf{FS}(\mathscr{A})$ of all *functor-argument structures* over a set $\mathscr{A}$ of *atoms* is defined as the smallest set fulfilling conditions: $\mathscr{A} \subseteq \mathsf{FS}(\mathscr{A})$, if $A_j \in \mathsf{FS}(\mathscr{A})$ for $j = 1, \ldots, n$, then $(A_1, \ldots, A_n)_i \in \mathsf{FS}(\mathscr{A})$. A_i is the *functor* of $(A_1, \ldots, A_n)_i$, whereas each A_j, for $j \neq i$ is its *argument*. We set $\mathsf{width}((A_1, \ldots, A_n)_i) = n$.

The *signature* ρ — a function that assigns a sequence of pairs of natural numbers to each functor-argument structure, is defined inductively as follows: if $a \in \mathscr{A}$ then $\rho(a) = \lambda$ (empty string), $\rho((A_1, \ldots, A_n)_i) = \rho(A_i) \cdot \langle n, i \rangle$, where $\cdot$ denotes concatenation.

For any $T \subseteq \mathsf{FS}(\mathscr{A})$ we define the set $\mathsf{SUB}(T)$ of all its *substructures*, as the smallest set fulfilling: $T \subseteq \mathsf{SUB}(T)$, if $(A_1, \ldots, A_n)_i \in \mathsf{SUB}(T)$, then $A_j \in \mathsf{SUB}(T)$, for $j = 1, \ldots, n$.

By $\mathsf{SUB}_\mathsf{a}(T)$ we will denote the set of all *argument substructures* of the elements from T:

$$B \in \mathsf{SUB}_\mathsf{a}(T) \leftrightarrow (\exists (A_1, \ldots, A_n)_i \in \mathsf{SUB}(T))(\exists j \in \{1, \ldots, n\} \setminus \{i\})(B = A_j).$$

The *functorial path* of a structure $A \in \mathsf{FS}(\mathscr{A})$, is the maximal sequence $F_1, \ldots, F_k$ of substructures of A, such that F_1 is the functor of A, and for all $i = 2, \ldots, k$, F_i is the functor of F_{i-1}. Then F_k is an atom and it will be denoted by $\uparrow_f(A)$. The length of the functorial path of a structure A will be denoted by $\mathsf{height}(A)$ (for $a \in \mathscr{A}$ we set $\mathsf{height}(a) = 0$).

2.2 Substitutions

Suppose that $\mathscr{A} = \mathscr{C} \cup \mathscr{V}$, where $\mathscr{C}$ denotes a set of *constants*, $\mathscr{V}$ *variables* and $\mathscr{C} \cap \mathscr{V} = \emptyset$. A function $\Theta : \mathsf{FS}(\mathscr{A}) \mapsto \mathsf{FS}(\mathscr{A})$ is a *substitution* over $\mathscr{A}$, if $\Theta(c) = c$, for each $c \in \mathscr{C}$, and $\Theta((A_1, \ldots, A_n)_i) = (\Theta(A_1), \ldots, \Theta(A_n))_i$, for every $(A_1, \ldots, A_n)_i \in \mathsf{FS}(\mathscr{A})$.

By $\circ$ we denote composition of substitutions: $\Theta = \Theta_1 \circ \Theta_2$ iff $\Theta(A) = \Theta_1(\Theta_2(A))$. If $T \subseteq \mathsf{FS}(\mathscr{A})$, then $\Theta[T]$ denotes the image of T.

A substitution Θ is *variable-pure*, if for each $X \in \mathscr{V}$ also $\Theta(X) \in \mathscr{V}$.

2.3 Functorial Languages

Let us fix a finite *vocabulary* (or *lexicon*) V. By a *functorial language* over V, we mean any subset of $\mathsf{FS}(V)$.

3 Functorial Contexts

We fix the set $\mathscr{X} = \{\mathbb{X}_{(i,j)} : i, j \in \mathbb{N}\}$ of special purpose variables (for technical reasons indexed with pairs of natural numbers). The elements of $\mathscr{X}$ will play the role of *argument placeholders*. We also assume that $V \cap \mathscr{X} = \emptyset$.

Definition 1. *A* functorial context *(f-context) is a functor argument structure over $V \cup \mathscr{X}$, defined inductively as follows:*

- *each $v \in V$ is an f-context of* $\mathsf{height}(v) = 0$,
- *if C is an f-context and* $\mathsf{height}(C) = k - 1$, *then*

$$(\mathbb{X}_{(k,1)}, \ldots, \mathbb{X}_{(k,i-1)}, C, \mathbb{X}_{(k,i+1)}, \ldots, \mathbb{X}_{(k,n)})_i,$$

for $n > 1, 0 < i \leqslant n, k > 0$ is an f-context of height *equal to k.*

Definition 2. *Let $A \in \mathsf{FS}(V)$. We define* $\mathsf{f\text{-}cont}(A)$ *— the functorial context determined by A:*

- *if $v \in V$, then* $\mathsf{f\text{-}cont}(v) = v$, $\mathsf{height}(v) = 0$,
- *if $A = (A_1, \ldots, A_n)_i$, then*

$$\mathsf{f\text{-}cont}(A) = (\mathbb{X}_{(k,1)}, \ldots, \mathbb{X}_{(k,i-1)}, \mathsf{f\text{-}cont}(A_i), \mathbb{X}_{(k,i+1)}, \ldots, \mathbb{X}_{(k,n)})_i,$$

where $k = \mathsf{height}(A) = \mathsf{height}(A_i) + 1$.

For a language $\mathbb{L} \subseteq \mathsf{FS}(V)$, we define:

$$\mathsf{F\text{-}CONT}_a(\mathbb{L}) = \{\mathsf{f\text{-}cont}(A) : A \in \mathsf{SUB_a}(\mathbb{L})\},$$

$$\mathsf{F\text{-}CONT}_L(\mathbb{L}) = \{\mathsf{f\text{-}cont}(A) : A \in \mathbb{L}\}.$$

$$\mathsf{F\text{-}CONT}(\mathbb{L}) = \mathsf{F\text{-}CONT}_L(\mathbb{L}) \cup \mathsf{F\text{-}CONT}_a(\mathbb{L})$$

Let $C = \mathsf{f\text{-}cont}(A)$. There is exactly one substitution Φ over $V \cup \mathscr{X}$, involving all and only the variables from $\mathscr{X}$, occurring in C, such that $\Phi(C) = A$. We will denote $[A]_{(i,j)} = \Phi(\mathbb{X}_{(i,j)})$, for each such a variable $\mathbb{X}_{(i,j)}$. If a variable $\mathbb{X}_{(i,j)}$ does not occur in the f-context of A, we assume that $[A]_{(i,j)}$ is unspecified.

In a way (i, j) may be treated as the coordinates of the argument substructure $[A]_{(i,j)}$ within the structure A.

In what follows we will assume that whenever used, both i and j are restricted to such values that $[A]_{(i,j)}$ is specified.

Fact 3. *For any $A, B \in \mathsf{FS}(V)$:*

$$\mathsf{f\text{-}cont}(A) = \mathsf{f\text{-}cont}(B) \quad \leftrightarrow \quad \uparrow_f(A) = \uparrow_f(B) \ \wedge \ \rho(A) = \rho(B).$$

Definition 4. *Let* $\mathbb{L} \subseteq \mathsf{FS}(V)$. *We define the relation* $\sim_{\mathbb{L}} \subseteq [\mathsf{SUB_a}(\mathbb{L})]^2$*:* $B_1 \sim_{\mathbb{L}} B_2$ *iff there are* A_1, A_2 *such that* $\{A_1, A_2\} \subseteq \mathbb{L}$ *or* $\{A_1, A_2\} \subseteq \mathsf{SUB_a}(\mathbb{L})$ *and* $i, j \in \mathbb{N}$, *such that* $\mathsf{f\text{-}cont}(A_1) = \mathsf{f\text{-}cont}(A_2)$ *and* $B_1 = [A_1]_{(i,j)}$, $B_2 = [A_2]_{(i,j)}$.

It is clear that the relation $\sim_{\mathbb{L}}$ is reflexive and symmetric, but not necessarily transitive. The sense of the relation is the following: two structures are related if they appear at least once (on corresponding positions) in the same functorial context.

Definition 5. *We define* the argument intersubstitutability *relation* $\approx_{\mathbb{L}} \subseteq [\mathsf{FS}(V)]^2$: $B_1 \approx_{\mathbb{L}} B_2$ *iff* B_1 *and* B_2 *are substitutable in all argument positions of the elements of* $\mathbb{L}$. *More precisely: Let* $A[B_1 := B_2]$ *denotes the outcome of the replacement of some argument occurrences of* B_1 *with* B_2. *Then:*

$$B_1 \approx_{\mathbb{L}} B_2 \text{ iff } (\forall A \in \mathsf{FS}(V))(A \in \mathbb{L} \leftrightarrow A[B_1 := B_2] \in \mathbb{L}).$$

Fact 6. *For any* $\mathbb{L} \subseteq \mathsf{FS}(V)$, $\approx_{\mathbb{L}}$ *is an equivalence relation on* $\mathsf{FS}(V)$.

Lemma 7. *Let* $B_1, B_2 \in \mathsf{FS}(V)$. *Then:*

$$\mathsf{f\text{-}cont}(B_1) = \mathsf{f\text{-}cont}(B_2) \rightarrow \left[(\forall i, j)\left([B_1]_{(i,j)} \approx_{\mathbb{L}} [B_2]_{(i,j)}\right) \rightarrow B_1 \approx_{\mathbb{L}} B_2\right].$$

Proof. Let the sequence $\langle i_1, j_1\rangle, \ldots, \langle i_n, j_n\rangle$ consists of all and only pairs such that $[B_1]_{(i_k,j_k)}$ is specified. (of course $[B_1]_{(i_k,j_k)}$ is specified, whenever $[B_2]_{(i_k,j_k)}$ is specified). Let $C_0, C_1, \ldots, C_k$ be such that $C_0 = B_1$ and $C_k = C_{k-1}\left[[B_1]_{(i_k,j_k)} := [B_2]_{(i_k,j_k)}\right]$ and $C_k = B_2$. For any $A \in \mathsf{FS}(V)$ and $k \in \{1, \ldots, n\}$ we have $A[C_{k-1} := C_k] = A\left[[B_1]_{(i_k,j_k)} := [B_2]_{(i_k,j_k)}\right]$, which implies $C_{k-1} \approx_{\mathbb{L}} C_k$. Our thesis is thus a consequence of transitivity of $\approx_{\mathbb{L}}$. □

Quite similarly one can prove the following:

Fact 8. *Let* $B_1 \in \mathbb{L}$, $B_2 \in \mathsf{FS}(V)$. *Then:*

$$\mathsf{f\text{-}cont}(B_1) = \mathsf{f\text{-}cont}(B_2) \rightarrow \left[(\forall i, j)([B_1]_{(i,j)} \approx_{\mathbb{L}} [B_2]_{(i,j)}) \rightarrow B_2 \in \mathbb{L}\right].$$

Definition 9. *The symbol* $\approx^a_{\mathbb{L}}$ *will denote the restriction of* $\approx_{\mathbb{L}}$ *to the set* $\mathsf{SUB_a}(\mathbb{L})$.

Fact 10. *For each* $\mathbb{L} \subseteq \mathsf{FS}(V)$ *we have* $\approx^a_{\mathbb{L}} \subseteq \sim_{\mathbb{L}}$.

Definition 11. *A language* $\mathbb{L} \subseteq \mathsf{FS}(V)$ *is said to be* argument-explicit, *if*

$$\sim_{\mathbb{L}} = \approx^a_{\mathbb{L}}.$$

Fact 12. *Let* $\mathbb{L}$ *be argument-explicit,* $A_1 \in \mathbb{L}$ *and* $A_2 \in \mathsf{FS}(V)$. *Then:*

$$\mathsf{f\text{-}cont}(A_1) = \mathsf{f\text{-}cont}(A_2) \rightarrow \left[(\forall i, j)([A_1]_{(i,j)} \approx^a_{\mathbb{L}} [A_2]_{(i,j)}) \leftrightarrow A_2 \in \mathbb{L}\right].$$

Definition 13. *A language* $\mathbb{L} \subseteq \mathsf{FS}(V)$ *is* finitely describable, *if*

- $\max\{\mathsf{height}(C) : C \in \mathsf{F\text{-}CONT}(\mathbb{L})\} < \aleph_0$,
- $\max\{\mathsf{width}(A) : A \in \mathsf{SUB}(\mathbb{L})\} < \aleph_0$.

We aim to prove, that the class of all finitely describable argument-explicit languages is identifiable in the limit.

4 Types and Grammars

Definition 14. *By* $\mathsf{Pr} = \mathsf{Var} \cup \{\mathsf{S}\}$ *we denote the set of* primitive (or atomic) types, *where* Var *is a countable set of variables and* $\mathsf{S} \notin \mathsf{Var}$ *is the only constant primitive type (sentence type). The elements of the set* $\mathsf{Tp} = \mathsf{FS}(\mathsf{Pr})$ *will be called* types.

Definition 15. *A* classical categorial grammar G *is an indexed family* $\{I_G(v)\}_{v \in V}$ *where, for each* $v \in V$, $I_G(v) \subseteq \mathsf{Tp}$. *The function* $I_G : V \mapsto 2^{\mathsf{Tp}}$ *is* the initial type assignment *of a grammar* G.

I_G extends to the terminal type assignment*:* $T_G : \mathsf{FS}(V) \mapsto 2^{\mathsf{Tp}}$*:*

- $T_G(v) = I_G(v)$, *for* $v \in V$,
- $T_G((A_1, \ldots, A_n)_i) =$
  $= \{t \in \mathsf{Tp} : (\exists (t_1, \ldots, t_n)_i \in T_G(A_i))(t = t_i \wedge t_j \in T_G(A_j),\ \text{for } j \neq i)\}$.

$\mathsf{FL}(G) = \{A \in \mathsf{FS}(V) : \mathsf{S} \in T_G(A)\}$ — *the functorial language determined by* G.

A grammar G *is* finite, *if each* $I_G(v)$ *is finite,* rigid, *if each* $I_G(v)$ *has at most one element. In such a case we will write* $t = T_G(A)$ *rather than* $t \in T_G(A)$.

We will write $G_1 \subseteq G_2$ *iff* $(\forall v \in V) I_{G_1}(v) \subseteq I_{G_2}(v)$.

Lemma 16. *Let* G *be a rigid grammar,* $A \in \mathsf{FS}(V)$, $T_G(A) = t$. *Then* $\uparrow_f (T_G(\uparrow_f(A))) = \uparrow_f(t)$. *In particular, if* $T_G(A) = p \in \mathsf{Pr}$, *then* $\uparrow_f(T_G(\uparrow_f(A))) = p$.

Proof. By structural induction. The case $A \in V$ is trivial. Suppose $A = (A_1, \ldots, A_n)_i$ and the thesis holds for A_i. We have $T_G((A_1, \ldots, A_n)_i) = t_i$ for some $(t_1, \ldots, t_n)_i = T_G(A_i)$. Hence $\uparrow_f(T_G(\uparrow_f(A))) = \uparrow_f(T_G(\uparrow_f(A_i)))$ $\overset{\text{by induction}}{=} \uparrow_f((t_1, \ldots, t_n)_i) = \uparrow_f(t_i)$. □

Lemma 17. *Let* G *be a rigid grammar,* $A \in \mathsf{FS}(V)$ *and* $T_G(A) \neq \emptyset$. *Then*

$$\rho\left(T_G\left(\uparrow_f(A)\right)\right) = \rho\left(T_G\left(A\right)\right) \cdot \rho\left(A\right)^R.$$

where the superscript ^{R} *denotes reversal.*

In particular, if $T_G(A) = p \in \mathsf{Pr}$, *then* $\rho\left(T_G\left(\uparrow_f(A)\right)\right) = \rho\left(A\right)^R$.

Proof. By induction along functorial path of a structure A. If $A \in V$ then the thesis is trivially fulfilled. Suppose $A = (A_1, \ldots, A_n)_i$ and the thesis holds for A_i. We have

$$\uparrow_f(A) = \uparrow_f(A_i), \tag{1}$$

$$\rho\left(T_G\left(A_i\right)\right) = \rho\left(T_G\left(A\right)\right) \cdot \langle n, i\rangle, \tag{2}$$

$$\rho(A) = \rho(A_i) \cdot \langle n, i\rangle. \tag{3}$$

Hence:
$\rho\left(T_G\left(\uparrow_f(A)\right)\right) \overset{\text{by (1)}}{=} \rho\left(T_G\left(\uparrow_f(A_i)\right)\right) \overset{\text{by induction}}{=} \rho\left(T_G\left(A_i\right)\right) \cdot \rho\left(A_i\right)^R =$
$\overset{\text{by (2)}}{=} \rho\left(T_G(A)\right) \cdot \langle n, i\rangle \cdot \rho(A_i)^R = \rho\left(T_G(A)\right) \cdot \left(\rho(A_i) \cdot \langle n, i\rangle\right)^R \overset{\text{by (3)}}{=}$
$\rho\left(T_G(A)\right) \cdot \rho(A)^R$. □

Definition 18 ([6,7,9]). *Let $A \in \mathsf{FS}(V)$. We will describe a construction of a grammar $\mathsf{GF}(A)$* — general form *determined by A. Let us denote $\overline{V} = V \times \mathbb{N}$. Any $\langle v, i\rangle \in \overline{V}$ (denoted by v^i) is a copy of an atom $v \in V$. Now choose any $\overline{A} \in \mathsf{FS}(\overline{V})$, such that each atom from $\overline{V}$ occurs in $\overline{A}$ at most once, and $\overline{A}$ becomes A after erasing all superscripts following atoms. By induction, we define the mapping $\mapsto$ from $\mathsf{SUB}(\{\overline{A}\})$ to Tp:*

- *at first we set $\overline{A} \mapsto \mathsf{S}$,*
- *then we assign different variables to all the elements of $\mathsf{SUB_a}(\{\overline{A}\})$.*
- *Finally we 'calculate' types of all functor substructures of $\overline{A}$, following the rule:*
  If $(\overline{A}_1, \ldots, \overline{A}_n)_i \mapsto t$ and $\overline{A}_j \mapsto x_{k_j}$ for each $j \neq i$, then

$$\overline{A}_i \mapsto (x_{k_1}, \ldots, x_{k_{i-1}}, t, x_{k_{i+1}}, \ldots, x_{k_n})_i.$$

We can now define the grammar $\mathsf{GF}(A)$:

$$I_{\mathsf{GF}(A)}(v) = \{t : (\exists i \in \mathbb{N}) v^i \mapsto t\}.$$

For $\mathbb{L} \subseteq \mathsf{FS}(V)$ we set:

$$I_{\mathsf{GF}(\mathbb{L})}(v) = \bigcup_{A \in \mathbb{L}} I_{\mathsf{GF}(A)}(v),$$

assuming that each variable may occur in $\mathsf{GF}(A)$ for at most one $A \in \mathbb{L}$. The grammar $\mathsf{GF}(\mathbb{L})$ will be called the general form *determined by $\mathbb{L}$. We admit the case of $\mathbb{L}$ being infinite.*

It is easy to observe that a general form has the following properties:

Fact 19. *Let $\mathbb{L} \subseteq \mathsf{FS}(V)$, $v \in V$ and $t \in I_{\mathsf{GF}(\mathbb{L})}(v)$. Then:*

- $t_1 \in \mathsf{SUB_a}(t) \;\rightarrow\; t_1 \in \mathsf{Var}$,
- *each variable occurs in t at most once.*

Fact 20 ([9,15]). *For any (even infinite) $\mathbb{L} \subseteq \mathsf{FS}(V)$, we have*

$$\mathsf{FL}(\mathsf{GF}(\mathbb{L})) = \mathbb{L}.$$

5 Unification

From now on we will consider only substitutions over Pr.

Definition 21. *Let φ be a substitution. For a family $\mathscr{T} = \{T_i\}_{i\in I}$ of sets of types, we define $\varphi[\mathscr{T}] = \{\varphi[T_i]\}_{i\in I}$.*
In particular, we denote $\varphi[G] = \{\varphi[I_G(v)]\}_{v\in V}$.

Fact 22 ([9]). *For any grammar G and a substitution φ:*

$$\mathsf{FL}(G) \subseteq \mathsf{FL}(\varphi[G])$$

Definition 23. *Let $\mathscr{T} = \{T_i\}_{i\in I}$ be a family of sets of types. A substitution σ* unifies *a family $\mathscr{T}$, if $\mathsf{card}(\sigma(T_i)) \leqslant 1$, for each $i \in I$.*

A unifier η of $\mathscr{T}$ is called a most general unifier *(mgu) of $\mathscr{T}$ if, for any unifier σ of $\mathscr{T}$, there exists a substitution α, fulfilling $\sigma = \alpha \circ \eta$*

Fact 24 ([16]). *Let $\mathscr{T} = \{T_i\}_{i\in I}$ be finite and unifiable family of sets of types. There exists a family $\mathscr{U} = \{U_i\}_{i\in I}$ of finite sets, such that $U_i \subseteq T_i$ for each $i \in I$, and for any family $\mathscr{V} = \{V_i\}_{i\in I}$, fulfilling $U_i \subseteq V_i \subseteq T_i$ for $i \in I$, there are mgus η of $\mathscr{T}$ and σ of $\mathscr{V}$, such that, for each $i \in I$: $\eta[T_i] = \sigma[V_i]$.*

Definition 25. *Two types t_1 and t_2 are* alphabetic *(or* substitutional*)* variants *($t_1 \bowtie t_2$), if there are two substitutions φ_1 and φ_2 fulfilling: $t_1 = \varphi_2(t_2)$ and $t_2 = \varphi_1(t_1)$.*

Fact 26. *Let $v \in V$, types $t_1, t_2 \in I_{\mathsf{GF}(\mathbb{L})}(v)$ are such $\rho(t_1) = \rho(t_2)$ and the primitives $\uparrow_f(t_1)$ and $\uparrow_f(t_2)$ are both variables or both equal to S. Then $t_1 \bowtie t_2$.*

Fact 27. *$\bowtie$ is an equivalence relation on Tp.*

Definition 28. *Let $\sim$ be any equivalence relation on Tp and $T \subseteq \mathsf{Tp}$. By $T/\!\sim$ we denote the partition of T induced by $\sim$. For a family $\mathscr{T} = \{T_i\}_{i\in I}$ of sets of types, we set*

$$\mathscr{T}/\!\sim = \bigcup_{i\in I} (T_i/\!\sim)\,.$$

Please observe, that $\mathscr{T}/\!\sim$ is also a family of sets of types.

Fact 29. *For any family $\mathscr{T} = \{T_i\}_{i\in I}$ of sets of types, the family $\mathscr{T}/\!\bowtie$ is unifiable (but does not have to be finite).*

Fact 30. *For any family $\mathscr{T} = \{T_i\}_{i\in I}$ of sets of types, an mgu of $\mathscr{T}/\!\bowtie$ is variable-pure.*

Fact 31. *If $\mathbb{L} \subseteq \mathsf{FS}(V)$ is finitely describable, then $\mathsf{GF}(\mathbb{L})/\!\bowtie$ is finite.*

Definition 32. *Let $\mathbb{L} \subseteq \mathsf{FS}(V)$ be finitely describable and let η be an mgu of $\mathsf{GF}(\mathbb{L})/\!\bowtie$. We define the grammar $\mathsf{AEG}(\mathbb{L}) = \eta[\mathsf{GF}(\mathbb{L})]$.*

Lemma 33. *The grammar $\mathsf{AEG}(\mathbb{L})$ has the following properties:*

a) all argument subtypes are variables:

$$(\forall v \in V)(\forall t \in I_{\mathsf{AEG}(\mathbb{L})}(v))(\forall t_1 \in \mathsf{Tp})(t_1 \in \mathsf{SUB_a}(t) \rightarrow t_1 \in \mathsf{Var}),$$

b) for any $v \in V$ and types $t_1, t_2 \in I_{\mathsf{AEG}(\mathbb{L})}(v)$ such that $\rho(t_1) = \rho(t_2)$, if $\uparrow_f (t_1) = \uparrow_f (t_2)$ or $\{\uparrow_f (t_1), \uparrow_f (t_2)\} \subseteq \mathsf{Var}$, then $t_1 = t_2$.

c) structures with the same functorial context get the same variable type:

$$\mathsf{f\text{-}cont}(A) = \mathsf{f\text{-}cont}(B) \wedge T_{\mathsf{AEG}(\mathbb{L})}(A) \cap \mathsf{Var} \neq \emptyset \wedge T_{\mathsf{AEG}(\mathbb{L})}(B) \cap \mathsf{Var} \neq \emptyset \rightarrow \\ \rightarrow T_{\mathsf{AEG}(\mathbb{L})}(A) \cap \mathsf{Var} = T_{\mathsf{AEG}(\mathbb{L})}(B) \cap \mathsf{Var},$$

d) each structure gets at most one variable type:

$$\{x, y\} \subseteq T_{\mathsf{AEG}(\mathbb{L})}(A) \quad \rightarrow \quad x = y,$$

Proof. a) The property is inherited from the grammar $\mathsf{GF}(\mathbb{L})$ (see Fact 19), since variable-pure substitution does not affect it.

b) There are $u_1, u_2 \in I_{\mathsf{GF}(\mathbb{L})}$ such that $t_1 = \eta(u_1)$ and $t_2 = \eta(u_2)$, where $\mathsf{AEG}(\mathbb{L}) = \eta[\mathsf{GF}(\mathbb{L})]$. Since variable-pure substitution does not change the signature, we have $\rho(u_1) = \rho(u_2)$ and hence, by Fact 26, $u_1 \bowtie u_1$, which implies $t_1 = t_2$.

c) Let $x \in T_{\mathsf{AEG}(\mathbb{L})}(A)$ and $y \in T_{\mathsf{AEG}(\mathbb{L})}(B)$. By Fact 3, $v = \uparrow_f (A) = \uparrow_f (B)$. By Lemma 16 there are types $t_1, t_2 \in I_{\mathsf{AEG}(\mathbb{L})}(v)$ such that $\uparrow_f (t_1) = x$ and $\uparrow_f (t_2) = y$. By Fact 3 and Lemma 17 $\rho(t_1) = \rho(t_2)$. From b) we get $t_1 = t_2$ and $x = y$ accordingly.

d) It suffices to replace B with A in the proof of c). □

Definition 34. *Let $\mathbb{L} \subseteq \mathsf{FS}(V)$ be finitely describable. We denote:*

$$\mathsf{AEL}(\mathbb{L}) = \mathsf{FL}(\mathsf{AEG}(\mathbb{L})).$$

From Fact 20 and Fact 22, one gets immediately:

Fact 35. $\mathbb{L} \subseteq \mathsf{AEL}(\mathbb{L})$.

For a finitely describable $\mathbb{L}$ the grammar $\mathsf{AEG}(\mathbb{L})$ is finite by, Fact 31, so the language $\mathsf{AEL}(\mathbb{L})$ is also finitely describable. It is now easy to observe the following characteristics of finitely describable languages:

Fact 36. *A language $\mathbb{L} \subseteq \mathsf{FS}(V)$ is finitely describable, if and only if $\mathbb{L} \subseteq \mathsf{FL}(G)$ for some finite grammar G.*

Lemma 37. *Let $p \in \mathsf{Pr}$, $A_1, A_2 \in \mathsf{SUB}(\mathsf{AEL}(\mathbb{L}))$.*
If $p \in T_{\mathsf{AEL}(\mathbb{L})}(A_1) \cap T_{\mathsf{AEL}(\mathbb{L})}(A_2)$ and $\mathsf{f\text{-}cont}(A_1) = \mathsf{f\text{-}cont}(A_2)$, then

$$(\forall i, j)(\exists x \in \mathsf{Var}) T_{\mathsf{AEG}(\mathbb{L})}\left([A_1]_{(i,j)}\right) \cap T_{\mathsf{AEG}(\mathbb{L})}\left([A_2]_{(i,j)}\right) \cap \mathsf{Var} = \{x\}.$$

Proof. Let $v = \uparrow_f (A_1) = \uparrow_f (A_2)$, $\alpha = \rho(A_1) = \rho(A_2)$. There is exactly one type $t \in I_{\mathsf{AEG}(\mathbb{L})}(v)$ such that $\rho(t) = \alpha^R$ and $\uparrow_f (t) = p$. Since the type of a functor determines the types of its arguments, our thesis holds. □

Theorem 38. *For any finitely describable* $\mathbb{L} \subseteq \mathsf{FS}(V)$*, the language* $\mathsf{AEL}(\mathbb{L})$ *is argument-explicit.*

Proof. Assume $B_1 \sim_{\mathsf{AEL}(\mathbb{L})} B_2$. There exist A_1 and A_2 such that

$$\{A_1, A_2\} \subseteq \mathsf{SUB_a}(\mathsf{AEL}(\mathbb{L})) \text{ or } \{A_1, A_2\} \subseteq \mathsf{AEL}(\mathbb{L}),$$

$\mathsf{f\text{-}cont}(A_1) = \mathsf{f\text{-}cont}(A_2)$ and $B_1 = [A_1]_{(i,j)}$, $B_2 = [A_2]_{(i,j)}$. Since $T_{\mathsf{AEG}(\mathbb{L})}(A_1) \cap T_{\mathsf{AEG}(\mathbb{L})}(A_2) \cap \mathsf{Pr} \neq \emptyset$, by Lemma 37, both B_1 and B_2 must have the same unique variable type and are therefore intersubstitutable on argument positions. □

The above proof justifies the following:

Corollary 39. *For any* $B_1, B_2 \in \mathsf{SUB_a}(\mathsf{AEL}(\mathbb{L}))$, $B_1 \approx^a_{\mathsf{AEL}(\mathbb{L})} B_2$ *iff* $\mathsf{AEG}(\mathbb{L})$ *assigns the same variable type to both* B_1 *and* B_2*.*

Fact 40

$$\mathsf{F\text{-}CONT}_L(\mathbb{L}) = \mathsf{F\text{-}CONT}_L(\mathsf{AEL}(\mathbb{L})),$$
$$\mathsf{F\text{-}CONT}_a(\mathbb{L}) = \mathsf{F\text{-}CONT}_a(\mathsf{AEL}(\mathbb{L})).$$

We intend to show, that $\mathsf{AEL}(\mathbb{L})$ is the smallest argument-explicit language containing $\mathbb{L}$. We need some auxiliaries.

Definition 41. *Let* $\simeq_{\mathbb{L}}$ *denote the* closure *of* $\sim_{\mathbb{L}}$ *— the smallest equivalence relation on* $\mathsf{FS}(V)$*, fulfilling:*

a) $\sim_{\mathbb{L}} \subseteq \simeq_{\mathbb{L}}$,
b) if $A_1, A_2 \in \mathsf{FS}(V)$ *are such that* $\mathsf{f\text{-}cont}(A_1) = \mathsf{f\text{-}cont}(A_2)$ *and for each pair* i, j *holds* $[A_1]_{(i,j)} \simeq_{\mathbb{L}} [A_2]_{(i,j)}$*, then* $A_1 \simeq_{\mathbb{L}} A_2$*.*

Lemma 42. *Let* $\mathbb{L}'$ *be argument-explicit and* $\mathbb{L} \subseteq \mathbb{L}'$*. Then* $\simeq_{\mathbb{L}} \subseteq \approx_{\mathbb{L}'}$*.*

Proof. Since we have $\sim_{\mathbb{L}} \subseteq \sim_{\mathbb{L}'} = \approx^a_{\mathbb{L}'} \subseteq \approx_{\mathbb{L}'}$, the relation $\approx_{\mathbb{L}'}$ fulfills the condition a) from the Definition 41. By Lemma 7 it also fulfills the condition b). Hence the thesis follows from the minimality of $\simeq_{\mathbb{L}}$. □

Lemma 43. *Let* $\mathbb{L} \subseteq \mathsf{FS}(V)$ *be finitely describable. Then*

$$\approx^a_{\mathsf{AEL}(\mathbb{L})} \subseteq \simeq_{\mathbb{L}}.$$

Proof. We first show the following:

$$A_1, A_2 \in \mathsf{SUB_a}(\mathbb{L}) \wedge T_{\mathsf{AEG}(\mathbb{L})}(A_1) \cap T_{\mathsf{AEG}(\mathbb{L})}(A_2) \cap \mathsf{Var} \neq \emptyset \rightarrow A_1 \simeq_{\mathbb{L}} A_2. \quad (4)$$

If $A_1, A_2 \in \mathsf{SUB_a}(\mathbb{L})$, then the grammar $\mathsf{GF}(\mathbb{L})$ assigns some variables to A_1 and A_2, say x_i and x_j respectively. By Fact 24 there are: a finite grammar G such that $\mathsf{GF}(\{A_1, A_2\}) \subseteq G \subseteq \mathsf{GF}(\mathbb{L})$, the mgu σ of $G/\bowtie$ such that $\mathsf{AEG}(\mathbb{L}) = \sigma[G]$. For any $x, y \in \mathsf{Var}$ we will write $x \simeq y$ if there are $v \in V$, $t_1, t_2 \in I_G(v)$ such that $t_1 \bowtie t_2$ and x occurs in t_1 in the same position as y in t_2 (here types are treated as sequences of symbols representing them). Since $\sigma(x_i) = \sigma(x_j)$ there is some finite

sequence of variables $x_{i_0}, \ldots, x_{i_n}$, introduced during the construction of $\mathsf{GF}(\mathbb{L})$, such that $x_{i_0} = x_i$, $x_{i_n} = x_j$, and for each $k = 1, \ldots, n$ we have $x_{i_{k-1}} \simeq x_{i_k}$ (see [14,9]). Since $\mathsf{GF}(\mathbb{L})$ assigns each variable to at most one structure, there is exactly one sequence of structures $A_{i_0}, \ldots, A_{i_n}$ such that $x_{i_k} \in T_{\mathsf{GF}(\mathbb{L})}(A_{i_k})$ for $k = 0, \ldots, n$.

Observe that $x_{i_{k-1}}$ and x_{i_k} either both occur on argument positions of some t_1 and t_2, or $x_{i_{k-1}} = \uparrow_f(t_1)$ and $x_{i_k} = \uparrow_f(t_2)$. In the former case we have $A_{i_{k-1}} \simeq_{\mathbb{L}} A_{i_k}$, in the latter $A_{i_{k-1}} \simeq_{\mathbb{L}} A_{i_k}$ by conditions a) and b) of the Definition 41 respectively. Since $A_1 = A_{i_0}$, $A_2 = A_{i_n}$ and $\simeq_{\mathbb{L}}$ is transitive, we get $A_1 \simeq_{\mathbb{L}} A_2$, which finishes the proof of (4).

Our next task is as follows:

$$(\forall B \in \mathsf{SUB_a}(\mathsf{AEL}(\mathbb{L})))\,(\exists A \in \mathsf{SUB_a}(\mathbb{L}))$$
$$\big(A \simeq_{\mathbb{L}} B \wedge T_{\mathsf{AEG}(\mathbb{L})}(A) \cap T_{\mathsf{AEG}(\mathbb{L})}(B) \cap \mathsf{Var} \neq \emptyset\big)\,. \quad (5)$$

We prove the above by structural induction on $\mathsf{f\text{-}cont}(B)$. When $B \in V$ then $A = B$ fulfills (5).

Suppose $B \in \mathsf{SUB_a}(\mathsf{AEL}(\mathbb{L}))$ and (5) holds for all $[B]_{(i,j)}$. By Fact 40 there exists $A \in \mathsf{SUB_a}(\mathbb{L})$ such that $\mathsf{f\text{-}cont}(A) = \mathsf{f\text{-}cont}(B)$. According to Lemma 33, the grammar $\mathsf{AEG}(\mathbb{L})$ assigns to A and B the same variable type, so it suffices to show that $A \simeq_{\mathbb{L}} B$.

By the induction hypothesis, for each pair i, j there exists $C_{ij} \in \mathsf{SUB_a}(\mathbb{L})$ fulfilling: $C_{ij} \simeq_{\mathbb{L}} [B]_{(i,j)}$ and $T_{\mathsf{AEG}(\mathbb{L})}(C_{ij}) \cap T_{\mathsf{AEG}(\mathbb{L})}([B]_{(i,j)}) \cap \mathsf{Var} \neq \emptyset$. Moreover, by Lemma 37, $T_{\mathsf{AEG}(\mathbb{L})}([A]_{(i,j)}) = T_{\mathsf{AEG}(\mathbb{L})}([B]_{(i,j)}) \cap \mathsf{Var} \neq \emptyset$. Then also $T_{\mathsf{AEG}(\mathbb{L})}(C_{ij}) \cap T_{\mathsf{AEG}(\mathbb{L})}([A]_{(i,j)}) \cap \mathsf{Var} \neq \emptyset$ and therefore, by (4), $C_{ij} \simeq_{\mathbb{L}} [A]_{(i,j)}$. Consequently $[A]_{(i,j)} \simeq_{\mathbb{L}} [B]_{(i,j)}$ which justifies (5).

Finally, Assume that $B_1 \approx^a_{\mathsf{AEL}(\mathbb{L})} B_2$. By Corollary 39 we have $T_{\mathsf{AEG}(\mathbb{L})}(B_1) \cap T_{\mathsf{AEG}(\mathbb{L})}(B_2) \cap \mathsf{Var} = \{x\}$ for some $x \in \mathsf{Var}$. By (5) there are $A_1, A_2 \in \mathsf{SUB_a}(\mathbb{L})$ such that $A_1 \simeq_{\mathbb{L}} B_1$, $A_2 \simeq_{\mathbb{L}} B_2$ and $T_{\mathsf{AEG}(\mathbb{L})}(A_1) = T_{\mathsf{AEG}(\mathbb{L})}(A_2) = \{x\}$. Then we have $B_1 \simeq_{\mathbb{L}} B_2$ because $A_1 \simeq_{\mathbb{L}} A_2$ on the ground of (4). □

Theorem 44. *If $\mathbb{L} \subseteq \mathsf{FS}(V)$ is finitely describable, then $\mathsf{AEL}(\mathbb{L})$ is the smallest finitely describable argument-explicit language, containing $\mathbb{L}$.*

Proof. Let $\mathbb{L} \subseteq \mathbb{L}'$, where $\mathbb{L}'$ is argument-explicit. Let $A \in \mathsf{AEL}(\mathbb{L})$. By Fact 40 there is some $B \in \mathbb{L}$ such that $\mathsf{f\text{-}cont}(A) = \mathsf{f\text{-}cont}(B)$. For any i, j we have:
$[A]_{(i,j)} \approx^a_{\mathsf{AEL}(\mathbb{L})} [B]_{(i,j)}$ — by Fact 12,
$[A]_{(i,j)} \simeq_{\mathbb{L}} [B]_{(i,j)}$ — by Lemma 43,
$[A]_{(i,j)} \approx_{\mathbb{L}'} [B]_{(i,j)}$ — by Lemma 42.
Consequently, by Fact 8, we have $A \in \mathbb{L}'$. □

6 Learnability

We limit our attention to the learnability in the sense of Gold's identification in the limit (see [11]).

Definition 45. *Let* CCG *denote the class of all classical categorial grammars. We define a learning function* $\phi : \mathsf{FS}(V)^* \mapsto \mathsf{CCG}$*:*

$$\phi(\langle A_1, \ldots A_n\rangle) = \mathsf{AEG}(\{A_1, \ldots, A_n\}).$$

Theorem 46. *The function* ϕ *identifies in the limit (from functor-argument structures) the class of all finitely describable argument-explicit languages.*

Proof. We have to show that for any infinite sequence $A_1, A_2, \ldots$ fulfilling

$$\{A_1, A_2, \ldots\} = \mathbb{L},$$

where $\mathbb{L}$ is some finitely describable argument-explicit language, there is some $n \in \mathbb{N}$ such that for all $i \geqslant n$ we have $\mathsf{FL}(\phi(\langle A_1, \ldots, A_i\rangle)) = \mathbb{L}$. Indeed, let $\mathbb{L} \subseteq \mathsf{FS}(V)$ be an element of the class under consideration. By Theorem 44, $\mathbb{L} = \mathsf{AEL}(\mathbb{L}) = \mathsf{FL}(\mathsf{AEG}(\mathbb{L}))$. By Fact 24, there is some finite $\mathbb{D} \subseteq \mathbb{L}$ fulfilling $\mathsf{AEG}(\mathbb{E}) = \mathsf{AEG}(\mathbb{L})$ for each $\mathbb{E}$ such that $\mathbb{D} \subseteq \mathbb{E} \subseteq \mathbb{L}$. □

Fact 47. *It is easy to observe, that the learning function* ϕ *is*

- *set driven — follows directly from the definition of* ϕ*,*
- *consistent —* $\{A_1, \ldots, A_n\} \subseteq \mathsf{FL}(\phi(\langle A_1, \ldots, A_n\rangle))$*,*
- *responsive — because* ϕ *is always defined,*
- *prudent —* $\mathsf{FL}(\phi(\langle A_1, \ldots, A_n\rangle))$ *is argument-explicit,*
- *conservative — if* $A_{n+1} \in \mathsf{FL}(\phi(\langle A_1, \ldots, A_n\rangle))$ *then* $\phi(\langle A_1, \ldots, A_{n+1}\rangle) = \phi(\langle A_1, \ldots, A_n\rangle)$ *(up to the choice of variables).*

Definition 48 ([20]). *A class* $\mathscr{C}$ *of languages has* infinite elasticity *if there exist an infinite sequence of structures* $\langle A_i\rangle_{i\in\mathbb{N}}$ *and an infinite sequence* $\langle \mathbb{L}_i\rangle_{i\in\mathbb{N}}$ *of languages from* $\mathscr{C}$ *such that, for each* $i \in \mathbb{N}$

- $\{A_0, \ldots, A_i\} \subseteq \mathbb{L}_{i+1}$,
- $A_i \notin \mathbb{L}_i$

A class of languages has finite elasticity *if it does not have infinite elasticity.*

Classes with infinite elasticity are considered to be 'difficult' from the point of view of their learnability. For examples of such classes see for instance [3,4].

Theorem 49. *The class of all finitely describable argument-explicit languages has infinite elasticity.*

Proof. It suffices to show that there exists an infinite sequence $\langle \mathbb{L}_i\rangle_{i\in\mathbb{N}}$ of argument-explicit languages with the property: $\mathbb{L}_0 \subsetneq \mathbb{L}_1 \subsetneq \cdots \subsetneq \mathbb{L}_n \subsetneq \ldots$ Let $\mathbb{L}_0$ be any finitely describable argument-explicit language. By induction, let A_i be any structure such that $\mathsf{f\text{-}cont}(A_i) \notin \mathsf{F\text{-}CONT}(\mathbb{L})$. Such a structure always exists since every finitely describable language is bounded in height and width. We define $\mathbb{L}_{i+1} = \mathsf{AEL}(\mathbb{L}_i \cup \{A_i\})$. Then of course $A_i \in \mathbb{L}_{i+1} \setminus \mathbb{L}_i$ and $\mathbb{L}_i \subseteq \mathbb{L}_{i+1}$. □

References

1. Angluin, D.: Finding patterns common to a set of strings. J. Comput. Syst. Sci. 21(1), 46–62 (1980)
2. Angluin, D.: Inductive inference of formal languages from positive data. Information and Control 45(2), 117–135 (1980)
3. Angluin, D.: Inference of reversible languages. Journal of the ACM 29(3), 741–765 (1982)
4. Béchet, D., Foret, A.: k-valued non-associative lambek grammars are learnable from generalized functor-argument structures. Theor. Comput. Sci. 355(2), 139–152 (2006)
5. Buszkowski, W.: Typed functorial languages. Bull. Polish Acad. Sci. Math. 21, 495–505 (1986)
6. Buszkowski, W.: Discovery procedures for categorial grammars. In: Klein, E., van Benthem, J. (eds.) Categories, Polymorphism and Unification. Universiteit van Amsterdam, Amsterdam (1987)
7. Buszkowski, W.: Solvable problems for classical categorial grammars. Bull. Polish Acad. Sci. Math. (35), 373–382 (1987)
8. Buszkowski, W.: Classical categorial grammars. In: Kálmán, L., Pólos, L. (eds.) Papers from the Second Symposium on Logic and Language, Budapest, Akadémiai Kiadó, pp. 243–260 (1990)
9. Buszkowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Studia Logica XLIX(4), 431–454 (1990)
10. Clark, A., Eyraud, R.: Polynomial identification in the limit of substitutable context-free languages. Journal of Machine Learning Research 8, 1725–1745 (2007)
11. Gold, E.M.: Language identification in the limit. Information and Control 10, 447–474 (1967)
12. Kanazawa, M.: Identification in the limit of categorial grammars. Journal of Logic, Language and Information 5(2), 115–155 (1996)
13. Kanazawa, M.: Learnable Classes of Categorial Grammars. In: Studies in Logic, Language and Information. CSLI Publications & FoLLI, Stanford (1998)
14. Lloyd, J.W.: Foundations of Logic Programming. Springer, Berlin (1987)
15. Marciniec, J.: Infinite set unification with application to categorial grammar. Studia Logica LVIII(3), 339–355 (1997)
16. Marciniec, J.: Optimal unification of infinite sets of types. Fundamenta Informaticae 62(3,4), 395–407 (2004)
17. Solomonoff, R.J.: A formal theory of inductive inference. part I. Information and Control 7(1), 1–22 (1964)
18. Solomonoff, R.J.: A formal theory of inductive inference. part II. Information and Control 7(2), 224–254 (1964)
19. Tarski, A.: A decision method for elementary algebra and geometry. Technical Report R-109, The Rand Corporation, Santa Monica, CA, USA (1957)
20. Wright, K.: Identification of unions of languages drawn from an identifiable class. In: COLT 1989: Proceedings of the Second Annual Workshop on Computational Learning Theory, pp. 328–333. Morgan Kaufmann Publishers Inc., San Francisco (1989)

Globally Deterministic CD-Systems of Stateless R(1)-Automata*

Benedek Nagy[1] and Friedrich Otto[2]

[1] Department of Computer Science, Faculty of Informatics
University of Debrecen, Egyetem tér 1, 4032 Debrecen, Hungary
nbenedek@inf.unideb.hu
[2] Fachbereich Elektrotechnik/Informatik
Universität Kassel, 34109 Kassel, Germany
otto@theory.informatik.uni-kassel.de

Abstract. Although the component automata of a cooperating distributed system (CD-system) of stateless deterministic R(1)-automata are all deterministic, the CD-system itself is not. Here we study CD-systems of stateless deterministic R(1)-automata that are themselves completely deterministic. These CD-systems correspond to *deterministic finite-state acceptors with translucent letters*. We investigate the expressive power of these systems, study the closure properties of the class of languages they accept, and show that the inclusion problem for these systems is undecidable, while their universe problem is decidable.

Keywords: Restarting automaton, cooperating distributed system.

1 Introduction

Cooperating distributed systems (CD-systems) of restarting automata have been defined in [4], and in [5,6] various types of deterministic CD-systems of restarting automata have been studied. As expected CD-systems are much more expressive than their component automata themselves. On the other hand, stateless restarting automata, that is, restarting automata with only a single state, have been introduced and studied in [3]. In [8] we introduced CD-systems of stateless deterministic R(1)-automata, that is, of stateless deterministic restarting automata with a read/write window of size 1 that restart immediately after executing a rewrite operation. The languages accepted by these CD-systems in mode = 1 have semi-linear Parikh images, they include all rational trace languages, and this class of languages is closed under union, product, Kleene star, and commutative closure, but it is not closed under intersection with regular languages,

* This work was supported by grants from the Balassi Intézet Magyar Ösztöndíj Bizottsága (MÖB) and the Deutsche Akademischer Austauschdienst (DAAD). The first author was also supported by the TÁMOP 4.2.1/B-09/1/KONV-2010-0007 project, which is implemented through the New Hungary Development Plan, co-financed by the European Social Fund and the European Regional Development Fund.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 390–401, 2011.

complementation, or ε-free morphisms [7]. In addition, for these CD-systems the emptiness and the finiteness problems are easily solvable, while the regularity, inclusion, and equivalence problems are undecidable in general.

Although all the component automata of such a CD-system are deterministic, in general the CD-system itself is not. In fact, these CD-systems correspond to *nondeterministic finite-state acceptors with translucent letters* [9]. Here we study CD-systems of stateless deterministic R(1)-automata that are themselves completely deterministic. Actually, following the development in [5,6] we introduce two different kinds of deterministic CD-systems: the *strictly deterministic* systems and the *globally deterministic* systems. In a *strictly deterministic* system, there is a single initial component, and each component automaton has a single successor component only. This ensures that all computations of such a system are deterministic, but at the same time it severely restricts the expressive power of these systems. In fact, these systems do not even accept all finite languages. Therefore, we concentrate on *globally deterministic* systems, which also have a single initial component only, but for which the successor component of an R(1)-automaton is chosen deterministically based on the symbol that is deleted in the current cycle. This still guarantees that each computation is deterministic, but it allows for much more flexibility. In fact, globally deterministic CD-systems of R(1)-automata correspond to *deterministic finite-state acceptors with translucent letters* [9]. These systems accept all regular languages, but they are strictly less expressive than the CD-systems considered in [8].

This paper is structured as follows. In Section 2 we repeat the definition of stl-det-local-CD-R(1)-systems from [8], we restate some of their main properties, and we shortly consider strictly deterministic CD-systems of stateless deterministic R(1)-automata. Then in Section 3 we define the stl-det-global-CD-R(1)-systems. We show that these systems accept all regular languages, we present a normal form for them, and we prove that they do not accept all rational trace languages. Also we present some closure and non-closure properties on the class of languages accepted by these CD-systems, showing that with respect to closure properties these systems are much weaker than the systems of [8]. Finally, we consider decision problems for these systems in Section 4. While the decidability of the membership, emptiness, and finiteness problems follows immediately from the corresponding results in [7], closure under complementation implies that also the universe problem is decidable for these systems. This is an important constrast to the situation for stl-det-local-CD-R(1)-systems, where the regularity, inclusion, and equivalence problems are shown to be undecidable by a reduction from the universe problem. Here we present a reduction from the Post Correspondence Problem to show that the inclusion problem is undecidable for stl-det-global-CD-R(1)-systems. The paper closes with a short summary and some open problems.

2 CD-Systems of Stateless Deterministic R(1)-Automata

A *stateless deterministic* R(1)*-automaton* is a one-tape machine that is described by a 5-tuple $M = (\Sigma, ¢, \$, 1, \delta)$, where Σ is a finite alphabet, the symbols $¢, \$ \notin \Sigma$

serve as markers for the left and right border of the workspace, respectively, the size of the *read/write window* is 1, and $\delta : \Sigma \cup \{¢, \$\} \to \{\mathsf{MVR}, \mathsf{Accept}, \varepsilon\}$ is the (partial) *transition function*. There are three types of transition steps: *move-right steps* (MVR), which shift the window one step to the right, combined *rewrite/restart steps* (denoted by ε), which delete the content a of the window, thereby shortening the tape, and place the window over the left end of the tape, and *accept steps* (Accept), which cause the automaton to halt and accept. We use the notation $\delta(a) = \emptyset$ to express the fact that the function δ is undefined on a. Some additional restrictions apply in that the sentinels ¢ and \$ must not be deleted, and that the window must not move right on seeing the \$-symbol.

A *configuration* of M is described by a pair (α, β), where either $\alpha = \varepsilon$ (the empty word) and $\beta \in \{¢\} \cdot \Sigma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Sigma^*$ and $\beta \in \Sigma^* \cdot \{\$\}$; here $\alpha\beta$ is the current content of the tape, and it is understood that the head scans the first symbol of β. A *restarting configuration* is of the form $(\varepsilon, ¢w\$)$, where $w \in \Sigma^*$; to simplify the notation a restarting configuration $(\varepsilon, ¢w\$)$ is usually simply written as $¢w\$$. By $\vdash_M$ we denote the single-step computation relation of M, and $\vdash_M^*$ denotes the reflexive transitive closure of $\vdash_M$.

The automaton M proceeds as follows. Starting from an initial configuration $¢w\$$, the window moves right until a configuration of the form $(¢x, ay\$)$ is reached such that $\delta(a) = \varepsilon$. Here $w = xay$ and $a \in \Sigma$. Now the latter configuration is transformed into the restarting configuration $¢xy\$$. This sequence of computational steps, which is called a *cycle*, is expressed as $w \vdash_M^c xy$. A computation of M consists of a finite sequence of cycles that is followed by a tail computation, which consists of a sequence of move-right operations possibly followed by an accept step. An input word $w \in \Sigma^*$ is *accepted* by M, if the computation of M which starts with the initial configuration $¢w\$$ finishes by executing an accept step. By $L(M)$ we denote the language consisting of all words accepted by M.

We can partition the alphabet Σ into four disjoint subalphabets:

$$(1.)\ \Sigma_1 = \{\, a \in \Sigma \mid \delta(a) = \mathsf{MVR} \,\}, \quad (3.)\ \Sigma_3 = \{\, a \in \Sigma \mid \delta(a) = \mathsf{Accept} \,\},$$
$$(2.)\ \Sigma_2 = \{\, a \in \Sigma \mid \delta(a) = \varepsilon \,\}, \quad (4.)\ \Sigma_4 = \{\, a \in \Sigma \mid \delta(a) = \emptyset \,\}.$$

It has been shown in [8] that the language $L(M)$ can be characterized as

$$L(M) = \begin{cases} \Sigma^*, & \text{if } \delta(¢) = \mathsf{Accept}, \\ (\Sigma_1 \cup \Sigma_2)^* \cdot \Sigma_3 \cdot \Sigma^*, & \text{if } \delta(¢) = \mathsf{MVR} \text{ and } \delta(\$) \neq \mathsf{Accept}, \\ (\Sigma_1 \cup \Sigma_2)^* \cdot ((\Sigma_3 \cdot \Sigma^*) \cup \{\varepsilon\}), & \text{if } \delta(¢) = \mathsf{MVR} \text{ and } \delta(\$) = \mathsf{Accept}. \end{cases}$$

A CD-system of stateless deterministic R(1)-automata consists of a finite collection $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0)$ of stateless deterministic R(1)-automata $M_i = (\Sigma, ¢, \$, 1, \delta_i)$ $(i \in I)$, *successor relations* $\sigma_i \subseteq I$ $(i \in I)$, and a subset $I_0 \subseteq I$ of *initial indices*. Here it is required that $I_0 \neq \emptyset$, and that $\sigma_i \neq \emptyset$ for all $i \in I$. In [8] it was required in addition that $i \notin \sigma_i$ for all $i \in I$, but this requirement is easily met by using two isomorphic copies of each component automaton M_i, $i \in I$. Therefore, we abondon it here in order to simplify the presentation.

Various modes of operation have been introduced and studied for CD-systems of restarting automata, but here we are only interested in mode = 1 computations. A computation of $\mathcal{M}$ in mode = 1 on an input word w proceeds as follows.

First an index $i_0 \in I_0$ is chosen nondeterministically. The automaton M_{i_0} starts the computation with the initial configuration $¢w\$$, and executes a single cycle. Thereafter an index $i_1 \in \sigma_{i_0}$ is chosen nondeterministically, and M_{i_1} continues the computation by executing a single cycle. This continues until, for some $l \geq 0$, the machine M_{i_l} accepts. Such a computation will be denoted as

$$(i_0, w) \vdash^c_{\mathcal{M}} (i_1, w_1) \vdash^c_{\mathcal{M}} \cdots \vdash^c_{\mathcal{M}} (i_l, w_l) \vdash^*_{M_{i_l}} \mathsf{Accept}.$$

Should at some stage the chosen automaton M_{i_l} be unable to execute a cycle or to accept, then the computation fails. By $L_{=1}(\mathcal{M})$ we denote the language that consists of all words $w \in \Sigma^*$ that are accepted by $\mathcal{M}$ in mode $= 1$. By $\mathcal{L}_{=1}(\mathsf{stl\text{-}det\text{-}local\text{-}CD\text{-}R(1)})$ we denote the class of languages that are accepted by mode $= 1$ computations of stl-det-local-CD-R(1)-systems, that is, by CD-systems of stateless deterministic R(1)-automata.

Example 1. Let $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0)$, where $\Sigma = \{a, b, c\}$, $I = \{a, b, c\}$, $I_0 = \{a\}$, $\sigma_a = \{b\}$, $\sigma_b = \{c\}$, $\sigma_c = \{a\}$, and M_a, M_b, and M_c are the stateless deterministic R(1)-automata that are given by the following transition functions:

$$\begin{aligned}
M_a &: \delta_a(¢) = \mathsf{MVR}, \quad \delta_a(a) = \varepsilon, \quad \delta_a(\$) = \mathsf{Accept},\\
M_b &: \delta_b(¢) = \mathsf{MVR}, \quad \delta_b(a) = \mathsf{MVR}, \delta_b(c) = \mathsf{MVR}, \quad \delta_b(b) = \varepsilon,\\
M_c &: \delta_c(¢) = \mathsf{MVR}, \quad \delta_c(a) = \mathsf{MVR}, \delta_c(b) = \mathsf{MVR}, \quad \delta_c(c) = \varepsilon.
\end{aligned}$$

Then $L_{=1}(\mathcal{M})$ is the non-context-free language $L_{abc} = \{\, w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c \geq 0, \text{ and for each prefix } u \text{ of } w : |u|_a \geq \max\{|u|_b, |u|_c\} \,\}$.

In [8] the following results were established. Here $\psi : \Sigma^* \to \mathbb{N}^{|\Sigma|}$ denotes the Parikh mapping defined by $\psi(w) = (|w|_{a_1}, \ldots, |w|_{a_n})$, if $\Sigma = \{a_1, \ldots, a_n\}$.

Proposition 1. (a) *Each language $L \in \mathcal{L}_{=1}(\mathsf{stl\text{-}det\text{-}local\text{-}CD\text{-}R(1)})$ contains a regular sublanguage E such that $\psi(L) = \psi(E)$ holds.*

(b) *$\mathcal{L}_{=1}(\mathsf{stl\text{-}det\text{-}local\text{-}CD\text{-}R(1)})$ properly contains the class of all rational trace languages, and therewith it contains all regular languages.*

It follows from Proposition 1 (a) that each language which is accepted by a stl-det-local-CD-R(1)-system in mode $= 1$ is semi-linear, that is, it has a semi-linear Parikh image. As the deterministic linear language $L = \{\, a^n b^n \mid n \geq 0 \,\}$ does not contain a regular sublanguage that is letter-equivalent to the language itself, L is not accepted by any stl-det-local-CD-R(1)-system. Together with Example 1 this implies that the language class $\mathcal{L}_{=1}(\mathsf{stl\text{-}det\text{-}local\text{-}CD\text{-}R(1)})$ is incomparable to the classes DLIN, LIN, DCFL, and CFL with respect to inclusion. Here DLIN denotes the class of *deterministic linear languages*, which is the class of languages that are accepted by deterministic one-turn pushdown automata, LIN is the class of *linear languages*, and DCFL and CFL denote the classes of *deterministic context-free* and *context-free languages*.

Although all the component automata of a stl-det-local-CD-R(1)-system are deterministic, in general the system itself is not. Here we introduce a type of CD-system of stateless R(1)-automata that is completely deterministic. The idea and

the notation is taken from [5], where a corresponding notion was introduced for CD-systems of general restarting automata.

A CD-system $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0)$ of stateless deterministic R(1)-automata is called *strictly deterministic* if $|I_0| = 1$ and $|\sigma_i| = 1$ for all $i \in I$. Then, for each word $w \in \Sigma^*$, $\mathcal{M}$ has a unique computation that begins with the initial configuration corresponding to input w. Thus, $\mathcal{M}$ is completely deterministic. By $\mathcal{L}_{=1}(\text{stl-det-strict-CD-R}(1))$ we denote the class of languages that are accepted by strictly deterministic stateless CD-R(1)-systems working in mode $= 1$.

Observe that the CD-system in Example 1 is strictly deterministic. On the other hand, we have the following negative result, which is easily established.

Lemma 1. *The finite language $L_0 = \{aaa, bb\}$ is not accepted by any strictly deterministic stateless* CD-R(1)*-system working in mode $= 1$.*

This yields the following immediate consequence.

Corollary 1. *The language class $\mathcal{L}_{=1}(\text{stl-det-strict-CD-R}(1))$ is incomparable under inclusion to the language classes* FIN *of finite languages,* REG *of regular languages, and* CFL *of context-free languages. In particular, it follows that the inclusion $\mathcal{L}_{=1}(\text{stl-det-strict-CD-R}(1)) \subseteq \mathcal{L}_{=1}(\text{stl-det-local-CD-R}(1))$ is strict.*

From the definition it is easily derived that $\mathcal{L}_{=1}(\text{stl-det-strict-CD-R}(1))$ is closed under complementation. On the other hand, based on Lemma 1 it can be shown that this class is an anti-AFL, that is, it is not closed under union, product, Kleene star, intersection with regular sets, ε-free morphisms, and inverse morphisms. Further, it is neither closed under reversal nor under the operation of taking the commutative closure.

3 Globally Deterministic CD-R(1)-Systems

A *stateless globally deterministic CD-R(1)-system* $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0, \delta)$ consists of a CD-system of stateless deterministic R(1)-automata $((M_i, \sigma_i)_{i \in I}, I_0)$ satisfying $|I_0| = 1$ and a *global successor function* $\delta : \bigcup_{i \in I}(\{i\} \times \Sigma_2^{(i)}) \to I$ satisfying $\delta(i, a) \in \sigma_i$ for all $i \in I$ and all $a \in \Sigma_2^{(i)}$. Here, for each $i \in I$, $\Sigma_2^{(i)}$ denotes the set of letters that are deleted by the component automaton M_i. The global successor function is used to determine the successor components within computations of $\mathcal{M}$. If (i, w) is the current configuration of $\mathcal{M}$, that is, the component M_i is activated with the restarting configuration $\text{¢}w\$$, and $w = uav$ for some $u \in \Sigma_1^{(i)*}$ and $a \in \Sigma_2^{(i)}$, then $\delta(i, a) \in \sigma_i$ gives the successor component, that is, we have the partial computation $(i, w) = (i, uav) \vdash_{\mathcal{M}}^{c} (j, uv)$, where $j = \delta(i, a)$. It follows that, for each input word $w \in \Sigma^*$, the system $\mathcal{M}$ has a unique computation that starts from the initial configuration corresponding to input w, that is, $\mathcal{M}$ is completely deterministic. By $\mathcal{L}_{=1}(\text{stl-det-global-CD-R}(1))$ we denote the class of languages that are accepted by stateless globally deterministic CD-R(1)-systems working in mode $= 1$.

Obviously, each stateless strictly deterministic CD-R(1)-system is globally deterministic. However, the stateless globally deterministic CD-R(1)-systems are

much more expressive than the strictly deterministic ones. In fact, we have the following proper inclusion.

Lemma 2. $\mathsf{REG} \subsetneq \mathcal{L}_{=1}(\mathsf{stl\text{-}det\text{-}global\text{-}CD\text{-}R(1)})$.

Proof. From Example 1 we see that $\mathcal{L}_{=1}(\mathsf{stl\text{-}det\text{-}global\text{-}CD\text{-}R(1)})$ contains languages that are not even context-free. Thus, it remains to show that each regular language is accepted by a stateless globally deterministic CD-R(1)-system.

Let $L \subseteq \Sigma^*$ be a regular language, and let $A = (Q, \Sigma, p_0, F, \delta_A)$ be a complete deterministic finite-state acceptor for L. From A we construct a stl-det-global-CD-R(1)-system $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0, \delta)$ as follows:

- $I = Q$, $I_0 = \{p_0\}$, $\sigma_i = I$ for all $i \in I$,
- for each $i \in I$, the automaton M_i is defined through

$$\delta_i(¢) = \mathsf{MVR},\ \delta_i(a) = \varepsilon \text{ for all } a \in \Sigma,\ \delta_i(\$) = \mathsf{Accept},\ \text{if } i \in F,$$

- and δ is defined through $\delta(i, a) = \delta_A(i, a)$ for all $i \in I$ and all $a \in \Sigma$.

By induction on $|w|$ it is now easily shown that, for all $w \in \Sigma^*$ and all $i \in I$, $\delta_A(p_0, w) = i$ if and only if $(p_0, w) \vdash_{\mathcal{M}}^{c^*} (i, \varepsilon)$. Hence, it follows that $L_{=1}(\mathcal{M}) = L$ holds. □

In particular, this yields the following proper inclusion.

Corollary 2. $\mathcal{L}_{=1}(\mathsf{stl\text{-}det\text{-}strict\text{-}CD\text{-}R(1)}) \subsetneq \mathcal{L}_{=1}(\mathsf{stl\text{-}det\text{-}global\text{-}CD\text{-}R(1)})$.

To simplify the discussions and proofs below we now introduce a normal form for stl-det-global-CD-R(1)-systems.

Definition 1. *A* stl-det-global-CD-R(1)*-system* $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, \{i_0\}, \delta)$ *is in* normal form *if it satisfies the following conditions:*

1. *Each component* M_i *is reachable from the initial component* M_{i_0}*, that is, for each* $i \in I$*, there exists an input* $w \in \Sigma^*$ *such that* $(i_0, w) \vdash_{\mathcal{M}}^{c^*} (i, z)$ *holds for some* $z \in \Sigma^*$*.*
2. *For each component* M_i*,* $\delta_i(¢) = \mathsf{MVR}$*.*
3. *For each component* M_i *and each letter* $a \in \Sigma$*,* $\delta_i(a) \in \{\mathsf{MVR}, \varepsilon\}$*, that is,* $\Sigma_3^{(i)} = \emptyset = \Sigma_4^{(i)}$ *for all* $i \in I$*.*

Thus, if $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, \{i_0\}, \delta)$ is in *normal form*, then each computation of $\mathcal{M}$ ends with a component that accepts or rejects on the \$-symbol. The following normalization result holds.

Proposition 2. *From a* stl-det-global-CD-R(1)*-system* $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, \{i_0\}, \delta)$*, a* stl-det-global-CD-R(1)*-system* $\mathcal{M}' = ((M'_j, \sigma'_j)_{j \in J}, \{j_0\}, \delta')$ *can be constructed such that* $\mathcal{M}'$ *is in normal form, and* $L_{=1}(\mathcal{M}') = L_{=1}(\mathcal{M})$*.*

Based on this normal form result the following inclusion can be derived.

Proposition 3. $\mathcal{L}_{=1}(\mathsf{stl\text{-}det\text{-}global\text{-}CD\text{-}R(1)}) \subseteq \mathcal{L}_{=1}(\mathsf{stl\text{-}det\text{-}local\text{-}CD\text{-}R(1)})$.

Below we prove that this inclusion is strict by showing that stl-det-global-CD-R(1)-systems do not even accept all rational trace languages. Recall from [8] (see also, e.g., [2]) that a language $L \subseteq \Sigma^*$ is a *rational trace language* if there exists a reflexive and transitive binary relation D on Σ (a *dependency relation*) such that $L = \bigcup_{w \in R} [w]_D$ for some regular language R on Σ. Here $[w]_D$ denotes the congruence class of w with respect to the congruence $\equiv_D = \{ (uabv, ubav) \mid u, v \in \Sigma^*, a, b \in \Sigma, (a, b) \notin D \}$.

Proposition 4. *The rational trace language*

$$L_\vee = \{ w \in \{a, b\}^* \mid \exists n \geq 0 : |w|_a = n \text{ and } |w|_b \in \{n, 2n\} \}$$

is not accepted by any stateless globally deterministic CD-R(1)*-system.*

Proof. Obviously, $L_\vee$ is simply the commutative closure of the regular language $R_\vee = (ab)^* \cup (abb)^*$, and hence, it is indeed a rational trace language.

Assume that $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, I_0, \delta)$ is a stl-det-global-CD-R(1)-system such that $L_{=1}(\mathcal{M}) = L_\vee$. Without loss of generality we can assume that $I = \{0, 1, \ldots, m-1\}$ and that $I_0 = \{0\}$.

Let $n > 2m$, and let $w = a^n b^n \in L_\vee$. Then the computation of $\mathcal{M}$ on input w is accepting, that is, it is of the form

$$(0, a^n b^n) \vdash^c_{\mathcal{M}} (i_1, w_1) \vdash^c_{\mathcal{M}} \cdots \vdash^c_{\mathcal{M}} (i_r, w_r) \vdash^*_{M_{i_r}} \mathsf{Accept},$$

where M_{i_r} accepts the tape contents $¢w_r\$$. If $|w_r|_a > 0$ and $|w_r|_b > 0$, then M_{i_r} would also accept the tape contents $w_r a^n b^{5m}$ for any $m \geq 0$, and therewith $\mathcal{M}$ would accept the input $wa^n b^{5n} = a^n b^n a^n b^{5n}$. As this word is not contained in $L_\vee$, this contradicts our assumption that $L_{=1}(\mathcal{M}) = L_\vee$. Hence, it follows that $|w_r|_a = 0$ or $|w_r|_b = 0$. If $w_r = a^s$ for some $s > 0$, then it follows analogously that with w, $\mathcal{M}$ would also accept the word wa^m for all $m \geq 0$. Hence, it would accept the word $wa^n = a^n b^n a^n \notin L_\vee$, which yields the same contradiction as above. Thus, $|w_r|_a = 0$, and analogously it can be shown that $|w_r|_b = 0$, that is, $w_r = \varepsilon$. Hence, in the above computation $2n$ cycles are executed that delete the input $w = a^n b^n$ symbol by symbol, and then M_{i_r} accepts the empty word.

As $n > m$, there exists an index $i \in I$ and integers $s, t, k, \ell \geq 0$, $m \geq s + t \geq 0$ and $m \geq k + \ell > 0$, such that the above computation can be written as follows:

$$(0, a^n b^n) \vdash^{c^*}_{\mathcal{M}} (i, a^{n-s} b^{n-t}) \vdash^{c^+}_{\mathcal{M}} (i, a^{n-s-k} b^{n-t-\ell}) \vdash^{c^*}_{\mathcal{M}} (i_r, \varepsilon) \vdash^*_{M_{i_r}} \mathsf{Accept}.$$

Obviously, $\mathcal{M}$ will also execute the following shortened computation:

$$(0, a^{n-k} b^{n-\ell}) \vdash^{c^*}_{\mathcal{M}} (i, a^{n-s-k} b^{n-t-\ell}) \vdash^{c^*}_{\mathcal{M}} (i_r, \varepsilon) \vdash^*_{M_{i_r}} \mathsf{Accept},$$

that is, $\mathcal{M}$ accepts on input $a^{n-k} b^{n-\ell}$. From our assumption that $L_{=1}(\mathcal{M}) = L_\vee$ we can therefore conclude that $k = \ell$, as $n > 2m$.

Now consider the computation of $\mathcal{M}$ on input $a^n b^{2n}$. As $a^n b^{2n} \in L_\vee$, this computation is accepting, that is, it has the following form:

$$(0, a^n b^{2n}) \vdash^{c^*}_{\mathcal{M}} (i, a^{n-s} b^{2n-t}) \vdash^{c^+}_{\mathcal{M}} (i, a^{n-s-k} b^{2n-t-k}) \vdash^{c^*}_{\mathcal{M}} (i', \varepsilon) \vdash^*_{M_{i'}} \mathsf{Accept}.$$

But then $\mathcal{M}$ will also execute the following computation:

$$(0, a^{n-k}b^{2n-k}) \vdash_{\mathcal{M}}^{c^*} (i, a^{n-s-k}b^{2n-t-k}) \vdash_{\mathcal{M}}^{c^*} (i', \varepsilon) \vdash_{M_{i'}}^{*} \mathsf{Accept},$$

that is, it accepts on input $a^{n-k}b^{2n-k} \notin L_\vee$. It follows that $L_{=1}(\mathcal{M}) \neq L_\vee$, that is, $L_\vee$ is not accepted by any stateless globally deterministic CD-R(1)-system working in mode $= 1$. □

As all rational trace languages are accepted by stateless locally deterministic CD-R(1)-systems, it follows that the inclusion in Proposition 3 is proper. The Dyck language $D_1'^*$ (see, e.g., [1]) is not a rational trace language, but it is easily seen that it is accepted by a stateless strictly deterministic CD-R(1)-system. Thus, we have the following consequence.

Corollary 3. $\mathcal{L}_{=1}$(stl-det-strict-CD-R(1)) *and* $\mathcal{L}_{=1}$(stl-det-global-CD-R(1)) *are incomparable to the class of rational trace languages with respect to inclusion.*

Next we concentrate on closure properties of $\mathcal{L}_{=1}$(stl-det-global-CD-R(1)).

Proposition 5. (a) *The language class* $\mathcal{L}_{=1}$(stl-det-global-CD-R(1)) *is closed under complementation.*

(b) *The language class* $\mathcal{L}_{=1}$(stl-det-global-CD-R(1)) *is not closed under union, intersection with regular sets, and alphabetic morphisms.*

(c) *The language class* $\mathcal{L}_{=1}$(stl-det-global-CD-R(1)) *is not closed under the operation of taking the commutative closure.*

Let $L_\geq$ be the language $L_\geq = \{ w \in \Sigma_0^* \mid |w|_a \geq |w|_b \geq 0 \}$ on $\Sigma_0 = \{a, b\}$. For this language we have the following technical results.

Lemma 3. (a) $L_\geq \in \mathcal{L}_{=1}$(stl-det-global-CD-R(1)).

(b) $L_\geq$ *is not accepted by any* stl-det-global-CD-R(1)*-system that first completely erases the given input and that then accepts on the empty word.*

Lemma 3 implies in particular that for stl-det-global-CD-R(1)-systems we do not have the *strong normal form* that we have for stl-det-local-CD-R(1)-systems (see, [7]). Based on this technical lemma we can now prove the following non-closure property.

Corollary 4. *The language class* $\mathcal{L}_{=1}$(stl-det-global-CD-R(1)) *is not closed under product.*

Proof. We consider the product $L_{pr} = L_\geq \cdot L_c$ of the languages $L_\geq$ and $L_c = \{c\}$, where $c \notin \{a, b\}$ is a new letter. While $L_\geq \in \mathcal{L}_{=1}$(stl-det-global-CD-R(1)) by Lemma 3 (a), $L_c \in \mathcal{L}_{=1}$(stl-det-global-CD-R(1)) is obvious. We claim, however, that the product L_{pr} is not accepted by any stl-det-global-CD-R(1)-system.

Assume to the contrary that $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, \{i_0\}, \delta)$ is a stl-det-global-CD-R(1)-system such that $L_{=1}(\mathcal{M}) = L_{pr}$.

Claim. For each word $w = uc \in L_{pr}$, the accepting computation of $\mathcal{M}$ on input w is of the form $(i_0, w) = (i_0, uc) \vdash_{\mathcal{M}}^{c^{|u|}} (i_m, c) \vdash_{\mathcal{M}}^{c} (j, \varepsilon) \vdash_{M_j}^{*} \mathsf{Accept}$.

Proof. As $\mathcal{M}$ must verify that the given input w ends with the symbol c, one of the components of $\mathcal{M}$ must read the symbol c in the course of the accepting computation of $\mathcal{M}$ on input w. Assume that M_{i_m} is this particular component. If $\delta_{i_m}(c)$ is undefined, then M_{i_m} would get stuck, and so the computation of $\mathcal{M}$ on input w would not accept. Thus, $\delta_{i_m}(c)$ is defined.

If $\delta_{i_m}(c) = \mathsf{MVR}$, then after executing the corresponding step, M_{i_m} would read the \$-symbol. As the computation considered is accepting, this means that M_{i_m} must accept at this point. But then $\mathcal{M}$ would also accept the word $wc = ucc \notin L_{pr}$, as the computation of $\mathcal{M}$ on input wc would be exactly the same as the one on input w. This, however, contradicts our assumption above. Hence, it follows that $\delta_{i_m}(c) = \varepsilon$.

Let $j = \delta(i_m, c)$ be the index of the corresponding successor component. Then the accepting computation of $\mathcal{M}$ on input w has the following form:

$$(i_0, w) = (i_0, uc) \vdash^r_{\mathcal{M}} (i_m, vc) \vdash^c_{\mathcal{M}} (j, v) \vdash^*_{\mathcal{M}} \mathsf{Accept}.$$

Thus, it remains to show that $v = \varepsilon$, that is, $r = |u|$ must hold. Assume to the contrary that $v \neq \varepsilon$. Then $\delta_{i_m}(x) = \mathsf{MVR}$ for all letters $x \in \{a, b\}$ satisfying $|v|_x \geq 1$. Let $v = v'x$, where $x \in \{a, b\}$, and let $z = u'cx$ be the word that is obtained from $w = uc$ by moving the last occurrence of the letter x to the right end of the word. Then the computation of $\mathcal{M}$ on input z looks as follows:

$$(i_0, z) = (i_0, u'cx) \vdash^r_{\mathcal{M}} (i_m, v'cx) \vdash^c_{\mathcal{M}} (j, v'x) = (j, v) \vdash^*_{\mathcal{M}} \mathsf{Accept}.$$

This, however, contradicts our assumption above, as $z = u'cx \notin L_{pr}$. Hence, it follows that $v = \varepsilon$, which proves the above claim. □

Note that, for each component M_i that can only encounter an occurrence of the symbol c in a non-accepting computation, we can simply take $\delta_i(c)$ to be undefined.

Now we modify the system $\mathcal{M}$ to obtain a stl-det-global-CD-R(1)-system $\mathcal{M}'$ as follows. For each index $i \in I$, if $\delta_i(c)$ is defined, that is, if $\delta_i(c) = \varepsilon$ according to our observations above, then we remove this transition and take $\delta_i(\$) = \mathsf{Accept}$. Then, for each word $u \in L_{\geq}$, the computation of $\mathcal{M}'$ on input u will parallel the computation of $\mathcal{M}$ on input uc, and thus, we see from the claim above that it will first erase u completely and then accept on reaching the empty word. Now Lemma 3 (b) implies that $L_{\geq} \subsetneq L_{=1}(\mathcal{M}')$, that is, there exists a word $u \in \{a, b\}^* \setminus L_{\geq}$ such that $\mathcal{M}'$ accepts on input u. But then $\mathcal{M}$ will accept on input $uc \notin L_{pr}$, which contradicts our assumption above. Hence, it follows that L_{pr} is not accepted by any stl-det-global-CD-R(1)-system. □

Consider the language $L^R_{pr} = \{\, cw \mid w \in \{a, b\}^*, |w|_a \geq |w|_b \geq 0 \,\}$. From Lemma 3 (a) we have a stl-det-global-CD-R(1)-system $\mathcal{M} = ((M_i, \sigma_i)_{i \in I}, \{0\}, \delta)$ for $L_{\geq}$. Let $\mathcal{M}'$ be obtained from $\mathcal{M}$ by introducing a new initial component M_{ini} that is defined by the transition function $\delta_{ini}(\mathfrak{c}) = \mathsf{MVR}$, $\delta_{ini}(c) = \varepsilon$, and the successor set $\sigma_{ini} = \{0\}$, and by extending the successor function δ by taking $\delta(ini, c) = 0$. Then it is easily seen that $L_{=1}(\mathcal{M}') = L^R_{pr}$ holds. Thus, together with the fact that the language L_{pr} is not accepted by any stl-det-global-CD-R(1)-system this yields the following additional non-closure result.

Corollary 5. *The language class* $\mathcal{L}_{=1}$(stl-det-global-CD-R(1)) *is not closed under reversal.*

Finally we claim that $\mathcal{L}_{=1}$(stl-det-global-CD-R(1)) is not closed under Kleene star. To derive this result we introduce the language $L_{pra} = L_{pr} \cdot \{a\}^*$, for which the following results can be shown.

Lemma 4

(a) $L_{pra} \in \mathcal{L}_{=1}$(stl-det-global-CD-R(1)).
(b) $L_* = (L_{pra})^* \notin \mathcal{L}_{=1}$(stl-det-global-CD-R(1)).

This lemma yields the following non-closure result.

Corollary 6. *The language class* $\mathcal{L}_{=1}$(stl-det-global-CD-R(1)) *is not closed under Kleene star.*

The table below summarizes the closure and non-closure properties of the language classes that are accepted by the various types of stateless CD-R(1)-systems, where $\cup$ denotes union, $\cap_{REG}$ denotes intersection with a regular language, ^{c} denotes complementation, $\cdot$ denotes product, * denotes Kleene star, h denotes ε-free morphisms, h^{-1} denotes inverse morphisms, *com* denotes commutative closure, and ^{R} denotes reversal:

Types of CD-Systems	**Operations**								
	$\cup$	$\cap_{REG}$	^{c}	$\cdot$	*	h	h^{-1}	*com*	^{R}
stl-det-local-CD-R(1)	+	−	−	+	+	−	?	+	?
stl-det-global-CD-R(1)	−	−	+	−	−	−	?	−	−
stl-det-strict-CD-R(1)	−	−	+	−	−	−	−	−	−

Here "+" denotes the fact that the corresponding class is closed under the given operation, "−" denotes the fact that it is not closed, and "?" indicates that the status of this property is still open.

4 Decision Problems

As stl-det-global-CD-R(1)-systems are a special type of stl-det-local-CD-R(1)-systems, and as the language class $\mathcal{L}_{=1}$(stl-det-global-CD-R(1)) is closed under complementation, we obtain the following decidability results from [7].

Corollary 7. *The membership problem, the emptiness problem, the universe problem, and the finiteness problem are effectively decidable for* stl-det-global-CD-R(1)*-systems.*

The universe problem is undecidable for stl-det-local-CD-R(1)-systems, and the regularity, inclusion and equivalence problems are shown to be undecidable for

stl-det-local-CD-R(1)-systems by a reduction from the universe problem [7]. Obviously, this proof does not carry over to stl-det-global-CD-R(1)-systems. Accordingly we have to find a new approach for establishing corresponding undecidability results for these systems. Below we begin this investigation by studying the following variant of the intersection emptiness problem:

Intersection With Regular Language Emptiness Problem:

Instance : A stl-det-global-CD-R(1)-system $\mathcal{M}$ and a finite-state acceptor A.
Question : Does $L_{=1}(\mathcal{M}) \cap L(A) = \emptyset$ hold?

Theorem 1. *The Intersection With Regular Language Emptiness Problem is undecidable for* stl-det-global-CD-R(1)*-systems.*

This result can be shown by a reduction from the *Post Correspondence Problem* (PCP). Based on this undecidability result we can prove that the following variant of the inclusion problem is undecidable.

Corollary 8. *The following inclusion problem is undecidable in general:*

Instance *: A* stl-det-global-CD-R(1)*-system $\mathcal{M}$ and a finite-state acceptor A.*
Question *: Does $L_{=1}(\mathcal{M}) \subseteq L(A)$ hold?*

Proof. Let $\mathcal{M}$ be a stl-det-global-CD-R(1)-system on Σ, and let A be a finite-state acceptor on Σ. From A we can construct a finite-state acceptor A^c for the language $\Sigma^* \setminus L(A)$. Then $L_{=1}(\mathcal{M}) \cap L(A) = \emptyset$ iff $L_{=1}(\mathcal{M}) \subseteq L(A^c)$. Thus, it follows from Theorem 1 that the above inclusion problem is undecidable. □

As each regular language is accepted by some stl-det-global-CD-R(1)-system, Corollary 8 yields the following undecidablility result.

Corollary 9. *The inclusion problem is undecidable for* stl-det-global-CD-R(1)*-systems.*

5 Concluding Remarks

We have studied two deterministic variants of the stl-det-local-CD-R(1)-systems: the stl-det-strict-CD-R(1)-systems and the stl-det-global-CD-R(1)-systems. The former type of system is quite weak, as it does not even accept all finite languages, while the latter type accepts all regular languages; however, it does not accept all rational trace languages. Thus, the three types of CD-systems of stateless deterministic R(1)-automata give a proper hierarchy of three levels.

We have investigated the closure properties of the language classes defined by the two types of CD-systems introduced in this paper. As it turned out, both classes are closed under complementation, but apart from that we could only establish non-closure properties. However, it remains open whether the language class $\mathcal{L}_{=1}$(stl-det-global-CD-R(1)) is closed under inverse morphisms.

Finally we have also considered decision problems for stl-det-global-CD-R(1)-systems. In contrast to the situation for stl-det-local-CD-R(1)-systems, the universe problem is decidable for stl-det-global-CD-R(1)-systems. On the other hand,

we could show that the inclusion problem is undecidable, but it remains open whether the regularity problem or the equivalence problem are decidable for these systems.

References

1. Berstel, J.: Transductions and Context-Free Languages. In: Leitfäden der angewandten Mathematik und Mechanik, vol. 38, Teubner Studienbücher, Teubner (1979)
2. Diekert, V., Rozenberg, G.: The Book of Traces. World Scientific, Singapore (1995)
3. Kutrib, M., Messerschmidt, H., Otto, F.: On stateless two-pushdown automata and restarting automata. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) Automata and Formal Languages, AFL 2008, Proc. Computer and Automation Research Institute, Hungarian Academy of Sciences, pp. 257–268 (2008)
4. Messerschmidt, H., Otto, F.: Cooperating distributed systems of restarting automata. International Journal of Foundations of Computer Science 18, 1333–1342 (2007)
5. Messerschmidt, H., Otto, F.: Strictly deterministic CD-systems of restarting automata. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 424–434. Springer, Heidelberg (2007)
6. Messerschmidt, H., Otto, F.: On deterministic CD-systems of restarting automata. International Journal of Foundations of Computer Science 20, 185–209 (2009)
7. Nagy, B., Otto, F.: CD-systems of stateless deterministic R-automata with window size one. Kasseler Informatikschriften 2/2010, Universität Kassel (April 2010), `https://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2010042732682`
8. Nagy, B., Otto, F.: CD-systems of stateless deterministic R(1)-automata accept all rational trace languages. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 463–474. Springer, Heidelberg (2010)
9. Nagy, B., Otto, F.: Finite-state acceptors with translucent letters. In: Bel-Enguix, G., Dahl, V., De La Puente, A. (eds.) BILC 2011: AI Methods for Interdisciplinary Research in Language and Biology. Proc., pp. 3–13. SciTePress, Portugal (2011)

Bit-coded Regular Expression Parsing*

Lasse Nielsen and Fritz Henglein

DIKU, University of Copenhagen, Denmark

Abstract. Regular expression parsing is the problem of producing a parse tree of a string for a given regular expression. We show that a compact bit representation of a parse tree can be produced efficiently, in time linear in the product of input string size and regular expression size, by simplifying the DFA-based parsing algorithm due to Dubé and Feeley to emit the bits of the bit representation without explicitly materializing the parse tree itself. We furthermore show that Frisch and Cardelli's greedy regular expression parsing algorithm can be straightforwardly modified to produce bit codings directly. We implement both solutions as well as a backtracking parser and perform benchmark experiments to gauge their practical performance. We observe that our DFA-based solution can be significantly more time and space efficient than the Frisch-Cardelli algorithm due to its sharing of DFA-nodes, but that the latter may still perform better on regular expressions that are "more deterministic" from the right than the left. (Backtracking is, unsurprisingly, quite hopeless.)

1 Introduction

A *regular expression* over finite alphabet Σ, as introduced by Kleene [12], is a formal expression generated by the regular tree grammar

$$E, F ::= 0 \mid 1 \mid a \mid E + F \mid E \times F \mid E^*$$

where $a \in \Sigma$, and *, $\times$ and $+$ have decreasing precedence. (In concrete syntax, we may omit $\times$ and write $\mid$ instead of $+$.) Informally, we talk about a regular expression *matching* a string, but what exactly does that mean?

In classical theoretical computer science, regular expression matching is the problem of *deciding* whether a string belongs to the regular *language* denoted by a regular expression; that is, it is *membership testing* [1]. In this sense, *abdabc* matches `((ab)(c|d)|(abc))*`, but *abdabb* does not. This is captured in the *language interpretation* for regular expressions in Figure 1.

In *programming*, however, membership testing is rarely good enough: We do not only want a yes/no answer, we also want to obtain proper *matches* of substrings against the subexpressions of a regular expression so as to *extract* parts of the input for further processing. In a *Perl Compatible Regular Expression (PCRE)*[1]

* This work has been partially supported by the Danish Strategic Research Council under Project "TrustCare". The order of authors is insignificant.

[1] See `http://www.pcre.org`

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 402–413, 2011.

matcher, for example, matching *abdabc* against $E = \texttt{((ab)(c|d)|(abc))*}$ yields a substring match for each of the 4 parenthesized subexpressions ("groups"): They match *abc*, *ab*, *c*, and ϵ (the empty string), respectively. If we use a POSIX matcher [10] instead, we get *abc*, ϵ, ϵ, *abc*, however. The reason for the difference is that `((ab)(c|d)|(abc))*` is *ambiguous*: the string *abc* can match the left or the right alternative of `(ab)(c|d)|(abc)`, and returning substring matches makes this difference observable.

A limitation of Perl- and POSIX-style matching is that we only get at most one match for each group in a regular expression. This is why only *abc* is returned, the *last* substring of *abdabc* matching the group `((ab)(c|d)|(abc))` in the regular expression above. Intuitively, we might expect to get the *list* of all matches [*abd*, *abc*]. This is possible with *regular expression types* [9]: Each group in a regular expression can be named by a variable, and the output may contain multiple matches for the same variable. For a variable under *two* Kleene stars, however, we cannot discern the matches belonging to different level-1 Kleene-star groups.

An even more refined notion of matching is thus *regular expression parsing*: Returning a parse tree of the input string under the regular expression read as a *grammar*. Automata-theoretic techniques, which exploit equivalence of regular expressions under their language interpretation, typically change the grammatical structure of matched strings and are thus not directly applicable. Only recently have linear-time[2] regular expression *parsing* algorithms been devised [6,7].

In this paper we show how to generate a compact *bit-coded* representation of a parse tree highly efficiently, without explicitly constructing the parse tree first. A bit coding can be thought of as an oracle directing the expansion of a grammar—here we only consider regular expressions—to a particular string. Bit codings are interesting in their own right since they are typically not only smaller than the parse tree, but also smaller than the string being parsed and can be combined with other techniques for improved text compression [4,3].

In Section 2 we recall that parse trees can be identified with the elements of regular expressions interpreted as types, and in Section 3 we describe bit codings and conversions to and from parse trees. Section 4 presents our algorithms for generating bit coded parse trees. These are empirically evaluated in Section 5. Section 6 summarizes our conclusions.

2 Regular Expressions as Types

Parse trees for regular expressions can be formalized as ad-hoc data structures [6,2], representing exactly how the string can be expressed in the regular expression. This means that both membership testing, substring groups and regular expression types can be found by filtering away the extra information in the parse tree. Interestingly, parse trees also arise completely naturally by interpreting regular expressions as *types* [7,8]; see Figure 2. For example, the

[2] This is the data complexity; that is for a fixed regular expression, whose size is considered constant.

$$\mathcal{L}[\![0]\!] = \emptyset \qquad \mathcal{L}[\![1]\!] = \{\epsilon\} \qquad \mathcal{L}[\![a]\!] = \{a\}$$
$$\mathcal{L}[\![E + F]\!] = \mathcal{L}[\![E]\!] \cup \mathcal{L}[\![F]\!] \qquad \mathcal{L}[\![E \times F]\!] = \mathcal{L}[\![E]\!]\,\mathcal{L}[\![F]\!] \qquad \mathcal{L}[\![E^*]\!] = \{\bigcup_{i \geq 0} \mathcal{L}[\![E]\!]^i$$

where ϵ is the empty string, $S\,T = \{s\,t \mid s \in S \wedge t \in T\}$, and $S^0 = \{\epsilon\}, S^{i+1} = S\,S^i$.

Fig. 1. The language interpreation of regular expressions

$$\mathcal{T}[\![0]\!] = \emptyset \qquad \mathcal{T}[\![1]\!] = \{()\} \qquad \mathcal{T}[\![a]\!] = \{a\}$$
$$\mathcal{T}[\![E + F]\!] = \mathcal{T}[\![E]\!] + \mathcal{T}[\![F]\!] \qquad \mathcal{T}[\![E \times F]\!] = \mathcal{T}[\![E]\!] \times \mathcal{T}[\![F]\!] \qquad \mathcal{T}[\![E^*]\!] = \mathcal{T}[\![E]\!]\,\mathrm{list}$$

where () is distinct from all alphabet symbols, $S + T = \{\mathrm{inl}\,x \mid x \in S\} \cup \{\mathrm{inr}\,y \mid y \in T\}$ is the disjoint union, $S \times T = \{(x, y) \mid x \in S, y \in T\}$ the Cartesian product of S and T, and $S\,\mathrm{list} = \{[v_1, \ldots, v_n] \mid v_i \in S\}$ the finite lists over S.

Fig. 2. The type interpreation of regular expressions

type interpretation of regular expression $((\mathtt{ab})(\mathtt{c}|\mathtt{d})|(\mathtt{abc}))*$ is $((\{a\} \times \{b\}) \times (\{c\} + \{d\}) + \{a\} \times (\{b\} \times \{c\}))\,\mathrm{list}$. We call elements of a type *values*; e.g., $p_1 = [\mathrm{inl}\,((a, b), \mathrm{inr}\,d), \mathrm{inr}\,(a, (b, c))]$ and $p_2 = [\mathrm{inl}\,((a, b), \mathrm{inr}\,d), \mathrm{inl}\,((a, b), \mathrm{inl}\,c)]$ are different elements of $((\{a\} \times \{b\}) \times (\{c\} + \{d\}) + \{a\} \times (\{b\} \times \{c\}))\,\mathrm{list}$ and thus represent different parse trees for regular expression $((\mathtt{ab})(\mathtt{c}|\mathtt{d})|(\mathtt{abc}))*$.

We can *flatten* (unparse) any value to a string by removing the tree structure.

Definition 2.1. *The* flattening *function* flat(.) *from values to strings is defined as follows:*

$$\mathrm{flat}(()) = \epsilon \qquad \mathrm{flat}(a) = a$$
$$\mathrm{flat}(\mathrm{inl}\,v) = \mathrm{flat}(v) \qquad \mathrm{flat}(\mathrm{inr}\,w) = \mathrm{flat}(w)$$
$$\mathrm{flat}((v, w)) = \mathrm{flat}(v)\,\mathrm{flat}(w) \qquad \mathrm{flat}(\mathrm{fold}\,v) = \mathrm{flat}(v)$$

Flattening the type interpretation of a regular expression yields its language interpretation:

Theorem 2.1. $\mathcal{L}[\![E]\!] = \{\mathrm{flat}(v) \mid v \in \mathcal{T}[\![E]\!]\}$

A regular expression is *ambiguous* if and only if its type interpretation contains two distinct values that flatten to the same string. With p_1, p_2 as above, since $\mathrm{flat}(p_1) = \mathrm{flat}(p_2) = abdabc$, this shows that $((ab)(c|d)|(abc))^*$ is grammatically ambiguous.

3 Bit-coded Parse Trees

The description of bit coding in this section is an adaptation from Henglein and Nielsen [8]. Bit coding for more general types than the type interpretation of regular expressions is well-known [11]. It has been applied to certain context-free grammars [3], but its use in this paper for efficient regular expression parsing seems to be new.

$$
\begin{array}{ll}
\mathrm{code}((): 1) & = \epsilon \\
\mathrm{code}(a : a) & = \epsilon \\
\mathrm{code}(\mathrm{inl}\, v : E + F) & = 0\ \mathrm{code}(v : E) \\
\mathrm{code}(\mathrm{inr}\, w : E + F) & = 1\ \mathrm{code}(w : F) \\
\mathrm{code}((v, w) : E \times F) & = \mathrm{code}(v : E)\ \mathrm{code}(w : F) \\
\mathrm{code}([v_1, \ldots, v_n] : E^*) & = 0\ \mathrm{code}(v_1 : E) \ldots\ 0\ \mathrm{code}(v_n : E)\ 1
\end{array}
$$

Fig. 3. Type-directed encoding function from syntax trees to bit sequences

$$
\begin{array}{ll}
\mathrm{decode}'(d : 1) & = ((), d) \\
\mathrm{decode}'(d : a) & = (a, d) \\
\mathrm{decode}'(0d : E + F) & = \mathbf{let}\ (v, d') = \mathrm{decode}'(d : E) \\
 & \quad \mathbf{in}\ (\mathrm{inl}\, v, d') \\
\mathrm{decode}'(1d : E + F) & = \mathbf{let}\ (w, d') = \mathrm{decode}'(d : F) \\
 & \quad \mathbf{in}\ (\mathrm{inr}\, w, d') \\
\mathrm{decode}'(d : E \times F) & = \mathbf{let}\ (v, d') = \mathrm{decode}'(d : E) \\
 & \qquad (w, d'') = \mathrm{decode}'(d' : F) \\
 & \quad \mathbf{in}\ ((v, w), d'') \\
\mathrm{decode}'(0d : E^*) & = \mathbf{let}\ (v_1, d') = \mathrm{decode}'(d : E) \\
 & \qquad (\vec{v}, d'') = \mathrm{decode}'(d' : E^*) \\
 & \quad \mathbf{in}\ (v_1 :: \vec{v}, d') \\
\mathrm{decode}'(1d : E^*) & = ([], d) \\
\mathrm{decode}(d : E) = & = \mathbf{let}\ (w, d') = \mathrm{decode}'(d : E) \\
 & \quad \mathbf{in}\ \mathbf{if}\ d' = \epsilon\ \mathbf{then}\ w\ \mathbf{else}\ \mathit{error}
\end{array}
$$

Fig. 4. Type-directed decoding function from bit sequences to syntax trees

A *bit-coded parse tree* is a bit sequence representing a parse tree for a *given* regular expression. Intuitively, bit coding factors a parse tree p into its static part, the regular expression E it is a member of, and its dynamic part, a bit sequence that uniquely identifies p as a particular element of E. The basic idea is that the bit sequence serves as an oracle for the alternatives that must be taken to expand a regular expression into a particular string.

Consider, for example, the values $p_1 = [\mathrm{inl}\,((a, b), \mathrm{inr}\, d), \mathrm{inr}\,(a, (b, c))]$ and $p_2 = [\mathrm{inl}\,((a, b), \mathrm{inr}\, d), \mathrm{inl}\,((a, b), \mathrm{inl}\, c)]$ from Section 2, which represent distinct parse trees of *abdabc* for regular expression `((ab)(c|d)|(abc))*`. The bit coding arises from throwing away everything in the parse tree except the list and the tag constructors, which yields $[\mathrm{inl}\,\mathrm{inr}\,, \mathrm{inr}\,]$. We code inl by 0 and inr by 1, which gives us $[01, 1]$. Finally we code the list itself: Each element is prefixed by 0, and the list is terminated by 1. The resulting bit coding is $b_1 = 001\,01\,1$ (whitespace added for readability). Similarly, the bit coding of p_2 is $b_2 = 001\,000\,1$. More compact codings for lists are possible by generalizing regular expressions to tail-recursive μ-terms [8]. We stick to the given coding of lists here, however, since the focus of this paper is on constructing the bit codings, not their effect on text compression.

Figures 3 and 4 define regular-expression directed linear-time coding and decoding functions from parse trees to their bit codings and back:

Theorem 3.1. *If* $v \in \mathcal{T}[\![E]\!]$ *then* $\mathrm{decode}(\mathrm{code}(v : E) : E) = v$

Proof: By structural induction on v.

Note that bit codings are only meaningful in the context of a regular expression, because the same bit sequence may represent different strings for different regular expressions, and may be invalid for other regular expressions.

Bit codings are not only more compact than parse trees. As we shall see, they are also more suitable for automaton output, as it is not necessary to generate list structure, pairing or even the alphabet symbols occurring in a parse tree.

4 Parsing Algorithms

We present two bit coding parsing algorithms in this section. The first can be understood as a simplification of Dubé and Feeley's [6] DFA-generation algorithm, producing bit codings instead of explicit parse tree. We also show that Frisch and Cardelli's [7] algorithm can be straightforwardly modified to produce bit codings.

4.1 Dubé/Feeley-Style Parsing

Our algorithm DFA performs the following steps: Given regular expression E and input string s,

1. generate an enhanced Thompson-style NFA with output actions (finite state transducer);
2. use the subset construction to produce an enhanced DFA with additional information on edges to capture the output actions from the NFA;
3. use the enhanced DFA as a regular DFA on s;
   - if it rejects terminate with error (no parse tree);
   - if it accepts, return the path in the DFA induced by the input string;
4. combine, in reverse order of the path, the output information on each edge traversed to construct the bit coding of a parse tree.

The steps are described in more detail below.

Enhanced NFA generation. The left column of Figure 5 shows Thompson-style NFA generation. We enhance it by adding single bit outputs to the outedges of those states that have two outgoing ϵ-transitions, shown in the right column. The output bits can be thought of indicators for an agent traversing the NFA: 0 means turn left, 1 means turn right. The other edges carry no output bit since their traversal is forced.

When traversing a path p in an enhanced NFA, the sequence of symbols read is denoted by $\mathrm{read}(p)$, and the sequence of symbols written is denoted by $\mathrm{write}(p)$.

Lemma 4.1 (Soundness and completeness of enhanced NFAs). *Let* N_E *be the extended NFA for regular expression* E *according to Figure 5 (right). Then for each* $s \in \Sigma^*$ *we have* $\{v | v \in \mathcal{T}[\![E]\!] \wedge \mathrm{flat}(v) = s\} = \{\mathrm{decode}(\mathrm{write}(p) : E) \mid p$ *is a path in* N_E *from initial state to final state such that* $\mathrm{read}(p) = s\}$.

PROOF: By structural induction on E.

In other words, an extended NFA generates exactly the bit codings of the parse trees of the strings it accepts. Observe furthermore that no two distinct paths from initial to final state have the same output bits. This means that a bit coding uniquely determines a particular path from initial to final state, and vice versa. Dubé and Feeley [6] also instrument Thompson-style NFAs, but with more output symbols on more edges so as to be able to generate an external representation of a parse tree. Figure 6 shows their enhanced NFA for $E = a \times (b + c)^\star \times a$ on the left. Our corresponding enhanced NFA is shown on the right.

Enhanced subset construction. During subset construction additional information is computed:

1. A map init from the NFA states q in the initial DFA state to an output $\text{init}(q)$. The output $\text{init}(q)$ must be $\text{write}(p)$ for some path p from the initial NFA state to q where $\text{read}(p) = \varepsilon$. These paths are traversed when finding the ε-closure of the initial NFA state, which is how the initial DFA state is constructed in the subset construction, and is thus easy to generate.
2. A map output_e for each edge e in the DFA, that maps each NFA state q_2 in the destination DFA state to a pair (q_1, o) of an NFA state q_1 in the source DFA state, and an output o. The output o must be $\text{write}(p)$ for some path p from q_1 to q_2 where $\text{read}(p)$ is the input of the DFA edge e. These paths are traversed, when the destination state of the edge is computed, and are thus simple to generate.

The DFA for the NFA from Fig. 6 (right) is shown in Fig. 7, and the result of adding the information described above is shown in Fig. 8.

The additional information captures basically the same information as in Dubé and Feeley's DFA construction, but it stores the additional information directly in the DFA edges, where Dubé and Feeley use an external 3-dimensional table. Most importantly, the additional information we need to store is reduced, since we only generate bit codings, not explicit parse trees.

Bit code construction. After accepting $s = a_1 \ldots a_n$ we have a path $p = [A_0, A_1, \cdots, A_n]$ in the extended DFA, where $A_0, \ldots, A_n$ are DFA states, each consisting of a set of NFA states, with A_0 containing the initial NFA state q_i and A_n the final NFA state q_f.

We construct the bit code of a parse tree for s by calling $\text{write}(p, q_f)$ where write traverses p from right to left as follows:

$$\begin{aligned} \text{write}([A_0], q) &= \text{init}_q \\ \text{write}([A_0, A_1, \ldots, A_{k-1}, A_k], q) &= \text{write}([A_0, A_1, \ldots, A_{k-1}], q') \cdot b' \\ &\quad \text{where } (q', b') = \text{output}_{A_{k-1} \to A_k}(q) \end{aligned}$$

Lemma 4.2 (Bit coding preservation). *Let D_E be the extended DFA generated from the extended NFA N_E for E. If p is a path from the initial state in D*

to a state containing the NFA state q and if $\mathrm{write}(p, q) = b$ *then there is a path* p' *in* N_E *from the initial state in* N_E *to the state* q *such that* $\mathrm{read}(p) = \mathrm{read}(p')$ *and* $\mathrm{write}(p, q) = \mathrm{write}(p')$.

Proof: Induction on the number of steps in p.

We can now conclude that the bit sequence found represents a parse tree for the input string.

Theorem 4.1 (Correctness of DFA algorithm). *If* D_E *is an extended DFA generated from an extended NFA* N_E *for regular expression* E*, and* p *is a path from the initial state to a final state in* D_E*, and* q_f *is the final state of* N_E *then* $\mathrm{read}(p) = \mathrm{flat}(\mathrm{decode}(\mathrm{write}(p, q_f) : E))$.

Proof: This follows from first applying Lemma 4.2 and then Lemma 4.1.

Once the DFA has been generated, this method results in very efficient regular expression parsing. The DFA traversal takes time $\Theta(|s|)$, and the bit code generation takes time $\Theta(|s| + |b|)$, where $|b|$ is the length of the output bit sequence. This means the total run time complexity of parsing is $\Theta(|s| + |b|)$ which is (sequentially) optimal, since the entire string must be read, and the entire bit sequence must be written.

Example. Consider the extended DFA for the regular expression $a(b + c)^\star a$ in Fig. 8. If we use it to accept the string *abcba*, then we get the path $p = 0 \to 1 \to 3 \to 4 \to 3 \to 2$. Tracing the path backwards, keeping track of the output bit-sequences b and NFA states q we get the steps in the table on the right. Since $\mathrm{init}_0 =$ "" we get the bit code $b =$ "" $\cdot$ "00" $\cdot$ "01" $\cdot$ "00" $\cdot$ "1" $\cdot$ "" = "0001001". We can verify that the DFA parsing algorithm has given us a correct bit coding since $\mathrm{flat}(\mathrm{decode}(\text{"0001001"} : a(b + c)^\star a)) = abcba$.

DFA steps				
Step	q	Symbol	new q	b
$3 \to 2$	9	a	8	""
$4 \to 3$	8	b	3	"1"
$3 \to 4$	3	c	4	"00"
$1 \to 3$	4	b	3	"01"
$0 \to 1$	3	a	0	"00"

4.2 Frisch/Cardelli-Style Parsing

Instead of building a Thompson-style NFA from the regular expressions, Frisch and Cardelli [7] build an NFA with one node for each position in the regular expression, with the final state as the only additional node. The regular expression positions are identified by the path used to reach the position. The positions $\lambda_{\mathbf{end}}(E)$ in a regular expression E are defined as lists of choices (the choices are **fst** and **snd** for sequence, **lft** and **rgt** for sum and **star** for $\star$). $E.l$ is used to denote the subexpression of E found by following the path l. The transitions $\delta(E)$ in the NFA of a regular expression E are defined using a successor relation **succ** on the paths.

The NFA is used to generate a table $Q(l, i)$, which maps each position $l \in \lambda_{\mathbf{end}}(E)$ and input string position i to `true`, if l accepts the ith suffix of s and `false` otherwise.

The table Q can be constructed by starting with $Q(l,i) = \texttt{false}$ for all $l \in \lambda_{\mathbf{end}}(E)$ and $i = 1 \ldots |s|$, and calling $\mathbf{SetPrefix}(Q, \mathbf{end}, |s|)$, which updates Q as defined below.

$$\begin{array}{l}\mathbf{SetPrefix}(Q,l,i) = \text{if } Q(l,i) \text{ then return;}\\ \qquad\qquad\qquad\quad Q(l,i) := \texttt{true};\\ \qquad\qquad\qquad\quad \text{for } (l',\varepsilon,l) \in \delta(E) \text{ do } \mathbf{SetPrefix}(Q,l',i);\\ \qquad\qquad\qquad\quad \text{if } i > 0 \text{ then for } (l',s[i],l) \in \delta(E) \text{ do } \mathbf{SetPrefix}(Q,l',i-1);\end{array}$$

The run time cost and memory consumption of computing Q is asymptotically bounded by the size of Q, which is in $\Theta(|s| \cdot |E|)$.

After Q is built, it is easy to check whether s matches the regular expression E, simply by looking up $Q([], 0)$. We modify Frisch and Cardelli's build function to construct a bit coding representing the greedy leftmost (or first and greedy [14]) parse tree for s, if s matches E, as follows (notice that $l :: x$ means appending x after l):

$$\begin{array}{l}\mathbf{build}(l,i) = \text{case } E.l \text{ of}\\ \qquad\qquad a\text{: return } (\varepsilon, i+1)\\ \qquad\qquad 1\text{: return } (\varepsilon, i)\\ \qquad\qquad E_1 \times E_2\text{: let } (b_1,j) = \mathbf{build}(l :: \mathbf{fst}, i)\\ \qquad\qquad\qquad\qquad \text{in let } (b_2,k) = \mathbf{build}(l :: \mathbf{snd}, j) \text{ in return } (b_1 b_2, k);\\ \qquad\qquad E_1 + E_2\text{: if } Q(l :: \mathbf{lft}, i)\\ \qquad\qquad\qquad\qquad \text{then let } (b_1,j) = \mathbf{build}(l :: \mathbf{fst}, i) \text{ in retuen } (0b_1, j)\\ \qquad\qquad\qquad\qquad \text{else let } (b_2,j) = \mathbf{build}(l :: \mathbf{snd}, i) \text{ in return } (1b_2, j);\\ \qquad\qquad E_1^\star\text{: if } Q(l :: \mathbf{star}, i)\\ \qquad\qquad\qquad \text{then let } (b_1,j) = \mathbf{build}(l :: \mathbf{star}, i)\\ \qquad\qquad\qquad\qquad \text{in let } (b_2,k) = \mathbf{build}(l, j) \text{ in return } (0b1b2, k)\\ \qquad\qquad\qquad \text{else return } (1, j);\end{array}$$

The run time cost and memory consumption of $\mathbf{build}([], 0)$ is asymptotically bounded by the number of cells in Q, which is in $\Theta(|E| \cdot |s|)$ and which is therefore the time complexity and memory consumption of the entire algorithm.

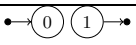
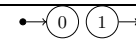
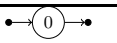
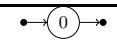
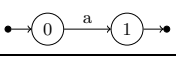
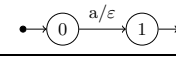
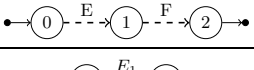
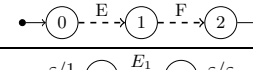
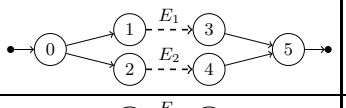
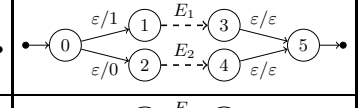
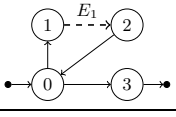
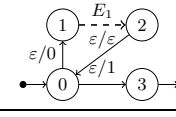
E	NFA	Extended NFA
0		
1		
a		
$E_1 \times E_2$		
$E + F$		
$E_1{}^\star$		

Fig. 5. NFA generation schema

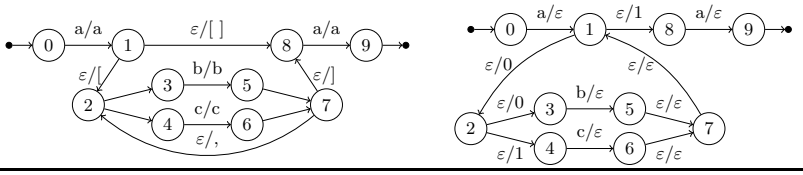


Fig. 6. Extended NFAs for $a(b+c)^\star a$

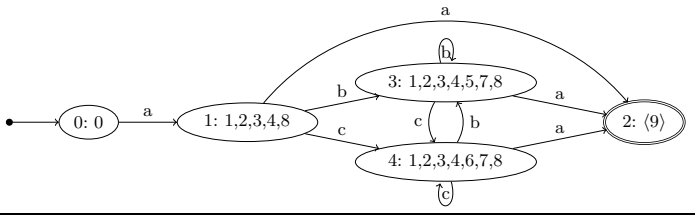


Fig. 7. DFA for $a(b+c)^\star a$

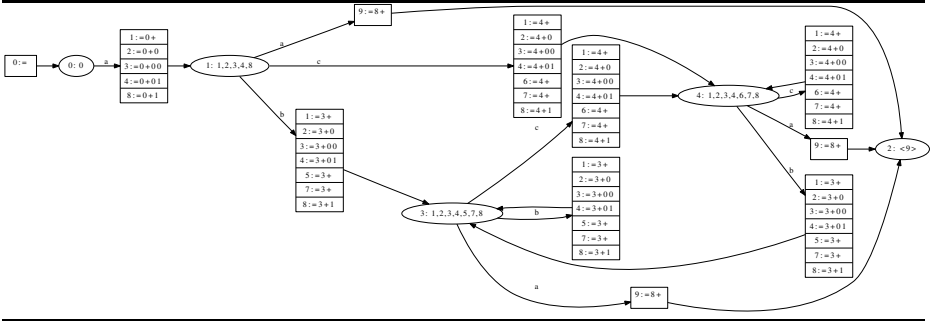


Fig. 8. Extended DFA for $a(b+c)^\star a$

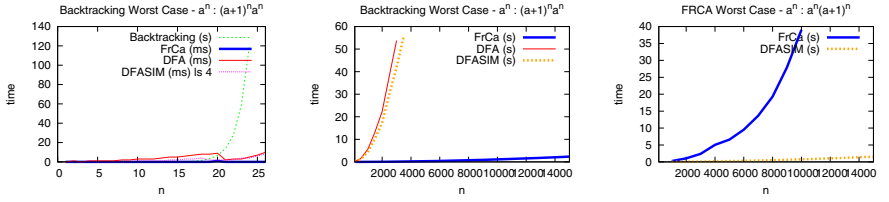
5 Empirical Evaluation

We have implemented the algorithms described in Section 4 as a `C++` library [13] and performed a series of performance tests on a PC with a 2.50GHz Intel Core2 Duo CPU and 4Gb of memory, running Ubuntu 10.4. We test four different parsing methods. NFA based backtracking (backtracking), implemented by a depth-first search for an accepting path in our enhanced Thompson-style NFA. FRCA is the algorithm based on Frisch and Cardelli from Section 4.2. DFA is the algorithm based on Dubé and Feeley from Section 4.1. DFASIM is the same algorithm as DFA, but where the nodes and edges of the DFA are not precomputed, but generated dynamically by need.

5.1 Backtracking Worst Case: $(\mathtt{a}^n : (a+1)^n a^n)$

The regular expression is $(a+1)^n a^n$, where we use the notation E^n to represent $E \times \cdots \times E$ (n copies). This is a well-known example [5], which captures the

problematic cases for backtracking.[3] The results of matching $\mathtt{a}^n$ (denoting n as) to $(a+1)^n \times a^n$ are in the two leftmost graphs below.

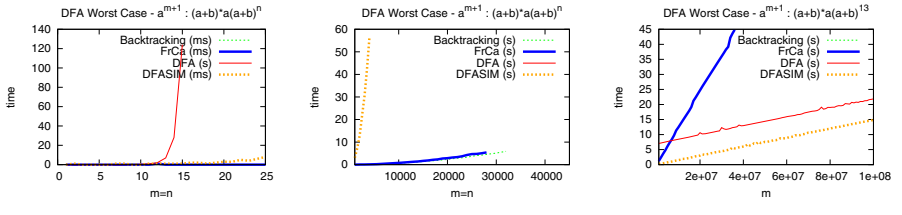


When parsing a string with n as, the backtracking algorithm traverses 2^n different paths before eventually finding the match. The cost of generating the DFA is $\Theta(n^2)$ ($2 \cdot n$ nodes containing n NFA-nodes on average). Since only half of the DFA-nodes are used, DFASIM is faster than generating the whole DFA, but the run time complexity is still $\Theta(n^2)$. The time used for FRCA is $\Theta(n)$, since there is exactly one suffix that can be parsed from each position in the regular expression. The reason FRCA performs much better than the other algorithms in this example is that the example was designed to be hard to parse from the left to right, and FRCA processes the string in its first phase from right to left. If we change the regular expression to $a^n(1+a)^n$, it becomes hard to process from right to left, as shown in the rightmost graph above. It is generally advantageous to process a string in the "more deterministic" direction of the regular expression.

5.2 DFA Worst Case ($\mathtt{a}^{m+1} : (a+b)^\star a(a+b)^n$)

The following is a worst-case scenario for the DFA based algorithm, and a best-case scenario for the FRCA and backtracking algorithms. The regular expression is $(a+b)^\star a(a+b)^n$, and the string is $\mathtt{a}^{m+1}$.

The two leftmost graphs below show the execution time when $n = m$, and the right graph shows the execution time when $n = 13$. When n is fixed to 13, the runtime of both FRCA, DFASIM and DFA are linear, but even though DFA has a large initialization time for building the DFA, FRCA uses more time for large m, because it uses more time per character.



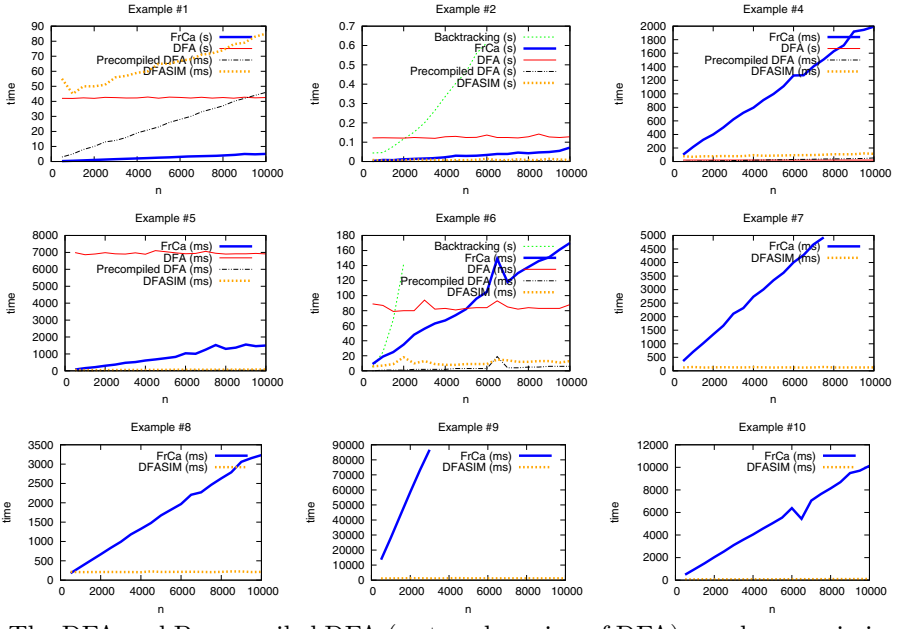
The DFA will have $\mathcal{O}(2^n)$ DFA-nodes. This causes the DFA algorithm to have a run time complexity of $\Theta(m \cdot 2^n)$. This exponential explosion is avoided by DFASIM, which only builds as many states as needed for the particular input string.

[3] If a fixed regular expression is preferred, then $(a+a)^\star \times b$ or $(a^\star \times a)^\star \times b$ provokes the same behavior.

5.3 Practical Examples

We have tested 9 of the 10 examples of "real world" regular expressions from Veanes et al. [15] to compare the performance of each algorithm. (Their example nr. 3 is uninteresting for performance testing since it only accepts strings of a bounded length).

Examples 1,4,5,7,8,9,10 are different ways of expressing the language of email addresses, while Example 2 defines the language of dollar-amounts, and Example 6 defines the language of floating point values.



The DFA and Precompiled DFA (a staged version of DFA) graphs are missing in many of the examples. This is because the DFA generation runs out of memory. We may conclude that there are many cases where it is not feasible to generate the full DFA. The two best algorithms for these tests are FRCA and DFASIM, with DFASIM being faster by a large factor (at least 10) in all cases. Apart from the direction of processing NFA-nodes in their respective first passes, the key difference between DFASIM and FRCA is that DFASIM memoizes and reuses DFA-states at multiple positions in the input string, whereas FRCA essentially produces what amounts to a separate DFA-state for each position in the input string. In comparison to FRCA, the DFA-state memoization not only saves space, but also computation time whenever the same transition is traversed more than once.

6 Conclusion

We have designed and implemented a number of regular expression parsing algorithms, which produce bit coded representations of parse trees without ever materializing the parse trees during parsing. Producing bit codings is advantageous since it carries the dual advantage of yielding a compressed parse tree

representation and of speeding its construction. Our DFA simulation algorithm DFASIM, in the style of Dubé and Feeley [6], and FRCA, a modified version of the greedy algorithm of Frisch and Cardelli [7], have shown the best asymptotic performance, with DFA simulation beating FRCA on a suite of real world examples. As for the potential for further improvements, compact NFA-construction, efficient computation of the sub-NFA induced by an input string (left-to-right or right-to-left or, preferably, something better), and memoized DFA-state construction appear to be key to obtaining practically improved regular expression parsing without sacrificing asymptotic scalability.

References

1. Bille, P., Thorup, M.: Faster regular expression matching. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 171–182. Springer, Heidelberg (2009)
2. Brabrand, C., Thomsen, J.: Typed and unambiguous pattern matching on strings using regular expressions. In: Proc. 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming, PPDP (2010)
3. Cameron, R.: Source encoding using syntactic information source models. IEEE Transactions on Information Theory 34(4), 843–850 (1988)
4. Contla, J.: Compact coding of syntactically correct source programs. Software: Practice and Experience 15(7), 625–636 (1985)
5. Cox, R.: Regular expression matching can be simple and fast
6. Dubé, D., Feeley, M.: Efficiently building a parse tree from a regular expression. Acta Informatica 37(2), 121–144 (2000)
7. Frisch, A., Cardelli, L.: Greedy regular expression matching. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 618–629. Springer, Heidelberg (2004)
8. Henglein, F., Nielsen, L.: Declarative coinductive axiomatization of regular expression containment and its computational interpretation (preliminary version). In: Proc. 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL) (January 2011)
9. Hosoya, H., Vouillon, J., Pierce, B.C.: Regular expression types for xml. ACM Trans. Program. Lang. Syst. 27(1), 46–90 (2005)
10. Institute of Electrical and Electronics Engineers (IEEE): Standard for information technology — Portable Operating System Interface (POSIX) — Part 2 (Shell and utilities), Section 2.8 (Regular expression notation), New York, IEEE Standard 1003.2 (1992)
11. Jansson, P., Jeuring, J.: Polytypic compact printing and parsing. In: Swierstra, S.D. (ed.) ESOP 1999. LNCS, vol. 1576, p. 639. Springer, Heidelberg (1999)
12. Kleene, S.C.: Representation of events in nerve nets and finite automata. Automata Studies 34, 3–41 (1956)
13. Nielsen, L.: Regular expression compression parser, `http://www.thelas.dk/index.php/Rcp`
14. Vansummeren, S.: Type inference for unique pattern matching. ACM Trans. Program. Lang. Syst. 28(3), 389–428 (2006)
15. Veanes, M.V.M., de Halleux, P., Tillmann, N.: Rex: Symbolic regular expression explorer. In: Proc. 3d Int'l Conf. on Software Testing, Verification and Validation, April 6-10. IEEE Computer Society Press, Paris (2010)

Descriptional Complexity of Unambiguous Nested Word Automata

Alexander Okhotin[1,2] and Kai Salomaa[3]

[1] Department of Mathematics, University of Turku, 20014 Turku, Finland
alexander.okhotin@utu.fi
[2] Academy of Finland
[3] School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Abstract. It is known that converting an n-state nondeterministic nested word automaton (a.k.a. input-driven automaton; a.k.a. visibly pushdown automaton) to a corresponding deterministic automaton requires in the worst case $2^{\Theta(n^2)}$ states (R. Alur, P. Madhusudan: Adding nesting structure to words, DLT'06). We show that the same worst case $2^{\Theta(n^2)}$ size blow-up occurs when converting a nondeterministic nested word automaton to an unambiguous one, and an unambiguous nested word automaton to a deterministic one. In addition, the methods developed in this paper are used to demonstrate that the state complexity of complementation for nondeterministic nested word automata is $2^{\Theta(n^2)}$, and that the state complexity of homomorphism for deterministic nested word automata is $2^{\Theta(n^2)}$.

1 Introduction

Nested words provide a natural computational model for applications like XML document processing, where the data has a dual linear-hierarchical structure [1,4,5]. Finite automata operating on nested words, called *nested word automata*, were introduced by Alur and Madhusudan [3,4]. This model is equivalent to *input-driven pushdown automata* [23,7] operating on ordinary strings without a hierarchical structure, which have recently been actively studied under yet another name of *visibly pushdown automata* [2,4,8,26]. Finite tree automata [9] on unranked (or ranked) trees can be viewed as special cases of nested word automata and the latter model is equivalent to pushdown forest automata [10,24]. The model of nested word automata has been recently extended to more general graph automata by Madhusudan and Parlato [22]. In the same way as an input-driven pushdown automaton can be simulated by a finite automaton operating on nested words, other types of machines with auxiliary storage can be realized by finite graph automata working on specialized graphs, and the structure of the graphs allows one to obtain decidability results for the corresponding auxiliary storage model [22].

The family of regular nested word languages recognized by finite nested word automata retains many of the desirable properties of ordinary regular languages.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 414–426, 2011.

In particular, a nondeterministic nested word automaton can be determinized, however, the state space blow-up is worse than in the case of ordinary finite automata. A deterministic nested word automaton equivalent to a nondeterministic automaton with $O(n)$ states needs in the worst case 2^{n^2} states [3]. Since nested word automata are a nontrivial generalization of ordinary finite automata, questions related to state complexity of conversions between different types of models are of interest. Recent work on state complexity of finite automata is presented in survey papers by Holzer and Kutrib [15,16].

Here we consider unambiguous nested word automata, that is, nondeterministic automata where each accepted word has a unique accepting computation. There has been much work done on unambiguous finite automata (UFA) and on automata employing different degrees of ambiguity [11,19,20,29,25].

We show that converting an n-state unambiguous nested word automaton to a deterministic one, or an n-state nondeterministic automaton to an unambiguous automaton requires in the worst case $2^{\Theta(n^2)}$ states, that is, in both cases the state space explosion is the same as for determinizing a nondeterministic nested word automaton.

To establish lower bounds on the size of nondeterministic nested word automata, we use fooling set methods [13,28], that have been originally introduced for proving lower bounds for (ordinary) nondeterministic finite automata (NFAs) [6,33]. Fooling set methods can be viewed as a special case of communication complexity arguments [17]. The only known general lower bound method for UFAs is based on the rank of a fooling set matrix; it was developed by E.M. Schmidt [32] already in 1978, see Leung [20] for a self-contained presentation. In this paper, we will extend this method to unambiguous nested word automata.

We consider also complementation of nondeterministic nested word automata. It is known that the complement of an n-state NFA needs in the worst case 2^n states [14,18]. In other recent work, the complementation of two-way finite automata was studied by Geffert et al. [12], and complementation of unary UFAs was investigated by Okhotin [25].

Operational state complexity of deterministic and nondeterministic nested word automata has been considered in [13,27,28,31]. In particular, Han and Salomaa [13] gave a lower bound of $\sqrt{n!}$ for the complement of nondeterministic nested word automata, leaving the precise state complexity of complementation open. As a modification of our lower bound construction for converting a nondeterministic nested word automaton into an unambiguous automaton, we show in Section 6 that in order to complement a nondeterministic nested word automaton, in the worst case, we have to essentially determinize the automaton and the state complexity of complementation of nondeterministic nested word automata is $2^{\Theta(n^2)}$. In the last Section 7, we give a tight state complexity bound for homomorphic images of deterministic nested word automata. The lower bound construction again relies on a modification of the nested word languages used for the lower bound for the conversion from a nondeterministic to an unambiguous nested word automaton.

This extended abstract omits many of the proofs.

2 Preliminaries

We assume that the reader is familiar with with the basics of formal languages and finite automata, see [30,33,34]. Here we briefly recall some definitions associated with automata operating on nested words. More details on nested words and their applications can be found in the works by Alur and Madhusudan [3,4].

In the following Σ denotes always a finite alphabet. The set of strings over Σ is Σ^*, and Σ^+ is the set of nonempty strings. Let $\Sigma^{\leqslant m}$ with $m \geqslant 0$ denote the set of all strings over Σ of length at most m. We refer to sequences of symbols as *strings,* while sequences of symbols associated with a hierarchical nesting structure (as defined below) will be called *nested words.*

For $m \in \mathbb{N}$ we denote $[m] = \{1, \ldots, m\}$. For every binary string $w = b_{\ell-1} \cdot b_1 b_0 \in \{0,1\}^*$, denote its numerical value by $(w)_2 = \sum_{i=0}^{\ell-1} b_i 2^i$.

The *tagged alphabet* corresponding to an alphabet Σ is $\hat{\Sigma} = \Sigma \cup \langle\Sigma \cup \Sigma\rangle$, where $\langle\Sigma = \{\langle a \mid a \in \Sigma\}$ is the set of *call symbols* and $\Sigma\rangle = \{a\rangle \mid a \in \Sigma\}$ is the set of *return symbols.* Elements of Σ are called *internal symbols.* A *tagged string* over Σ is a sequence of symbols of $\hat{\Sigma}$, $w = a_1 a_2 \cdots a_m$, with $a_i \in \hat{\Sigma}$ for $i = 1, 2, ..., m$. We define recursively a hierarchical matching relation in a tagged string. For w as above, a call symbol $a_i \in \langle\Sigma$ *matches* a return symbol $a_j \in \Sigma\rangle$, $i < j$, if in the subsequence $a_{i+1}a_{i+2} \cdots a_{j-1}$ every call symbol (respectively, return symbol) has a matching return symbol (respectively, call symbol). Symbol occurrences $a_i \in \langle\Sigma$ that do not have a matching return, $1 \leqslant i \leqslant m$, are *pending calls*, and $a_i \in \Sigma\rangle$ that does not have a matching call is a *pending return.* The above conditions define a unique matching relation between the call symbol occurrences and the return symbol occurrences in any tagged string.

By a *nested word* we mean a tagged string that is associated with the usual linear ordering of symbols and the hierarchical matching relation between occurrences of call and return symbols. The *underlying string* of a nested word is the corresponding tagged string without the hierarchical matching relation, that is, the underlying string is an ordinary string over the alphabet $\hat{\Sigma}$. When there should be no confusion, we sometimes refer to a nested word simply as a "word", for short. The set of nested words over Σ is denoted $\mathrm{NW}(\Sigma)$. A nested word language is any subset of $\mathrm{NW}(\Sigma)$.

A nested word is *well-matched* if every call symbol has a matching return and *vice versa.* An example of a nested word is $ab\rangle a\langle caa\langle dc\rangle ad\rangle ab\rangle a\langle b$. Here all occurrences of a are linear, the call-symbol $\langle c$ (respectively, $\langle d$) matches return symbol $d\rangle$ (respectively, $c\rangle$), both occurrences of $b\rangle$ are pending returns and $\langle b$ is a pending call.

Language operations are extended in the natural way to sets of nested words. Let $h : \hat{\Sigma} \to \mathrm{NW}(\Sigma)$ be a mapping such that for each $b \in \Sigma$, $h(b)$ is well-nested and for each $b \in \langle\Sigma$ (respectively, $b \in \Sigma\rangle$), $h(b)$ contains exactly one pending call and no pending returns (respectively, contains exactly one pending return and no pending calls). The mapping h determines a *homomorphism* $\overline{h} : \mathrm{NW}(\Sigma) \to \mathrm{NW}(\Sigma)$ by setting $\overline{h}(\varepsilon) = \varepsilon$, and for $b \in \hat{\Sigma}$, $w \in \mathrm{NW}(\Sigma)$ we define $\overline{h}(bw)$ as the unique nested word whose underlying string is $h(b)\overline{h}(w)$. When there is no confusion we use h in place of $\overline{h}$. We say that the homomorphism h is an *internal*

relabeling if h is identity on $\langle\Sigma$ and $\Sigma\rangle$, and maps internal symbols of Σ into Σ (not necessarily injectively).

The notion of "homomorphism that respects nesting" has been considered in [4]. These mappings are, in general, multiple-valued substitutions and require that in words of $h(\langle a)$ (respectively, $h(a\rangle)$) the unmatched call (respectively, unmatched return) has to occur as the first (respectively, the last) symbol. A one-valued homomorphism that respects nesting is a special case of the homomorphism defined above.

Next we recall the definition of nested word automata from Alur and Madhusudan [4]. This definition explicitly distinguishes the linear states that the computation passes following the linear ordering of the symbols and the hierarchical states that are passed from a call symbol to a matching return symbol. An earlier paper [3] used a simplified definition that does not make the distinction between different types of states.

Definition 2.1. *A* nondeterministic nested word automaton, *NNWA, is a tuple*

$$A = (\Sigma, Q, Q_0, Q_f, P, P_0, P_f, \delta_c, \delta_i, \delta_r),$$

where Σ is the input alphabet, Q is the finite set of linear states, $Q_0 \subseteq Q$ is the set of initial linear states, $Q_f \subseteq Q$ is the set of final linear states, P is the finite set of hierarchical states, $Q \cap P = \emptyset$, $P_0 \subseteq P$ is the set of initial hierarchical states, $P_f \subseteq P$ is the set of final hierarchical states, $\delta_c : Q \times \langle\Sigma \longrightarrow 2^{Q\times P}$ is the call transition function, $\delta_i : Q \times \Sigma \longrightarrow 2^Q$ is the internal transition function, and $\delta_r : Q \times P \times \Sigma\rangle \longrightarrow 2^Q$ is the return transition function.

Consider a nested word $w = u_1 \cdots u_m$, $u_i \in \hat{\Sigma}$, $i = 1, \ldots, m$ that has k occurrences of call symbols and let A be as in Definition 2.1. A *computation* of A on w consists of a sequence of linear states $q_i \in Q$, $i = 0, \ldots, m$, and a sequence of hierarchical states $p_j \in P$, $j = 1, \ldots, k$, corresponding to call symbol occurrences in w, such that $q_0 \in Q_0$, and for $i \in \{1, \ldots, m\}$, the following holds:

(i) If $u_i \in \Sigma$ is an internal symbol, then $q_i \in \delta_i(q_{i-1}, u_i)$.
(ii) If $u_i \in \langle\Sigma$ is the jth call symbol occurrence, $1 \leqslant j \leqslant k$, then $(q_i, p_j) \in \delta_c(q_{i-1}, u_i)$.
(iii) Assume that $u_i \in \Sigma\rangle$ is a return symbol occurrence. If u_i is matched with the j_ith call symbol occurrence, $1 \leqslant j_i \leqslant k$, then $q_i \in \delta_r(q_{i-1}, p_{j_i}, u_i)$. If u_i is a pending return, then $q_i \in \delta_r(q_{i-1}, p_0, u_i)$ for some $p_0 \in P_0$.

Intuitively, A begins a nondeterministic computation in some initial linear state $q_0 \in Q_0$. It reads an internal symbol using the internal transition function similarly as an ordinary NFA. When encountering a call symbol $\langle a$ in a linear state q, A sends along the linear edge a state $q' \in Q$ and along the hierarchical edge a state $p' \in P$ where $(q', p') \in \delta_c(q, \langle a)$ is nondeterministically chosen. When A encounters a return-symbol $a\rangle$ in a linear state state q and receives state $p \in P$ along the hierarchical edge, the computation continues in some linear state of $\delta_r(q, p, a\rangle)$. If $a\rangle$ is a pending return, A uses an arbitrary initial hierarchical state $p_0 \in P_0$ as the second argument for δ_r.

The *frontier* of a computation of A corresponding to a prefix w_1 of the input $w = w_1 \cdot w_2$ is a tuple $(p_1, \ldots, p_r, q)$, where $p_i \in P$, $i = 1, \ldots, r$, $r \geqslant 0$, are the states sent along pending hierarchical edges and $q \in Q$ is the linear state reached at the end of w_1. Here pending hierarchical edges refer to call symbols such that the current prefix w_1 does not have a matching return. A matching return may, or may not, occur in the remainder of the input w_2. The frontier of the computation completely determines how the computation can be continued on the remainder of the input.

Analogously, for $w \in \mathrm{NW}(\Sigma)$ and $(p_1, \ldots, p_r, q) \in P^r \times Q$, $r \geqslant 0$, $p_i \in P$, $1 \leqslant i \leqslant r$, $q \in Q$, by a computation of A on w with *initial frontier* $(p_1, \ldots, p_r, q)$ we mean a computation beginning with linear state q and where for the first r unmatched returns in w (if w has r unmatched returns) the computation uses the hierarchical states p_r, p_{r-1}, ..., p_1, in this order. If w has more than r unmatched calls, the computation uses the initial hierarchical state for the additional unmatched calls.

A computation C of the NNWA is *accepting* if the frontier at the end of the computation is of the form $(p_1, \ldots, p_r, q)$, $q \in Q_f$, $p_i \in P_f$, $i = 1, \ldots, r$, $r \geqslant 0$. The NNWA A *accepts* a nested word w if it has an accepting computation on w. The nested word language recognized by A is denoted $L(A)$. A nested word language is *regular* if it is recognized by an NNWA.

We note that there is a natural correspondence between nested word automata and *input-driven pushdown automata* [23,7], recently studied under the new name of *visibly pushdown automata* [2,4,8]. The computation of an NNWA A on a nested word w can be viewed as a computation of an input-driven pushdown automaton B on the underlying string of w where the linear states of A constitute the set of states of B and the hierarchical states are the stack symbols of B.

As special cases of the automata of Definition 2.1 we obtain the unambiguous and the deterministic nested word automata. A nested word automaton A is *unambiguous* (a UNWA) if it has exactly one accepting computation for any $w \in L(A)$. A nested word automaton A is said to be *deterministic* (a DNWA) if Q_0 and P_0 are singleton sets and δ_c (respectively, δ_i, δ_r) is a partial function $Q \times \langle \Sigma \to Q \times P$ (respectively, $Q \times \Sigma \to Q$, $Q \times P \times \Sigma\rangle \to Q$). Note that we allow a DNWA to be incomplete, that is, some values of the transition functions may be undefined. Any DNWA can be completed in the usual way, that is, the transition functions can be made to be total functions by adding at most one linear and one hierarchical state.

An extension of the subset construction allows a deterministic simulation of an NNWA. An NNWA is said to be *linearly accepting* if all hierarchical states are final. A linearly accepting NNWA decides whether or not to accept the input based only on the linear state it reaches at the end of the computation. An arbitrary NNWA can be converted to a linearly accepting one by doubling the number of states. The following result by Alur and Madhusudan [4,3], gives an upper bound for the size blow-up of determinizing an NNWA. The upper bound is tight within a multiplicative constant.

Proposition 2.1 (Alur and Madhusudan [4]). *A linearly accepting NNWA with k linear and h hierarchical states can be simulated by a DNWA with $2^{k \cdot h}$ linear and 2^{h^2} hierarchical states. There exist languages of nested words L_n, $n \geqslant 1$, recognized by an NNWA with $O(n)$ (linear and hierarchical) states such that any DNWA for L_n needs 2^{n^2} states.*

3 Lower Bounds for the Size of Nested Word Automata

We recall some techniques for establishing lower bounds on the size of deterministic and nondeterministic nested word automata [13,28]. These results are straightforward extensions of the well known fooling set method for NFAs [6,33]. However, note that also in the case of *deterministic* nested word automata these methods do not always yield a precise lower bound [31].

A finite set of nested words S is a *k-set*, $k \geqslant 1$, if each word in S has exactly k pending calls. A finite set of pairs of nested words $F = \{(x_i, y_i) \mid i = 1, 2, \ldots, m\}$ is said to be a *paired k-set*, $k \geqslant 1$, if each word x_i has exactly k pending calls.

Definition 3.1. *Let L be a nested word language over Σ.*

(i) *A k-set $S \subseteq \mathrm{NW}(\Sigma)$ is a* k-separator set *for L if every element of S is a prefix of some word in L and for any two element set $\{u, v\} \subseteq S$, $u \neq v$, there exists $x \in \mathrm{NW}(\Sigma)$ such that $ux \in L$ if and only if $vx \notin L$.*

(ii) *A paired k-set $F = \{(x_i, y_i) \mid i = 1, 2, \ldots, m\}$ is said to be a* k-fooling set *for L if:*
(iia) *$x_i y_i \in L$, $i = 1, 2, \ldots, m$, and*
(iib) *for any $1 \leqslant i < j \leqslant m$, $x_i y_j \notin L$ or $x_j y_i \notin L$.*

Lemma 3.1 ([13,28]). *Let A be a (deterministic or nondeterministic) nested word automaton with a set of linear states Q and a set of hierarchical states P.*

(i) *If A is a DNWA and S is a k-separator set for $L(A)$ then $|P|^k \cdot |Q| \geqslant |S|$.*
(ii) *If $L(A)$ has a k-fooling set F, then $|P|^k \cdot |Q| \geqslant |F|$.*

The definition of a fooling set is similar to the so called (extended) fooling sets of Birget [6]. In the case of DNWAs there is a unique computation on a given prefix which means that instead of pairs of words it is sufficient to consider only individual words to be separated. The requirement that all words of a separator set (or first components of a fooling set) must have the same number of unmatched calls limits the use of Lemma 3.1.

We need a stronger lower bound condition for UNWAs to establish a tradeoff for converting a nondeterministic automaton to an unambiguous one. The first lower bound argument for UFAs was given by Schmidt [32, Thm. 3.9] in his proof of a $2^{\Omega(\sqrt{n})}$ lower bound on the NFA–UFA tradeoff. We recall here a general statement of Schmidt's lower bound method due to Leung [20]:

Schmidt's Theorem [32,20]. *Let $L \subseteq \Sigma^*$ be a regular language and let $\{(u_1, v_1), \ldots, (u_n, v_n)\}$ with $n \geqslant 1$ and $u_i, v_i \in \Sigma^*$ be a finite set of pairs of strings. Consider the integer matrix $M \in \mathbb{Z}^{n \times n}$ defined by $M_{i,j} = 1$ if $u_i v_j \in L$, and $M_{i,j} = 0$ otherwise. Then every UFA recognizing L has at least* $\operatorname{rank} M$ *states.*

We translate Schmidt's Theorem for unambiguous nested word automata. Let L be a regular nested word language and $F = \{(x_i, y_i) \mid i = 1, \ldots, n\}$ is a paired k-set. Analogously as above we define an integer matrix $M(F, L) \in \mathbb{Z}^{n \times n}$ by setting $M(F, L)_{i,j} = 1$ if $x_i y_j \in L$ and $M(F, L)_{i,j} = 0$ otherwise.

Lemma 3.2. *Let $F = \{(x_i, y_i) \mid i = 1, \ldots, n\}$ be a paired k-set. Suppose that a regular nested word language L is recognized by an UNWA A with a set of linear states Q and a set of hierarchical states P. Then*

$$|P|^k \cdot |Q| \geqslant \operatorname{rank} M(F, L).$$

The proof is analogous to the one given by Leung [20, Thm. 2], except that instead of the states that A reaches at the end of the words x_i we now consider the frontiers of computations of A reached at the end of the words x_i.

4 From Unambiguous to Deterministic

The state blow-up of converting an UNWA to a DNWA turns out to be, in the worst case, the same as for determinizing a general nondeterministic nested word automaton.

Theorem 4.1. *For every $n \geqslant 1$ there exists a nested word language U_n recognized by an UNWA with $O(n)$ states, such that any DNWA for U_n needs 2^{n^2} states.*

Proof. Let $\Sigma = \{0, 1, \#, \$\}$. For $n \geqslant 1$ define the language

$$U'_n = \Big\{ \langle \#x_0\#x_1\# \ldots \#x_\ell\$v\$\rangle u \;\Big|\; x_i \in \{0,1\}^*,\ 1 \leqslant i \leqslant \ell,\ u, v \in \{0,1\}^{\lceil \log n \rceil}, \text{bit number } (v)_2 \text{ in } x_{(u)_2} \text{ is } 1 \Big\}$$

The language U'_n is recognized unambiguously as follows. In the following discussion we assume that the input word is in $\langle \#(\{0,1\}^*\#)^*\{0,1\}^*\$\{0,1\}^*\$\rangle\{0,1\}^*$, that is, of the general form given in the definition of the language U'_n. By increasing the number of states of the UNWA with a multiplicative constant it is easy to guarantee that all computations reject otherwise.

An UNWA A guesses at the first call symbol a word $u \in \{0,1\}^{\lceil \log n \rceil}$ and sends an encoding of u both in the linear and the hierarchical state. The linear computation finds the $(u)_2$th binary subword $x_{(u)_2}$, and nondeterministically guesses $v \in \{0,1\}^{\lceil \log n \rceil}$. After this the linear computation "remembers" v but not u. The computation verifies that the $(v)_2$'th bit of $x_{(u)_2}$ is 1 and then checks that the binary string occurring after the marker \$ equals to v. Finally the computation verifies using the hierarchical state that the binary string after the return symbol $\$\rangle$ is equal to u. All nondeterministic choices in a successful computation are "pre-determined" by the suffix of the input $v\$\rangle u$, and for any input there can be only one accepting computation.

For every set $R \subseteq \{0,\ldots,n-1\}^2$, consider the unique word

$$w_R = \langle \#x_0\#x_1\#\ldots\#x_{n-1}\$ \quad , \quad \text{where } x_i \in \{0,1\}^n, \\ \text{the } j\text{th bit of } x_i \text{ is 1 iff } (i,j) \in R, \;\; 0 \leqslant i,j \leqslant n-1.$$

We show that the set of words w_R, $R \subseteq \{0,\ldots,n-1\}^2$, is a 1-separator set for U'_n. Indeed, for any distinct sets $R_1 \neq R_2$ there is a pair (i,j) belonging to one of them but not to the other; assume, without loss of generality, that $(i,j) \in R_1 - R_2$. Let $i = (u)_2$ and $j = (v)_2$, $u,v \in \{0,1\}^{\lceil \log n \rceil}$. Now we have $w_{R_1} v\$\rangle u \in U'_n$ and $w_{R_2} v\$\rangle u \notin U'_n$.

By Lemma 3.1 (i), any DNWA for U'_n needs at least $2^{\frac{n^2}{2}}$ states. In the statement of the theorem we can choose $U_n = U'_{\lceil\sqrt{2}\rceil \cdot n}$. □

5 From Nondeterministic to Unambiguous

We show that transforming an NNWA to an equivalent UNWA entails, in the worst case, again the same 2^{n^2} state space blow-up.

Let $\Sigma = \{0,\$,\#,\%\}$. For $n \geqslant 1$, define

$$L_n = \Big\{ \langle \#\#u_1\$v_1\#u_2\$v_2\#\cdots\#u_r\$v_r\%v'_1\$u'_1\#v'_2\$u'_2\#\cdots\#v'_s\$u'_s\#\rangle \;\Big|\; \\ u_i, v_i, u'_j, v'_j \in 0^+,\ r,s \geqslant 1,\ 1 \leqslant i \leqslant r,\ 1 \leqslant j \leqslant s, \\ (\exists 1 \leqslant i_1 \leqslant r,\ 1 \leqslant j_1 \leqslant s)\ u_{i_1} = u'_{i_1},\ v_{j_1} = v'_{j_1},\ |u_{i_1}| \leqslant n,\ |v_{i_1}| \leqslant n \Big\}. \tag{1}$$

The language L_n can be recognized by an NNWA A with $O(n)$ states. At the first call symbol $\langle\#$ the automaton A guesses a string $u \in \{0\}^{\leqslant n}$ and sends an encoding of u in the linear and the hierarchical states. In the notations of (1), u is the word u_{i_1}. The linear computation of A nondeterministically selects in the prefix of the input preceding the marker % a subword $\#u_{i_1}\$v_{i_1}\#$ and verifies that $u = u_{i_1}$ and $|v_{i_1}| \leqslant n$. Now the linear computation "forgets" u and stores in the linear state an encoding of v_{i_1}. After seeing the middle marker %, the automaton nondeterministically selects a subword $\#v'_{j_1}\$u'_{j_1}\#$ and verifies that $v_{i_1} = v'_{j_1}$. The linear computation then checks that $|u'_{j_1}| \leqslant n$, stores it in the state and at the last call symbol with the help of the horizontal state verifies that $u'_{j_1} = u$. At each stage of the computation it is sufficient to store a unary string of length at most n, to compare the string stored in the state with a nondeterministically chosen unary string (limited by markers at both ends), and to remember whether or not the computation has passed the marker %. The computation can be done with n hierarchical and $O(n)$ linear states.

Lemma 5.1. *Any UNWA B for the language L_n needs at least $2^{\lfloor \frac{n^2}{2} \rfloor - 1}$ states.*

Proof. Denote $z = \lfloor \frac{n^2}{2} \rfloor$. In the following, let

$$p_{(1,0)}, p_{(1,1)}, p_{(2,0)}, p_{(2,1)}, \ldots, p_{(z,0)},\ p_{(z,1)} \tag{2}$$

be an arbitrary (but fixed) enumeration of $[n] \times [n]$, where n is even. If n is odd, the list (2) contains all elements of $([n] \times [n]) \setminus (n, n)$. For $\mathbf{v} = b_1 b_2 \cdots b_z$, $b_i \in \{0, 1\}$ and $i \in \{1, \ldots, z\}$, we define a binary relation on $[n]$ by setting

$$R_{\mathbf{v}} = \{p_{(1,b_1)}, p_{(2,b_2)}, \ldots, p_{(z,b_z)}\}.$$

The definition guarantees that for any $\mathbf{v_1}, \mathbf{v_2} \in \{0, 1\}^z$,

$$\mathbf{v_1} \neq \mathbf{v_2} \text{ implies } R_{\mathbf{v_1}} \cap \overline{R_{\mathbf{v_2}}} \neq \emptyset. \tag{3}$$

Here $\overline{R}$ stands for $[n] \times [n] \setminus R$. The property (3) will be the basis for constructing a suitable paired 1-set that can be used for Lemma 3.2.

For each $\mathbf{v} = b_1 b_2 \cdots b_z \in \{0, 1\}^z$ we define the nested words $x[\mathbf{v}]$ and $y[\mathbf{v}]$. Intuitively, $x[\mathbf{v}]$ will consist of a list of pairs of unary words that encode the elements of $R_{\mathbf{v}}$ and $y[\mathbf{v}]$ consists of a list of pairs of unary words that encode the complement of $R_{\mathbf{v}}$.

First, for each pair $p_{(i,b)}$ with $1 \leqslant i \leqslant z$ and $b \in \{0, 1\}$, as in (2), we denote

$$p_{(i,b)} = (\alpha_{i,b}, \beta_{i,b}).$$

We set

$$x[\mathbf{v}] = \langle \#\#0^{\alpha_{1,b_1}} \$ 0^{\beta_{1,b_1}} \# 0^{\alpha_{2,b_2}} \$ 0^{\beta_{2,b_2}} \# \cdots \# 0^{\alpha_{z,b_z}} \$ 0^{\beta_{z,b_z}} \%.$$

Let $(\delta_1, \gamma_1), \ldots, (\delta_{z'}, \gamma_{z'})$ be an arbitrary enumeration of elements of $\overline{R_{\mathbf{v}}}$. Here $z' = z$ if n is even and $z' = z + 1$ if n is odd. Denote

$$y[\mathbf{v}] = 0^{\gamma_1} \$ 0^{\delta_1} \# 0^{\gamma_2} \$ 0^{\delta_2} \# \cdots \# 0^{\gamma_{z'}} \$ 0^{\delta_{z'}} \# \rangle.$$

Now we define the paired 1-set

$$F_1 = \{\ (x[\mathbf{v}], y[\mathbf{v}])\ \mid\ \mathbf{v} \in \{0, 1\}^z\ \}.$$

By the choice of the words $x[\mathbf{v}]$ and $y[\mathbf{v}]$ for any $\mathbf{v_1}, \mathbf{v_2} \in \{0, 1\}^z$,

$$x[\mathbf{v_1}] \cdot y[\mathbf{v_2}] \in L_n \text{ iff } R_{\mathbf{v_1}} \cap \overline{R_{\mathbf{v_2}}} \neq \emptyset.$$

The above means that in the matrix $M(F_1, L_n)$ all diagonal elements are 0. On the other hand, the property (3) implies that all non-diagonal elements of $M(F_1, L_n)$ are 1 and, thus, rank $M(F_1, L_n)$ is equal to the number of rows, $2^{\lfloor \frac{n^2}{2} \rfloor}$.

Let A be an arbitrary UNWA for L_n with a set of linear states Q and a set of hierarchical states P. Now Lemma 3.2 implies that $|P| \cdot |Q| \geqslant 2^{\lfloor \frac{n^2}{2} \rfloor}$. □

We have proved the following:

Theorem 5.1. *There exists an NNWA A with n states such that any UNWA equivalent to A needs $2^{\Omega(n^2)}$ states.*

6 Complementing Nondeterministic Automata

We show that complementing an n-state NNWA requires, in the worst case, $2^{\Theta(n^2)}$ states. We consider again the languages L_n with $n \geqslant 1$, defined in (1). The following lemma can be proved by relying on Lemma 3.1 (ii) and the construction of the fooling set is slightly simpler than in the proof of Lemma 5.1.

Lemma 6.1. *Let A be an NNWA with sets of linear and hierarchical states Q and P, respectively. If A recognizes $\overline{L_n}$, then $|P| \cdot |Q| \geqslant 2^{n^2}$.*

Now we can state a state complexity bound for the complementation of an NNWA that is tight within a multiplicative constant.

Theorem 6.1. *The worst case state complexity of the complement of an n-state NNWA is $2^{\Theta(n^2)}$.*

Proof. Proposition 2.1 gives an upper bound $2^{O(n^2)}$. From Section 5 we recall that L_n can be recognized by an NNWA with $O(n)$ states and hence the result follows by Lemma 6.1. □

7 Homomorphic Images of Deterministic Automata

We establish a tight bound for the deterministic state complexity of homomorphisms. For the worst case lower bound it is sufficient to consider relabelings of the internal symbols.

We use a modification of the language (1), where, roughly speaking, the pairs of subwords to be compared occur in positions marked by a new symbol ♮. Let $\Sigma = \{0, 1, ♮, \#, \$, \%\}$. For $n \geqslant 1$ we define the language L'_n to consist of all words of the form

$$w\langle\#(\#0^+\$0^+)^*♮u\$v(\#0^+\$0^+)^*\%(0^+\$0^+\#)^*♮v\$u(\#0^+\$0^+)^*\#\rangle \tag{4}$$

where $w \in \{0,1\}^{\lceil \log n \rceil}$, $u = 0^{(w)_2}$, and $v \in 0^{\leqslant n}$. It can be verified that the language L'_n is recognized by a DNWA A with $O(n)$ states.

Let h be the homomorphism that maps the symbol ♮ to # and the symbol 1 to 0, and h maps the rest of the symbols to themselves. Note that h is an internal relabeling.

Lemma 7.1. *Any DNWA for $h(L'_n)$ needs $2^{\frac{n^2}{2}}$ states.*

Now we can state a tight bound for the state complexity of homomorphism for deterministic nested word automata. The upper bound can be established by constructing an NNWA to recognize the homomorphic image.

Theorem 7.1. *Let h be a homomorphism and A a DNWA with n states. The nested word language $h(L(A))$ can be recognized by a DNWA with $2^{O(n^2)}$ states.*

There exists an internal relabeling h and regular languages L''_n, $n \geqslant 1$, recognized by a DNWA with $O(n)$ states such that any DNWA for L''_n needs 2^{n^2} states.

8 Conclusion

We have shown that both the conversion of an n-state unambiguous nested word automaton to a deterministic one and the conversion of an n-state nondeterministic automaton to an unambiguous nondeterministic automaton needs in the worst case $2^{\Theta(n^2)}$ states. Both state complexity bounds are tight within a multiplicative constant. Future work may try to determine the associated multiplicative constants more precisely. It can be noted that also for the worst case lower bound given in [4] for determinizing a general nondeterministic nested word automaton the precise multiplicative constant is not known.

Another subject for future research is the state complexity of operations on unambiguous nested word automata, of which nothing is yet known. Not much is known about the operational state complexity of unambiguous finite automata either [25], and finding out these properties would lead to a better understanding of the power of unambiguous nondeterminism in automata.

References

1. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., Libkin, L.: First-order and temporal logics for nested words. In: Proc. of 22nd IEEE Symposium on Logic in Computer Science, pp. 151–160 (2007)
2. Alur, R., Kumar, V., Madhusudan, P., Viswanathan, M.: Congruences for visibly pushdown languages. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1102–1114. Springer, Heidelberg (2005)
3. Alur, R., Madhusudan, P.: Adding nesting structure to words. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 1–13. Springer, Heidelberg (2006)
4. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. Assoc. Comput. Mach. 56(3) (2009); full version of [3]
5. Arenas, M., Barceló, P., Libkin, L.: Regular languages of nested words: Fixed points, automata, and synchronization. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 888–900. Springer, Heidelberg (2007)
6. Birget, J.-C.: Intersection and union of regular languages and state complexity. Inform. Process. Lett. 43, 185–190 (1992)
7. von Braunmuhl, B., Verbeek, R.: nput-driven languages are recognized in $\log n$ space. In: Karpinski, M. (ed.) FCT 1983. LNCS, vol. 158, pp. 40–51. Springer, Heidelberg (1983)
8. Chervet, P., Walukiewicz, I.: Minimizing variants of visibly pushdown automata. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 135–146. Springer, Heidelberg (2007)
9. Comon, H., Gilleron, R., Jacquemard, F., Lugiez, D., Löding, C., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007), Electronic book available from: `tata.gforge.inria.fr`
10. Gauwin, O., Niehren, J., Roos, Y.: Streaming tree automata. Inform. Proc. Lett. 109, 13–17 (2008)

11. Goldstine, J., Leung, H., Wotschke, D.: On the relation between ambiguity and nondeterminism in finite automata. Inform. Comput. 100, 261–270 (1992)
12. Geffert, V., Mereghetti, C., Pighizzini, G.: Complementing two-way finite automata. Inform. Comput. 205(8), 1173–1187 (2007), http://dx.doi.org/10.1016/j.ic.2007.01.008
13. Han, Y.-S., Salomaa, K.: Nondeterministic state complexity of nested word automata. Theoret. Comput. Sci. 410, 2961–2971 (2009)
14. Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. Internat. J. Foundations of Comput. Sci. 14, 1087–1102 (2003)
15. Holzer, M., Kutrib, M.: Nondeterministic Finite Automata—Recent Results on the Descriptional and Computational Complexity. In: Ibarra, O.H., Ravikumar, B. (eds.) CIAA 2008. LNCS, vol. 5148, pp. 1–16. Springer, Heidelberg (2008)
16. Holzer, M., Kutrib, M.: Descriptional and computational complexity of finite automata. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 23–42. Springer, Heidelberg (2009)
17. Hromkovič, J.: Communication Complexity and Parallel Computing. Springer, Heidelberg (1997)
18. Jirásková, G.: State complexity of some operations on binary regular languages. Theoret. Comput. Sci. 330, 287–298 (2005)
19. Leung, H.: Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata. SIAM J. Comput. 27(4), 1073–1082 (1998)
20. Leung, H.: Descriptional complexity of NFA of different ambiguity. Internat. J. Foundations Comput. Sci. 16(5), 975–984 (2005), http://dx.doi.org/10.1142/S0129054105003418
21. Liu, G., Martin-Vide, C., Salomaa, A., Yu, S.: The state-complexity of two combined operations: Star of catenation and star of reversal. Inform. Comput. 206, 1178–1186 (2008)
22. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: Proc. 38th ACM Symposium on Principles of Programming Languages, POPL 2011, pp. 283–294 (2011)
23. Mehlhorn, K.: Pebbling mountain ranges and its application to DCFL-recognition. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 422–435. Springer, Heidelberg (1980), http://dx.doi.org/10.1007/3-540-10003-2_89
24. Neumann, A., Seidl, H.: Locating matches of tree patterns in forests. In: Arvind, V., Sarukkai, S. (eds.) FST TCS 1998. LNCS, vol. 1530, pp. 134–146. Springer, Heidelberg (1998)
25. Okhotin, A.: Unambiguous finite automata over a unary alphabet. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 556–567. Springer, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-15155-2_49
26. Okhotin, A.: Comparing linear conjunctive languages to subfamilies of the context-free languages. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K., Králović, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 431–443. Springer, Heidelberg (2011)
27. Okhotin, A., Salomaa, K.: State complexity of operations on input-driven pushdown automata (February 2011) (manuscript in preparation)
28. Piao, X., Salomaa, K.: Operational state complexity of nested word automata. Theoret. Comput. Sci. 410, 3290–3302 (2009)
29. Ravikumar, B., Ibarra, O.H.: Relating the type of ambiguity of finite automata to the succinctness of their representation. SIAM J. Comput. 18(6), 1263–1282 (1989), http://dx.doi.org/10.1137/0218083

30. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. I-III. Springer, Heidelberg (1997)
31. Salomaa, K.: Limitations of lower bound methods for deterministic nested word automata. Inform. Comput. 209, 580–589 (2011)
32. Schmidt, E.M.: Succinctness of Description of Context-Free, Regular and Unambiguous Languages, Ph. D. thesis. Cornell University (1978)
33. Shallit, J.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press, Cambridge (2009)
34. Yu, S.: Regular languages. In: [30], vol. I, pp. 41–110

Avalanche Structure in the Kadanoff Sand Pile Model*

Kévin Perrot and Eric Rémila

Université de Lyon
Laboratoire de l'Informatique du Parallélisme
(umr 5668 CNRS - ENS de Lyon - Université Lyon 1)
site Monod, ENS de Lyon, 46 allée d'Italie, 69364 Lyon Cedex 7, France
{kevin.perrot,eric.remila}@ens-lyon.fr

Abstract. Sand pile models are dynamical systems emphasizing the phenomenon of *Self Organized Criticality* (SOC). From N stacked grains, iterating evolution rules leads to some critical configuration where a small disturbance has deep consequences on the system, involving numerous steps of grain fall. Physicists L. Kadanoff *et al* inspire KSPM, a model presenting a sharp SOC behavior, extending the well known *Sand Pile Model*. In KSPM with parameter D we start from a pile of N stacked grains and apply the rule: $D-1$ grains can fall from column i onto the $D-1$ adjacent columns to the right if the difference of height between columns i and $i+1$ is greater or equal to D. We propose an iterative study of KSPM evolution where one single grain addition is repeated on a heap of sand. The sequence of grain falls following a single grain addition is called an avalanche. From a certain column precisely studied for $D = 3$, we provide a plain process describing avalanches. We hope that this process is a first stone toward the study of KSPM fixed points structure.

Keywords: discrete dynamical system, self-organized criticality, sand pile model.

1 Introduction

Sand pile models were introduced in [1] as systems presenting a critical self-organized behavior, a property of dynamical systems having critical points as attractors. In the scope of sand piles, starting from an initial configuration of N stacked grains the local evolution of particles is described by one or more iteration rules. Successive applications of such rules alter the configuration until it reaches an attractor, namely a stable state from which no rule can be applied. SOC property means those attractors are critical in the sense that a small perturbation — adding some more grains — involves an arbitrary deep reorganization of the system. Sand pile models were well studied in recent years ([9],[5],[6],[15]).

* Partially supported by IXXI (Complex System Institute, Lyon) and ANR project Subtile.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 427–439, 2011.

1.1 Kadanoff Sand Pile Model

In [12], Kadanoff proposed a generalization of classical models closer to physical behavior of sand piles in which more than one grain can fall from a column during one iteration. Informally, Kadanoff sand pile model with parameter D and N grains is a discrete dynamical system, which initial configuration is composed of N stacked grains, moving in discrete space and time according to a transition rule: if the height difference between column i and $i+1$ is greater or equal to D, then $D-1$ grains can fall from column i to the $D-1$ adjacent columns on the right (see figure 1).

Sand pile models are specializations of *Chip Firing Games* (CFG). A CFG is played on a directed graph in which each vertex v has a load $l(v)$ and a threshold $t(v) = deg^+(v)$,[1] and the transition rule is: if $l(v) \geq t(v)$ then v gives one unit to each of its neighbors (we say v is fired). As a consequence, we inherit all properties of CFGs.

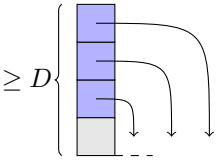


Fig. 1. KSPM(D) transition rule

Kadanoff sand pile is referred to a *linear chip firing game* in [11]. The authors show that the set of reachable configurations endowed with the order induced by the successor relation has a lattice structure, in particular it has a unique *fixed point*. Since the model is non-deterministic, they also prove *strong convergence i.e.* the number of iterations to reach the fixed point is the same whatever the evolution strategy is. The morphism from KSPM(3) to CFG is depicted on figure 2.

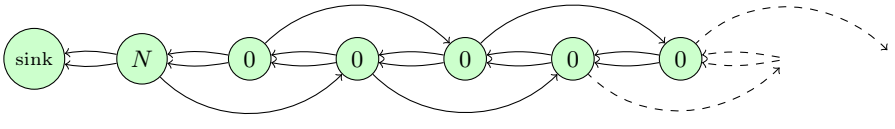


Fig. 2. The initial configuration σ of KSPM(3) is presented as a CFG where each vertex corresponds to a column (except the sink) seen as a difference of height

More formally, the sand pile models we consider are defined on the space of ultimately null decreasing integer sequences. Each integer represents a column of stacked sand grains and transition rules describe how grains can move from columns. Let $h = (h_0, h_1, h_2, \dots)$ denote a *configuration* of the model, where each integer h_i is the number of grains on column i. Configurations can also be given as height difference $\sigma = (\sigma_0, \sigma_1, \sigma_2, \dots)$, where for all $i \geq 0$, $\sigma_i = h_i - h_{i+1}$. We will use this latter representation throughout the paper, within the space of ultimately null non-negative integer sequences.

[1] $deg^+(v)$ denotes the out-degree of v.

Definition 1. *The Kadanoff sand pile model with parameter D, KSPM(D), is defined by:*

- *A set of* configurations*, consisting in ultimately null non-negative integer sequences.*
- *A set of* transition rules*: we have a transition from a configuration σ to a configuration σ' on column i, and we note $\sigma \xrightarrow{i} \sigma'$ if*
  - $\sigma'_{i-1} = \sigma_{i-1} + D - 1$ *(for $i \neq 0$)*
  - $\sigma'_i = \sigma_i - D$,
  - $\sigma'_{i+D-1} = \sigma_{i+D-1} + 1$
  - $\sigma'_j = \sigma_j$ *for* $j \notin \{i-1, i, i+D-1\}$.

Remark that according to the definition of the transition rules, a condition for σ' to be a configuration is that $\sigma_i \geq D$.

1.2 Strategies and Avalanches

A basic property of the KSPM model is the *diamond property*. If there exists two distinct integers i and j such that $\sigma \xrightarrow{i} \sigma'$ and $\sigma \xrightarrow{j} \sigma''$, then there exists a configuration σ''' such that $\sigma' \xrightarrow{j} \sigma'''$ and $\sigma'' \xrightarrow{i} \sigma'''$. We note $\sigma \to \sigma'$ when there exists an integer i such that $\sigma \xrightarrow{i} \sigma'$. We define the transitive closure $\xrightarrow{*}$ of $\to$, and say that σ' is *reachable* from σ when $\sigma \xrightarrow{*} \sigma'$.

A *strategy* is a sequence $s = (s_1, \ldots, s_T)$. We say that σ' is *reached* from σ via s when $\sigma \xrightarrow{s_1} \sigma'' \xrightarrow{s_2} \ldots \xrightarrow{s_T} \sigma'$ and we note $\sigma \xrightarrow{s} \sigma'$. We also say, for each integer t such that $0 < t \leq T$, that the column s_t *is fired* at *time* t in s. (informally, the index of the sequence is interpreted as time).

For any strategy s and any nonnegative integer i, we state $|s|_i = \#\{t | s_t = i\}$. Let s^0, s^1 be two strategies such that $\sigma \xrightarrow{s^0} \sigma^0$ and $\sigma \xrightarrow{s^1} \sigma^1$. We have the equivalence: $[\forall\ i, |s^0|_i = |s^1|_i] \Leftrightarrow \sigma^0 = \sigma^1$. A strategy s such that $\sigma \xrightarrow{s} \sigma'$ is called *leftmost* if it is the minimal strategy from σ to σ' according to lexicographic order. A leftmost strategy is such that at each iteration, the leftmost possible transition is performed.

We say that a configuration σ is *stable*, or a *fixed point* if no transition is possible from σ. As a consequence of the diamond property, one can easily check that, for each configuration σ, there exists a unique stable configuration, denoted by $\pi(\sigma)$, such that $\sigma \xrightarrow{*} \pi(\sigma)$. Moreover, for any configuration σ' such that $\sigma \xrightarrow{*} \sigma'$, we have $\pi(\sigma') = \pi(\sigma)$ (see [11] for details).

In this paper, we are interested in the iterative process defined below. Starting with no grain, we successively add a single grain on column 0, and make all the possible firings until a fixed point is reached. We denote by $\pi(k)$ the configuration obtained with this process using k grains (from the structure of KSPM described above, one easily checks that $\pi(k) = \pi((k, 0^\omega))$).

Let σ be a configuration, $\sigma^{\downarrow 0}$ is the configuration obtained by adding one grain on column 0. In other words, if $\sigma = (\sigma_0, \sigma_1, \ldots)$, then $\sigma^{\downarrow 0} = (\sigma_0 + 1, \sigma_1, \ldots)$.

Formally the process is defined by $\pi(0) = 0^\omega$ and the recurrence formula:

$$\pi(\pi(k-1)^{\downarrow 0}) = \pi(k).$$

The k^{th} *avalanche* s^k is the leftmost strategy from $\pi(k-1)^{\downarrow 0}$ to $\pi(k)$. The goal of the present paper is the description of avalanches. Informally, we want to describe what happens when a new grain is added in a previously stabilized sand pile.

For $D = 2$, i.e. the classical SPM, this description is easy: the added grain moves rightwards until it arrives in a plateau. But, for $D > 2$, the situation is not so simple. We now state our results.

In the general case, we prove (Section 2) the following properties:

- Each column is fired at most once,
- For any avalanche, as soon as an interval $\{L, L+1, \ldots, L+D-1\}$ of successive fired columns exists, the execution of the avalanche on the right part of this interval can be turned into a pseudo local and elementary process.

Informally, this means that the knowledge of such an interval guarantees a regular behavior of the avalanche on its right part. This behavior is precisely described in section 2.

In the case when $D = 3$, we prove (Section 3) the property below:

- For each avalanche s^k, there exists an integer $L(k)$ in $O(\log k)$ such that either no column is fired on the right of $L(k)$, or columns $L(k)$ and $L(k)+1$ both are fired (and therefore, the property of the second item above applies).

Informally, this means that the regular behavior emerges after a transitional and complex phase, but this phase has an asymptotically negligible size.

These results give a better understanding of avalanches for sufficiently large columns. We hope that in future work, they will help us in the approach of the structure of fixed points $\pi(k)$.

1.3 The Context

The problem of describing and proving regularity properties, experimentally checked, for models issued from basic dynamics is really a present challenge for physicists, mathematicians, and computer scientists. There exists a lot of conjectures, issued from simulations, on discrete dynamical systems with simple local rules (sandpile model [3] or chip firing games, but also rotor router [13], the famous Langton ant [7] [8]...) but very few results have actually been proved. As regards sand pile models, the *prediction problem* (namely, the problem of computing the fixed point $\pi(k)$ knowing $\pi(k-1)$) has been proven in [14] to be in $\mathbf{NC}^3$ for KSPM(1), which means that the time needed to compute an avalanche is in $O(\log^3 N)$ where N is the number of grains, and **P**-complete when the dimension is ≥ 3. A recent study ([10]) showed that in the two dimensional generalization of KSPM(D), the avalanche problem (given a configuration σ and a

column i on which we add one grain, does it have an influence an index j?) is **P**-complete, which points out an inherently sequential behavior.

This study will provide tools to understand sand pile evolution. We hope that those tools form a basis to obtain some good descriptions of fixed points $\pi(k)$, but are also deeply related with other subjects around sand piles such as unit elements of abelian group structures presented in [2] and [4].

2 Avalanche Process in the General Case

This section begins with a glance at avalanches, allowing notation simplifications. Then avalanches are studied in details, leading to a simplified description of its behavior.

Proposition 1. *For each strategy s such that $\pi(N)^{\downarrow 0} \xrightarrow{s} \pi(N+1)$ and any column $i \in \mathbb{N}$, we have $|s|_i \in \{0,1\}$.*

Proof. Let $s = (s_1, \ldots, s_T)$ be a strategy such that $\pi(N)^{\downarrow 0} \xrightarrow{s} \pi(N+1)$. We have to prove that, for $1 \leq l < m \leq T$, we have $s_l \neq s_m$ (obviously, $|s|_i \geq 0$ for all i). To do it, we prove by induction on $t \leq T$ that for $1 \leq l < m \leq t$, we have $s_l \neq s_m$.

For initialization this is obviously true for $t = 1$. Now assume that the condition is satisfied for an integer t such that $t < T$, and let i be a column such that there exists an integer $l \leq t$ such that $i = s_l$. Let σ be the configuration such that $\pi(N)^{\downarrow 0} \xrightarrow{s_1} \ldots \xrightarrow{s_t} \sigma$.

Notice that the transitions which can possibly change the value of the current configuration at i could be: i (which decreases the value by D units), $i+1$ (which increases the value by $D-1$ units) or $i-D+1$ (which increases the value by 1 unit).

Thus we have $\sigma_i \leq \pi(N)_i^{\downarrow 0} - D + D - 1 + 1$ since by definition, between $\pi(N)$ and σ, exactly one transition has occurred in i, at most one transition has occurred in $i+1$, and at most one transition has occurred in $i-D+1$. For $i \geq 1$, we get $\sigma_i \leq \pi(N)_i$. On the other hand, since $\pi(N)$ is a fixed point, we have: $\pi(N)_i < D$, which guarantees that $s_{t+1} \neq i$. For $i = 0$, there is no possible transition in $i - D + 1$, thus we get $\sigma_0 \leq \pi(N)_0^{\downarrow 0} - D + D - 1$, which is $\sigma_0 \leq \pi(N)_0 + 1 - D + D - 1$. Thus $\sigma_0 \leq \pi(N)_0 < D$ which also gives: $s_{t+1} \neq 0$.

This ensures that the result is true for $t+1$, and, by induction, for T.

When talking about an avalanche s, Lemma 1 allows us to write $i \in s$ instead of $|s|_i = 1$ without lose of information. We denote by $s_{[u,v]}$ the subsequence of s from u to v included.

We will now study avalanches in details. For $D = 2$, i.e. the classical SPM, avalanches are quite simple, the added grain moves rightwards until it finds a stable position. For $D > 2$, the situation is more complex, and needs a precise study, given by the following Lemmas.

We first explain the "pseudo locality" of avalanches : at a time $t+1$, a fired column can't be at distance greater — neither on the left nor on the right — than

$D-1$ of the greatest fired column of $s_{[1,t]}$. Imagine, during an avalanche, that you follow the greatest fired column with a frame of size $2(D-1)-1$, Lemma 1 tells that you won't miss any firing.

Lemma 1. *Let $s = (s_1, \ldots, s_{t_k})$ be the k^{th} avalanche. Let $r_t = \max s_{[1,t]}$.*

- *Assume that $s_{t+1} < r_t$. Then s_{t+1} is the largest column number satisfying this inequality, which has not yet been fired at time t. In other words:*

$$s_{t+1} = \max\{i \mid i < r_t \text{ and } i \notin s_{[1,t]}\}$$

  Moreover, we have $r_t - s_{t+1} < D-1$.
- *Assume that $s_{t+1} > r_t$. Then we have $s_{t+1} - r_t \leq D-1$.*

Proof. We order fired columns by causality. Precisely, a column i has two potential predecessors, which are $i+1$ and $i-D+1$. State $i = s_u$. These columns are really predecessors of i if they are elements of $s_{[1,u]}$, i.e if they are fired before i. Using the transitive closure, we define a partial order relation (denoted $<_{caus.}$) on fired columns for s.

Now, consider the set A_{t+1} of ancestors of s_{t+1} (i.e. the set of columns i such that $i <_{caus.} s_{t+1}$) and the set S_t of columns which have s_t as an ancestor (i.e. columns i such that $s_t <_{caus.} i$). We necessarily have $r_t \in A_{t+1}$. Otherwise, we have $A_{t+1} \cap S_t = \emptyset$, and this allows another strategy s', constructed from s by postponing the transitions at r_t and elements of S_t after the transition on s_{t+1}. This contradicts the fact that s is leftmost.

Let $(i_0, i_1, \ldots, i_p)$ be a finite sequence such that $i_0 = r_t$, $i_p = s_{t+1}$ and, for each j with $0 \leq j < p$, i_j is a predecessor of i_{j+1}. Such a sequence exists since $r_t \in A_{t+1}$. Let us prove by induction that $i_j = r_t - j$: this is true for $j = 0$. Assume it is true until the integer $j < p$. We have either $i_{j+1} = i_j - 1$ or $i_{j+1} = i_j + D - 1$. But from the induction hypothesis, $i_j + D - 1$ is an ancestor of i_{j+1}, thus $i_{j+1} = i_j - 1$. This gives that s_{t+1} is the largest column number i such that $i < r_t$ and $i \notin s_{[1,t]}$.

Now if we assume, by contradiction, that $p \geq D-1$, then $r_t - D + 1$ is not a predecessor of r_t, which yields that r_t has no predecessor, which is a contradiction. This gives the inequality of the first item. The second item is obvious, since s_{t+1} ha a unique predecessor which is $s_{t+1} - D + 1$.

Lemma 1 induces a partition of fired columns between those which make a progress (*i.e.* increases the greatest fired column) and those which do not. This distinction is important in further development, so let us give progress firings a name. Let $s = (s_1, \ldots, s_T)$ be an avalanche, a column s_t is called a *peak* if and only if $s_t > \max s_{[1,t-1]}$.

Remark 1. Two peaks $p \neq q$ can be compared using chronological ($<_T$) or spatial ($<_S$) orders. Nevertheless, by definition of peaks we obviously have $p <_T q \iff p <_S q$.

The next Lemma explains precisely the way peaks appear, as soon as a $D-1$ successive columns are fired. It follows an intuitive idea : a peak at time $t+1$ is

a column which only receives grains from the left part of the sand pile (within $s_{[1,t]}$). Therefore, the amount it receives is at most 1 and a peak must have an initial value of $D-1$ units of height difference. Also, a non-peak column isn't fired when it receives 1 unit of height difference so it has to wait for its right successor to be fired, in a kind of chain reaction.

Lemma 2. *Let s be the k^{th} avalanche. Assume that there exists a column l, such that for each column i with $l \leq i < l+D-1$, $i \in s$, and a fired column $i' \in s$ such that $i' \geq l+D-1$. Let l' be the lowest peak such that $l' \geq l+D-1$. There exists a time t such that* $\begin{cases} \text{for all } i \text{ with } l'-D+1 < i \leq l',\ i \in s_{[1,t]} \\ \text{for all } i \text{ with } l' < i,\ i \notin s_{[1,t]} \end{cases}$
Moreover l' is the lowest integer such that $l' \geq l+D-1$ and $\sigma^t_{l'} = D-1$.

Informally, the lemma above claims that the space threshold l' induces a corresponding time threshold t: columns fired before time t are on the left of l', while columns fired after time t are on the right of l'.

Proof. Let t_0 be the time when $s_{t_0} = l'$, i.e. the first time such that $s_{t_0} \geq l+D-1$, and let j be the largest integer such that, for $0 \leq j' \leq j$, we have $s_{t_0+j'} = s_{t_0} - j'$. Let us state $t = t_0 + j$. We have $j < D-1$.

Let σ^t denote the configuration obtained from $\pi(k-1)$ via $s_{[1,t]}$. Let i, with $i < l'$, such that $i \notin s_{[1,t]}$. We claim that we have: $i \notin s$. To prove it, we prove by induction that for any $t' \geq t$, $i \notin s_{[1,t']}$. Assume that this is satisfied for a fixed t'. This means that all the transitions of $s_{[t+1,t']}$ are done on columns larger than l'. Thus, $\sigma^{t'}_i = \sigma^t_i$ and no transition is possible on i for σ^t since s is leftmost (the only potential column to be fired is $s_{t_0} - j - 1$, but by assumption, either this column has been previously fired, or it cannot be fired by definition of j, according to Lemma 1) .

By contraposition, it follows that for each column i with $l \leq i < l+D-1$, we have $i \in s_{[1,t]}$. A simple (reverse sense) induction shows that, for $l+D-1 \leq i \leq l'$ Âă we have $i \in s$, since by hypothesis $i+1$, and $i+1-D$ both are in s. Thus, by contraposition of the claim above, for $l+D-1 \leq i \leq l'$, Âă we have $i \in s_{[1,t]}$. This gives the the fact that for all i with $l'-D+1 < i \leq l'$, $i \in s_{[1,t]}$

The fact that for all i with $l' < i$, $i \notin s_{[1,t]}$ is trivial, by definition of t_0 and t.

We have $l' > l+D-2$ and $\sigma^t_{l'} = D-1$. assume that there exists $l'' < l'$ satisfying the same properties. Notice that for $t_0 \leq t' \leq t$ we have $s'_t > l'-D-1$. Thus the time t_1 such that $s_{t_1} = l''-D+1$ is such that $t_1 < t_0$. That means that l'' should have been fired before t_0, a contradiction.

Lemma 2 describes in a very simple way the behavior of avalanches. Thanks to it, the study of an avalanche can be turned into a pseudo linear execution, in which transitions are organized in a clear fashion:

Theorem 1. *Let $s = (s_1, \ldots, s_T)$ be the k^{th} avalanche and $(p_1, \ldots, p_q)$ be its sequence of peaks. Assume that there exists a column l, such that for each column i with $l \leq i < l+D-1$, $i \in s$. Then for any column p such that $p \geq l+D-1$,*

$$p \text{ is a peak of } s \iff \pi(k-1)_p = D-1 \text{ and } \exists i \text{ s.t. } p_i < p \leq p_i + D - 1$$

Furthermore, Let $i \leq q$ *and* t *such that* $p_i = s_t$*, with* $p_i \geq l + D - 1$*. Then*

$$T \geq t + p_i - p_{i-1} - 1 \text{ and for all } t' \text{ s.t. } t < t' \leq t + p_i - p_{i-1} - 1,\ s_{t'} = s_{t'-1} - 1$$

A graphical representation of this statement is given on figure 3.

Proof. The first part is a straight induction on Lemma 2.
The second part follows from an induction summed up in the following fact: any column i such that $\pi(k-1)_i < D-1$ must wait for its right neighbor $i+1$ to be fired, and it should be fired when both $i+1$ and $i-D+1$ have been fired (besides, $i-D+1$ has already been fired). Since any of such i is fired to reach a fixed point, $T \geq t + p_i - p_{i-1} - 1$.

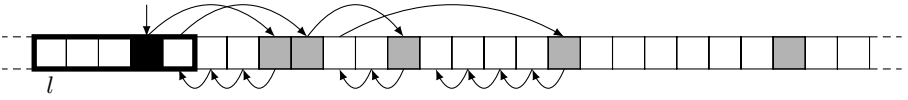


Fig. 3. Illustration of Theorem 1 with $D = 6$; surrounded columns l to $l + D - 2$ are supposed to be fired; black column is the greatest peak strictly lower than $l + D - 1$; a column is grey if and only if its value is $D - 1$; following arrows depicts the avalanche

Theorem 1 easily allows us to compute the right part of the k^{th} avalanche (from column $l + D - 1$), only knowing $\pi(k-1)$. The sequence of peaks is computed as follows. The first one is the lowest column i greater or equal to $l + D - 1$ such that $\pi(k-1)_i = D - 1$. Then, given a peak i, the next one is the lowest j such that $\pi(k-1)_j = D-1$ and $j - i \leq D - 1$. If such a j does not exist, then there is no more peak and i is the largest fired column.

We can distinguish two movements within an avalanche: before a certain column it has an unknown behavior, and from that column to the end the behavior is pseudo local, in the sense that when an index is fired ahead (on the right) then any 'hole' is filled before the progress can continue.

An important direct implication of Theorem 1 is that if there exists a column l such that for the k^{th} avalanche s^k, we have for all $l \leq i < l + D - 1$, $i \in s^k$, then for all j such that $l + D - 1 \leq j < \max s^k$, we have $j - (D-1), j, j+1 \in s^k$ and therefore $\pi(k)_j = \pi(k-1)_j$. Intuitively, this equality hints some similarity between successive avalanches.

Note that the previous results also apply for a grain addition on column 0 of any fixed point configuration of KSPM(D).

This study constitutes a simplified understanding of the behavior of avalanches, which we hope will be helpful toward the description of fixed points. As motivated above, next section studies, for KSPM(3), the previous result hypothesis that for an avalanche s^k there exists a column l such that for all $l \leq i < l + D - 1$, i is element of s^k.

3 Short Transitional Phase When $D = 3$

In this section we prove that in KSPM(3), there exists a column $l(N)$ in $O(\log N)$ such that Lemmas hypothesis is verified for any avalanche s^k, with $k \leq N$, such that $\max s^k > l(k)$. In other words, considering the N first avalanches, from a logarithmic column, we can apply Theorem 1 and consider avalanches pseudo locally, as described on figure 3. Here is the statement:

Proposition 2. *Let s be the k^{th} avalanche of KSPM(3). There exists a column $l(k)$ in $O(log\ k)$ such that for any k, when $\max s > l(k)$, $l(k)$ and $l(k)+1$ both are elements of s.*

Proof. Let i be a fixed column. If $i, i+1 \in s^k$, then, Theorem 1 states that for all i' such that $i \leq i' < \max\{j | j \in s\}$, we have $i', i'+1 \in s$. If $i, i+1 \notin s$, then, from Proposition 1, we have $\max s < i$.

Let j be a fixed positive integer. Assume that the avalanche s fires $2j$ but not $2j-1$. From Remarks above, columns $0, 2, 4, \ldots, 2j$ are fired in s while columns $1, 3, 5, \ldots, 2j-1$ are not fired.

If a column $i+1$ is fired while i is not, then we necessarily have $\pi(k-1)_i = 0$, since the firing in $i+1$ increases the value in column i from 2 units. Moreover, if the column $i+1$ is fired while $i+2$ is not, then we necessarily have $\pi(k-1)_{i+1} = 2$, since the $i+1$ receives at most one grain, by preceding firings.

On the other hand, obviously, the assumption on j enforces that $\pi(k-1)_0 = 2$ This yields that $(2,0)^{j-2}$ is a prefix of $\pi(k-1)$.

We have the following fact:

Claim. There exist constant numbers A and B, with $A > 0$ such that if a configuration $\pi(N)$ has a prefix of the form $(2,0)^j$ then $N > A4^j + B$

This is obtained by the linear algebra analysis below. This gives the result of the Proposition.

Let $\pi(N) = (\sigma_0, \sigma_1, \ldots)$ be the configuration and $a = (a_0, a_1, \ldots)$ be its *shot vector* i.e. a_i is the number of times when the column i has be fired in the first N avalanches. According to the iteration rule we have the relation:

$$\sigma_i = a_{i-2} - 3a_i + 2a_{i+1}$$

i.e.

$$a_{i+1} = \frac{1}{2}(\sigma_i - a_{i-2} + 3a_i)$$

We set $A := \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1/2 & 0 & 3/2 \end{pmatrix}$. We denote by v_i the column vector such that and $v_i^T = (0, 0, \sigma_i/2)$, and u_i the column vector such that $u_i^T = (a_{i-2}, a_{i-1}, a_i)$ (with the convention that u^T is the row vector obtained by transposition of the column vector u) . The equality above can be algebraically written in

$$u_{i+1} = Au_i + v_i$$

By iteration we get:

$$u_{i+2} = A^2 u_i + A v_i + v_{i+1}$$

If $i < j$ then $\sigma_{2i} = 2$ and $\sigma_{2i+1} = 0$ so $v_{2i}^T = (0,0,1)$ and $v_{2i+1}^T = (0,0,0)$. With this specification, we get:

$$u_{2(i+1)} = A^2 u_{2i} + b$$

with $b^T = (0,1,3/2)$. From this last relation we will deduce a condition on N to get the sequence $(2,0)^j$.

Let us first find a new basis to get the matrix A on Jordan canonical form. The characteristic polynomial of A is $\frac{1}{2}(2x+1)(x-1)^2$ and its eigenvalues are $-\frac{1}{2}$ of algebraic multiplicity 1 and 1 of algebraic multiplicity 2. Since $\dim(\ker(A-Id)) = 1$ the Jordan canonical form of A is

$$A_{Jordan} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1/2 \end{pmatrix}$$

And the new basis $E' = (e'_1, e'_2, e'_3)$ according to the canonical one $E = (e_1^T = (1,0,0), e_2^T = (0,1,0), e_3^T = (0,0,1))$ is given by the linear relations:

$$\begin{aligned} e'_1 &= e_1 + e_2 + e_3 \\ e'_2 &= e_2 + 2e_3 \\ e'_3 &= 4e_1 - 2e_2 + e_3 \end{aligned}$$

Let p be the linear mapping consisting in the projection on the line D_3 generated by e'_3 according to the direction the plane $P_{1,2}$ generated by e'_1 and e'_2. For any vector u, we have: $p(A^2u) = p(A^2(p(u)+u-p(u)) = p(A^2(p(u))+p(A^2(u-p(u))$. Notice that, by definition of p, $u-p(u)$ is element of $P_{1,2}$, and, therefore $A(u-p(u))$ and $A^2(u-p(u))$ also are elements of $P_{1,2}$. This yields that $p(A^2(u-p(u)))$ is null. On the other hand, since $p(u)$ is element of D_3, $A(p(u)) = \frac{-1}{2}p(u)$ and $A^2(p(u)) = \frac{1}{4}p(u)$; thus $p(A^2(p(u)) = \frac{1}{4}p(u)$. As a conclusion, we get $p(A^2u) = \frac{1}{4}p(u)$, which, in particular, allows the following equalities:

$$\begin{aligned} p(u_{2(i+1)}) &= p(A^2 u_{2i} + b) \\ &= p(A^2 u_{2i}) + p(b) \\ &= \tfrac{1}{4}p(u_{2i}) - \tfrac{1}{18}e'_3 \end{aligned}$$

Let v be the unique vector collinear with e'_3 satisfying the equation

$$v = \frac{1}{4}v - \frac{1}{18}e'_3$$

i. e. $v = \frac{-2}{27}e'_3$. Remember that $p(v) = v$. We have

$$\begin{aligned} p(u_{2(i+1)} - v) &= p(u_{2(i+1)}) - v \\ &= \tfrac{1}{4}p(u_{2i}) - \tfrac{1}{18}e'_3 - (\tfrac{1}{4}v - \tfrac{1}{18}e'_3)) \\ &= \tfrac{1}{4}p(u_{2i} - v) \end{aligned}$$

This gives by induction:

$$p(u_0 - v) = 4^j p(u_{2j} - v)$$

Now we specify the sequence of vectors u_i, assuming that values a_i are the shot vectors of a configuration σ beginning by $(2,0)^j$. (For convention we also state $a_{-2} = N$ and $a_{-1} = 0$, thus we have $u_0^T = (N, 0, a_0)$, $u_1^T = (0, a_0, a_1)$ and $u_i^T = (a_{i-2}, a_{i-1}, a_i)$ for $i \geq 2$).

An easy computation gives that: $p(u_0 - v) = \frac{N+a_0+\frac{2}{27}}{9} e'_3$

Let x_j be defined by $p(u_{2j} - v) = x_j e'_3$. We obtain the equality:

$$\frac{N + a_0 + \frac{2}{27}}{9} = 4^j x_j$$

Obviously $a_0 \leq \frac{N}{D} = \frac{N}{3}$, which ensures that $N + a_0 + \frac{2}{27} \leq \frac{4N}{3} + 1$.

Furthermore, we necessarily have $x_j > 0$, and, from Lemma 3 proved on the bounce, each element of $p(\mathbb{Z}^3)$ is a multiple of ce'_3, where c is a positive constant. If v is element of $p(\mathbb{Z}^3)$ we can conclude that $x_j \geq c$. If v is not element of $p(\mathbb{Z}^3)$, we can conclude that $x_j \geq min\{|ck + \frac{2}{27}|, k \in \mathbb{Z}\}$. In any case, there exists a positive real d, not depending on j, such that $x_j \geq d$.

We conclude that $\frac{4N}{3} + 1 \geq 4^j d$, which gives $N > \frac{3d}{4} 4^j - \frac{3}{4}$ to get a sand pile of the form $\sigma = (2,0)^j \sigma'$.

We now give the Lemma used in the previous proof.

Lemma 3. *(constant steps) There exists a positive real c such that $p_{e'_3}(\mathbb{Z}^3) = \{i\, ce'_3, i \in \mathbb{Z}\}$.*

Proof. The set of reals r such that there exists an element x in $\mathbb{Z}^3$ such that $p_{e'_3}(x) = re'_3$ is obviously a group. So we only have to prove that this group is discrete, *i.e.* that there is no sequence $(r_n)_{n\in\mathbb{Z}}$ of positive reals such that $\lim_{n\to\infty}(r_n) = 0$.

Assume, by contradiction, the existence of such a sequence, and let $(x_n)_{n\in\mathbb{Z}}$ be a sequence of vectors such that, for each integer n, $p_{e'_3}(x_n) = r_n$.

A key-point is that vectors e'_1, e'_2 and e'_3 have integer components, so we can state $x_n = a_n e'_1 + b_n e'_2 + c_n e'_3$. The sequence $(x'_n)_{n\in\mathbb{Z}}$ defined by $x'_n = (a_n - \lfloor a_n \rfloor)e'_1 + (b_n - \lfloor b_n \rfloor)e'_2 + c_n e'_3$ also is a sequence of integer vectors such that for each integer n, $p_{e'_3}(x'_n) = r_n$. Moreover this sequence is bounded. Thus $(x'_n)_n \in \mathbb{Z}$ takes a finite number of values, which enforces that the sequence $(r_n)_n \in \mathbb{Z}$ also takes a finite number of values, which is a contradiction.

For KSPM(3), after a short transitional of logarithmic length, the hypotheses of Theorem 1 are verified , and the study of avalanches can be turned into a pseudo linear process. Note that a trivial framing of the maximal non-empty column $e(N)$ of a fixed point with N grains shows that $e(N)$ is in $\Omega(\sqrt{N})$. As a consequence, pseudo local process stands for the asymptotically complete behavior of avalanches.

Unfortunately, the approach above does not hold for $D > 3$. The main reason is that, for $D = 3$ unfired columns induce a very particular and *periodic* prefix $((2,0)^j)$ on configurations. From $D = 4$, the structure of such a possible prefix is more complex and we did not yet get a tractable characterization of those prefixes.

4 Perspectives

In this paper we described avalanches as pseudo local processes from a certain column l.

We proved this column to be logarithmic in the number of grains N for KSPM(3), leading to an asymptotically complete description of avalanches in that case. Simulations for other parameter D suggest that the same outcome also holds.

The pseudo local process description involves some properties on avalanches, which we hope will be useful toward the study of fixed points shape. For an avalanche s, a particularly interesting consequence is that two successive fixed points are equal from $l+D-1$ to $(\max s)-1$, which hints that the next avalanche reaching this part of the configuration may have a similar behavior. This would lead to a knowledge on the likeness of successive avalanches and therefore a foresee on the shapes of fixed points. Further work may concentrate on this point, where the main purpose is to go ahead iterating evolution rules, and to characterize fixed points with a plain formula.

References

1. Bak, P., Tang, C., Wiesenfeld, K.: Self-organized criticality. Phys. Rev. A 38(1), 364–374 (1988)
2. Creutz, M.: Cellular automata and self organized criticality. In: Some New Directions in Science on Computers (1996)
3. Dartois, A., Magnien, C.: Results and conjectures on the sandpile identity on a lattice. In: Morvan, M., Rémila, É. (eds.) Discrete Models for Complex Systems, DMCS 2003. DMTCS Proceedings, Discrete Mathematics and Theoretical Computer Science, vol. AB, pp. 89–102 (2003)
4. Dhar, D.: Self-organized critical state of sandpile automaton models. Phys. Rev. Lett. 64(14), 1613–1616 (1990)
5. Durand-Lose, J.O.: Parallel transient time of one-dimensional sand pile. Theor. Comput. Sci. 205(1-2), 183–193 (1998)
6. Formenti, E., Masson, B., Pisokas, T.: Advances in symmetric sandpiles. Fundam. Inform. 76(1-2), 91–112 (2007)
7. Gajardo, A., Moreira, A., Goles, E.: Complexity of Langton's ant. Discrete Applied Mathematics 117(1-3), 41–50 (2002)
8. Gale, D., Propp, J., Sutherland, S., Troubetzkoy, S.: Further travels with my ant. Mathematical Entertainments Column, Mathematical Intelligencer 17, 48–56 (1995)
9. Goles, E., Kiwi, M.A.: Games on line graphs and sand piles. Theor. Comput. Sci. 115(2), 321–349 (1993)
10. Goles, E., Martin, B.: Computational Complexity of Avalanches in the Kadanoff Two-dimensional Sandpile Model. In: TUCS (ed.) Proceedings of JAC 2010 Journées Automates Cellulaires 2010, Turku Finland, pp. 121–132 F.1.1 (December 2010)

11. Goles, E., Morvan, M., Phan, H.D.: The structure of a linear chip firing game and related models. Theor. Comput. Sci. 270(1-2), 827–841 (2002)
12. Kadanoff, L.P., Nagel, S.R., Wu, L., Zhou, S.M.: Scaling and universality in avalanches. Phys. Rev. A 39(12), 6524–6537 (1989)
13. Levine, L., Peres, Y.: Spherical asymptotics for the rotor-router model in z d. Indiana Univ. Math. J., 431–450 (2008)
14. Moore, C., Nilsson, M.: The computational complexity of sandpiles. Journal of Statistical Physics 96, 205–224 (1999), 10.1023/A:1004524500416
15. Phan, T.H.D.: Two sided sand piles model and unimodal sequences. ITA 42(3), 631–646 (2008)

Well-Quasi-Ordering Hereditarily Finite Sets

Alberto Policriti[1] and Alexandru I. Tomescu[1,2]

[1] Dipartimento di Matematica e Informatica, Università di Udine,
Via delle Scienze, 206, 33100 Udine, Italy
{alberto.policriti,alexandru.tomescu}@uniud.it
[2] Faculty of Mathematics and Computer Science, University of Bucharest,
Str. Academiei, 14, 010014 Bucharest, Romania

Abstract. Recently, strong immersion was shown to be a well-quasi-order on the class of all tournaments. Hereditarily finite sets can be viewed as digraphs, which are also acyclic and extensional. Although strong immersion between extensional acyclic digraphs is not a well-quasi-order, we introduce two conditions that guarantee this property. We prove that the class of extensional acyclic digraphs corresponding to *slim* sets (i.e. sets in which every memebership is necessary) of bounded *skewness* (i.e. sets whose $\in$-distance between their elements is bounded) is well-quasi-ordered by strong immersion.

Our results hold for sets of bounded cardinality and it remains open whether they hold in general.

Keywords: well-quasi-order, hereditarily finite set, strong immersion, extensional digraph.

1 Introduction

Sets are directed graphs (*digraphs*) when one interprets the membership relation $\in$ as the adjacency relation $\leftarrow$:

$$a \in b \Leftrightarrow a \leftarrow b.$$

Such a digraph is called the *membership digraph* of a set. According to this view, set-theoretic axioms isolate classes of digraphs whose study can be "assisted" by the underlying set-theoretic semantics. The most natural and initial of such classes of digraphs is the one defined using the axiom of *extensionality*: no two vertices of a digraph in such a class have the same *set* of out-neighbors. More intriguing (even though still very basic) axioms can be considered. In this paper we study the problem of existence of well-quasi-orders on subclasses of the class of digraphs, with such set-theoretic assistance. We start with finite extensional acyclic digraphs, that is, membership digraphs of *hereditarily finite sets*.

One of the earliest results on well-quasi-orderings of graphs belongs to Kruskal [8], who showed that the class of all finite trees is well-quasi-ordered by the topological minor relation. This study culminated with the celebrated theorem of Robertson and Seymour [13] stating that the minor relation is a well-quasi-order

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 440–451, 2011.

on the class of all finite graphs. Later [14], they showed that this is also the case for weak immersion between graphs, which was a conjecture of Nash-Williams [10]. In the case of *digraphs* not much is known. Immersion between *eulerian* digraphs was studied by Johnson (cf. [3, p. 517], [5]). Recently, Chudnovsky and Seymour [5] proved that *strong immersion* between digraphs is a well-quasi-order on the set of all tournaments (i.e., orientations of complete graphs). In view of this result, we will focus in this paper on strong immersion between digraphs.

Notice that, requiring acyclicity and extensionality, tournaments are forced to be isomorphic to the membership digraphs of the so-called von Neumann's numerals. Starting from this observation and given that the full family of hereditarily finite sets is not well-quasi-ordered by strong immersion (see below), it is natural to ask for other (sub-)collections of the hereditarily finite sets that could be well-quasi-ordered by strong immersion.

The result of this paper has been obtained by introducing two conditions: the first (*slimness*) guarantees that the number of sets at any given rank in the transitive closure of a set is bounded by its cardinality; the second (bounded *skewness*) guarantees that arcs (memberships) do not reach too freely into its membership digraph. Notice that both these facts are true in the membership digraphs of von Neumann's numerals.

Our result is based on the fact that slimness together with bounded cardinality and skewness are sufficient to ensure that the entire transitive closure of a hereditarily finite set can be described as a sequence of characters in a suitable finite alphabet. From this, the existence of a wqo follows by standard means.

It is not clear exactly to what extent slimness and bounded skewness are necessary, even though in the conclusion we observe that the bounds on cardinality and skewness cannot be both dropped on slim sets without losing wqo.

Wqo's proved to be a key ingredient in generalizing and unifying many results concerning the decidability of verification problems (e.g. coverability) on infinite-state transition systems (cf. [6,1] and the references therein). To be more precise, a transition system is said to be *well-structured* when its transition relation is monotonic w.r.t. a wqo of its states; the classical example is that of Petri nets: the states of the transition system is the set of all configurations of the net, while the wqo is the inclusion between their markings. In this light, our contribution can also be viewed as laying the set-theoretic groundwork for a class of well-structured transition systems having as states the hereditarily finite sets considered in this paper.

The outline of the paper is the following. In Sec. 2 we give the main definitions and argue that weak immersion is not a wqo on the class of all extensional acyclic digraphs. Sec. 3 gives some lemmas about the structure of slim sets, which are then used in Sec. 4 to provide an encoding for slim sets of bounded cardinality and skewness. All results are assembled in Sec. 5, by proving that strong immersion is a wqo on the class of membership digraphs corresponding to slim sets of bounded cardinality and skewness. Finally, some open questions are put forward in Sec. 6.

2 Basics and Notation

2.1 Sets and Digraphs

Our notation follows [9,3]. In this paper we consider hereditarily finite sets, that is, sets belonging to the first ω levels of the von Neumann's cumulative hierarchy. Their collection, which we denote by HF, is defined as $\mathsf{HF} = \bigcup_{i\in\omega} \mathcal{V}_i$, where

$$\mathcal{V}_0 = \emptyset, \quad \mathcal{V}_{i+1} = \mathscr{P}(\mathcal{V}_i),$$

and $\mathscr{P}(\cdot)$ stands for the power-set operator. We will use the following, standard, notion of *rank* of a set x:

$$\mathsf{rk}(x) =_{\text{def}} \sup\{\mathsf{rk}(y) + 1 \mid y \in x\},$$

and we will denote the set of elements in x at a given rank r as $x^{=r} = \{y \in x \mid \mathsf{rk}(y) = r\}$. Analogously, $x^{\leqslant r} = \{y \in x \mid \mathsf{rk}(y) \leqslant r\}$. For any set x, we denote by $\mathsf{TrCl}(x)$ the *transitive closure* of x, defined through the recursion:

$$\mathsf{TrCl}(x) =_{\text{def}} x \cup \bigcup_{y\in x} \mathsf{TrCl}(y),$$

where we denote by $\bigcup x$ the *union-set* of x, that is the set $\{z \mid z \in y \wedge y \in x\}$. We say that a set x is *transitive* if $\bigcup x \subseteq x$. Plainly, the transitive closure of a set x is a transitive set.

Definition 1. *The* skewness *of a set x is*

$$\mathsf{skewness}(x) =_{\text{def}} \sup\{\mathsf{rk}(y) - \mathsf{rk}(z) \mid y, z \in \mathsf{TrCl}(x) \wedge z \in y\}.$$

We will also consider Ackermann's order $\prec_{\mathsf{A}}$ between hereditarily finite sets [2], defined recursively as

$$x \prec_{\mathsf{A}} y \Leftrightarrow_{\text{def}} \max_{\prec_{\mathsf{A}}}(x \setminus y) \prec_{\mathsf{A}} \max_{\prec_{\mathsf{A}}}(y \setminus x),$$

where we let, by convention, $\max_{\prec_{\mathsf{A}}} \emptyset \prec_{\mathsf{A}} x$, for any $x \in \mathsf{HF} \setminus \{\emptyset\}$.

Given a digraph $G = (V, E)$ we say that $V(G) =_{\text{def}} V$ is its set of *vertices* and $E(G) =_{\text{def}} E$ is its set of *arcs*. We will write uv as a shorthand for the arc $(u, v) \in E(G)$ (u and v are called its *endpoints*). Given $v \in V(G)$ we denote by $N^+(v)$ the set of its *out-neighbors* in G, namely the set $\{w \in V(G) \mid vw \in E(G)\}$. If $N^+(v) = \emptyset$, then we say that v is a *sink*, whereas v is a *source* if v is not an out-neighbor of any vertex of G. We say that H is a subdigraph of a digraph G if $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ and every arc of G with both endpoints in $V(H)$ is also in $E(H)$. In this case, we say that H is a subdigraph of G induced by the vertices in $V(H)$.

Any set in HF can be seen as a digraph, whose vertices correspond to sets and whose arc relation corresponds to the inverse of the membership relation, $\ni$ (see Fig. 1). To be more precise, consider the following definition.

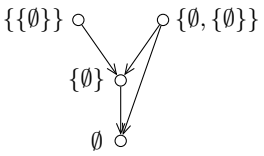


Fig. 1. The membership digraph of the set $x = \{\{\{\emptyset\}\}, \{\emptyset, \{\emptyset\}\}\}$; we have $\emptyset \prec_{\mathsf{A}} \{\emptyset\} \prec_{\mathsf{A}} \{\{\emptyset\}\} \prec_{\mathsf{A}} \{\emptyset, \{\emptyset\}\}$

Definition 2. *Given a set x, we denote by G_x the digraph (x, E_x), with*

$$E_x =_{\text{def}} \{uv \mid u, v \in x \wedge v \in u\},$$

and call the membership digraph *of x the digraph $G_{\mathsf{TrCl}(x)}$.*

The well-foundedness of the membership relation among sets ensures that membership digraphs are *acyclic*, while the *extensionality* principle guarantees that they are also *extensional*, in the following sense:

Definition 3. *A digraph G is* extensional *if for any distinct vertices u and v in $V(G)$, it holds $N^+(v) \neq N^+(u)$.*

Moreover, the converse also holds: any finite extensional acyclic digraph is the membership digraph of a hereditarily finite set. The following notion is instrumental to our results and establishes a link between a set and its graph-theoretical representation.

Definition 4. *A set x is* slim *if the digraph obtained by removing any arc from $G_{\mathsf{TrCl}(x)}$ is not extensional.*

Def. 4 is equivalent to saying that x is slim if for any vertex y of $G_{\mathsf{TrCl}(x)}$ and for any out-neighbor of it z, there exists a vertex y' of $G_{\mathsf{TrCl}(x)}$ whose set of out-neighbors is precisely $N^+(y) \setminus \{z\}$. In set-theoretic terms, a set x is slim if $\forall y \in \mathsf{TrCl}(x)$ and $\forall z \in y$, it holds that $y \setminus \{z\} \in \mathsf{TrCl}(x)$. Observe that the transitive closure of a slim set x is closed under taking subsets for its elements, in the sense that for any $y \in \mathsf{TrCl}(x)$, $\mathscr{P}(y) \subset \mathsf{TrCl}(x)$. For example, the set x whose membership digraph is depicted in Fig. 1 is slim.

In equivalent graph-theoretic terms, the rank of a set is the length of the longest simple (i.e. without repeated vertices) directed path in its membership digraph. The notion of skewness of a set can thus be interpreted as the length of the longest simple directed path in its membership digraph, so that its two endpoints are also connected by an arc. See Fig. 2 for an example of a membership digraph of skewness 5. Finally, it can be also easily seen that the Ackermann's order among hereditarily finite sets can be analogously defined for the vertices of membership digraphs.

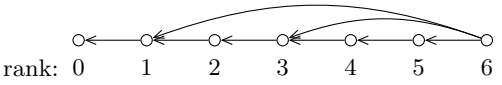


Fig. 2. A membership digraph of skewness 5

2.2 Well-Quasi-Orders and Digraph Immersion

A *quasi-order* is a pair $(Q, \preccurlyeq)$ where Q is a set and $\preccurlyeq$ is a transitive and reflexive binary relation on Q. We say that a quasi-order $(Q, \preccurlyeq)$ is a *well*-quasi-order, or *wqo*, if for every infinite sequence $(q_i)_{i=1,2,\ldots}$ of elements of Q, there exist $1 \leqslant i < j$ such that $q_i \preccurlyeq q_j$.

We refer here to the notions of *weak* and *strong immersion*, as considered in [5]. A *weak immersion* of a digraph H into G is a map η such that:

- for every $v \in V(H)$, $\eta(v) \in V(G)$;
- for every $u, v \in V(H)$ with $u \neq v$, it holds that $\eta(u) \neq \eta(v)$;
- for each arc $uv \in E(H)$, $\eta(uv)$ is a directed path in G from $\eta(u)$ to $\eta(v)$ (all paths considered are *simple*, i.e., do not have repeated vertices);
- if $e, f \in E(H)$ are distinct, then $\eta(e)$ and $\eta(f)$ have no arcs in common, although they may share vertices.

The map η is called a *strong immersion* when it also holds that if $v \in V(H)$, $e \in E(H)$, and e is not incident with v in H, then $\eta(v)$ is not a vertex on the directed path $\eta(e)$. We say that a digraph H is *weakly (strongly) immersed* into a digraph G, and write $H \preccurlyeq_i^w G$ ($H \preccurlyeq_i^s G$), if there exists a weak (strong) immersion of H into G.

As already observed in [5], weak immersion is not a wqo on the set of all digraphs. Just consider the acyclic digraphs D_n formed by orienting the edges of a cycle of length $2n$ alternately clockwise and counterclockwise (see Fig. 3). That being so, the collection $\{D_n \mid n \geqslant 2\}$ has the property that none of its elements can be weakly immersed into another one.

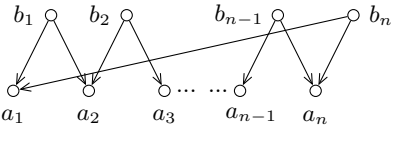


Fig. 3. Digraphs D_n, $n \geqslant 2$

Since the collection of transitive closures of sets is a collection of digraphs, it is natural to ask whether $\preccurlyeq_i^w$ or $\preccurlyeq_i^s$ form a wqo on such a collection. The answer is *no*, since the collection of membership digraphs F_n, $n \geqslant 3$, constructed as in Fig. 4 has, analogously to the corresponding D_n's, the property that no digraph can be weakly immersed into another one.

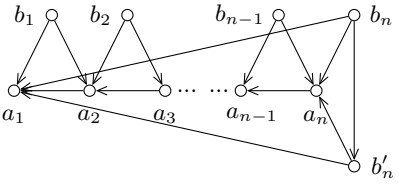


Fig. 4. Membership digraphs F_n, $n \geqslant 3$; notice that F_n is acyclic and extensional

Indeed, given F_n and F_m, with $n < m$, and supposing that η is a weak immersion of F_n into F_m, observe that $\eta(b_n)$ must be b_m, since b_n has 3 out-neighbors in F_n and b_m is the only vertex of F_m having more than 2 out-neighbors. Then, $\eta(a_1) = a_1$ and $\eta(b'_n) = b'_m$. Next, $\eta(b_1) = b_1$, since otherwise the directed paths $\eta(b_1a_1)$ and $\eta(a_2a_1)$ would have to share the arc a_2a_1, against the fact that η is an immersion. This implies that $\eta(a_2) = a_2$. By a similar argument, inductively, $\eta(b_i) = b_i$ and $\eta(a_{i+1}) = a_{i+1}$ hold for all $1 \leqslant i < n$. Moreover, for all $1 \leqslant i < n$, the image of the arc b_ia_i is the 2-vertex directed path (b_i, a_i), and the image of $a_{i+1}a_i$ is the directed path (a_{i+1}, a_i). At this point, the arc b_na_n must be mapped to the directed path $(b_m, a_m, a_{m-1}, \ldots, a_n)$, and b'_na_n is mapped to $(b'_m, a_m, a_{m-1}, \ldots, a_n)$, contradicting the arc-disjointness of η.

3 Slim Sets and Discrimination

Given a slim set x and a subset $\mathcal{F} \subseteq \mathsf{TrCl}(x)$, we will characterize in this section the elements of $\bigcup\mathcal{F}$. In particular, Lemma 4 tackles the case when $\mathcal{F} = \mathsf{TrCl}(x)^{=r}$, for some rank r. To begin with, consider a slightly weaker version of slimness.

Definition 5. *Let $\mathcal{F}$ be a finite family of sets. A $\mathbf{z} \in \bigcup\mathcal{F}$ is said to be redundant if $|\mathcal{F}| = |\{\mathbf{v} \setminus \{\mathbf{z}\} \mid \mathbf{v} \in \mathcal{F}\}|$. We say that $\mathcal{F}$ is irredundant if no element of $\bigcup\mathcal{F}$ is redundant.*

Plainly, a slim set $\mathcal{F}$ whose elements have the same rank is irredundant, since if there would exist a redundant $\mathbf{z} \in \bigcup\mathcal{F}$ so that $\mathbf{z} \in \mathbf{v} \in \mathcal{F}$, then the removal of the arc $\mathbf{vz}$ from $G_{\mathsf{TrCl}(\mathcal{F})}$ would maintain the resulting digraph extensional.

If $\mathcal{F}$ is irredundant then $\bigcup\mathcal{F}$ is also called a *minimal differentiating set for* $\mathcal{F}$. The following lemma puts a bound on the size of minimal differentiating sets.

Lemma 1 ([12,4]). *Given an n-element irredundant family $\mathcal{F}$,*

$$\left|\bigcup\mathcal{F}\right| \leqslant n - 1.$$

Lemma 2 (Discrimination lemma [11]). *Given an n-element nonempty family $\mathcal{F} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$, we can determine $\mathbf{z}_1, \ldots, \mathbf{z}_k \in \bigcup\mathcal{F}$, with $k \leqslant n - 1$, so that the family*

$$\{\mathbf{v}_i \cap \{\mathbf{z}_1, \ldots, \mathbf{z}_k\} \mid \mathbf{v}_i \in \mathcal{F}\}$$

is irredundant and has cardinality n.

Lemma 3 (Slim discrimination lemma). *Given an n-element slim and nonempty family $\mathcal{F} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ such that $\emptyset \notin \mathcal{F}$, we can determine $\mathbf{z}_1, \ldots, \mathbf{z}_k \in \bigcup \mathcal{F}$, with $k \leqslant n$, so that:*

(1) *the family $\{\mathbf{v}_i \cap \{\mathbf{z}_1, \ldots, \mathbf{z}_k\} \mid \mathbf{v}_i \in \mathcal{F}\} \cup \{\emptyset\}$ is irredundant, of cardinality $n+1$ and*
(2) *if $\mathsf{rk}(\mathcal{F}) = \varrho + 2$ and there is a $1 \leqslant j \leqslant n$ so that $|\mathbf{v}_j \cap \{\mathbf{z}_1, \ldots, \mathbf{z}_k\}| > 1$, then $|\{\mathbf{z}_i \mid \mathsf{rk}(\mathbf{z}_i) = \varrho\}| < n$.*

Proof. We reason by induction on n. If $n = 1$ the claim is clear, since $\mathbf{v_1} \neq \emptyset$ and we can thus find $\mathbf{z_1} \in \mathbf{v_1}$ to satisfy (1) and (2).

If $n > 1$, apply the discrimination lemma to $\mathcal{F} \cup \{\emptyset\}$ and let $\mathbf{z}_1, \ldots, \mathbf{z}_k \in \bigcup \mathcal{F}$, with $k \leqslant n$, so that (1) holds for $\mathbf{z}_1, \ldots, \mathbf{z}_k$. If there is a $\mathbf{v}_j$ with $\mathsf{rk}(\mathbf{v}_j) \leqslant \varrho$, then, since $\mathbf{v}_j \cap \{\mathbf{z}_1, \ldots, \mathbf{z}_k\} \neq \emptyset$, one of $\mathbf{z}_1, \ldots, \mathbf{z}_k$ is of rank strictly smaller than ϱ, and we are done. Therefore, we can assume that $k = n$, and that for all $1 \leqslant j \leqslant n$, $\mathsf{rk}(\mathbf{v}_j) = \varrho + 1$ and $\mathsf{rk}(\mathbf{z}_j) = \varrho$.

Accordingly, suppose for a contradiction that there exists $1 \leqslant j \leqslant n$ so that

$$\mathbf{v}_j \cap \{\mathbf{z}_1, \ldots, \mathbf{z}_n\} = \{\mathbf{z}'_1, \ldots, \mathbf{z}'_l\}, \text{ where } l \geqslant 2.$$

Denote by $P(\mathbf{v}_j)$ the set

$$P(\mathbf{v}_j) =_{\text{def}} \{\mathbf{v} \subseteq \mathbf{v}_j \mid \mathbf{v} \cap \{\mathbf{z}'_1, \ldots, \mathbf{z}'_l\} \neq \emptyset\}$$

and observe that any element of $P(\mathbf{v}_j)$ belongs to $\mathcal{F}$, since $\mathsf{rk}(\mathbf{z}'_1) = \cdots = \mathsf{rk}(\mathbf{z}'_l) = \varrho$ and $\mathcal{F}$ is slim. Consider now $\mathcal{F}' =_{\text{def}} \mathcal{F} \setminus P(\mathbf{v}_j)$ and $\mathcal{Z}' =_{\text{def}} \{\mathbf{z}_1, \ldots, \mathbf{z}_n\} \setminus \{\mathbf{z}'_1, \ldots, \mathbf{z}'_l\}$, which entails $\mathcal{Z}' \subseteq \bigcup \mathcal{F}'$. Since $l \geqslant 2$, $|P(\mathbf{v}_j)| \geqslant 2^l - 1 > l$, and thus $|\mathcal{F}'| < |\mathcal{Z}'| < n$. This implies that there exists $\mathbf{v}' \in \mathcal{F}'$ with $|\mathbf{v}' \cap \mathcal{Z}'| > 1$. At this point, we can repeatedly apply the above argument to the strictly smaller finite sets $\mathcal{F}'$ and $\mathcal{Z}'$, which brings the desired contradiction. □

Lemma 4 (Slim structure lemma). *For any slim set x and for every $0 < r < \mathsf{rk}(x)$, the following properties hold:*

(1) $|\mathsf{TrCl}(x)^{=r-1} \setminus x| \leqslant |\mathsf{TrCl}(x)^{=r}|$;
(2) $|\mathsf{TrCl}(x)^{=r}| \leqslant |x^{\geqslant r}|$.

Proof. To prove (1), assume for a contradiction that $\mathbf{r}$ is a rank such that $|\mathsf{TrCl}(x)^{=\mathbf{r}-1} \setminus x| > |\mathsf{TrCl}(x)^{=\mathbf{r}}|$; this entails $\mathbf{r} < \mathsf{rk}(x) - 1$, since $\mathsf{TrCl}(x)^{=\mathsf{rk}(x)-1} \setminus x = \emptyset$.

We claim that there exists a $\mathbf{z} \in \mathsf{TrCl}(x)^{=\mathbf{r}-1} \setminus x$ which is not an element of any $y \in \mathsf{TrCl}(x)^{=\mathbf{r}}$. If this were not to hold, then by the discrimination lemma applied to the family $\mathsf{TrCl}(x)^{=\mathbf{r}} \cup \{\emptyset\}$ we obtain $\mathbf{z}_1, \ldots, \mathbf{z}_k$, $k \leqslant |\mathsf{TrCl}(x)^{=\mathbf{r}}|$, so that the family $\{v \cap \{\mathbf{z}_1, \ldots, \mathbf{z}_k\} \mid v \in \mathsf{TrCl}(x)^{=\mathbf{r}}\} \cup \{\emptyset\}$ is irredundant, of cardinality $|\mathsf{TrCl}(x)^{=\mathbf{r}}| + 1$. Therefore, there exists $\mathbf{t} \in \left(\mathsf{TrCl}(x)^{=\mathbf{r}-1} \setminus x\right) \setminus \{\mathbf{z}_1, \ldots, \mathbf{z}_k\}$ so that any arc from any element $y \in \mathsf{TrCl}(x)^{=\mathbf{r}}$ to $\mathbf{t}$ can be removed from $G_{\mathsf{TrCl}(x)}$ without interfering with its extensionality. This contradicts the slimness of x.

Accordingly, taking $\mathbf{z} \in \mathsf{TrCl}(x)^{=\mathbf{r}-1} \setminus x$ which is not an element of any $y \in \mathsf{TrCl}(x)^{=\mathbf{r}}$, there must be a $w \in \mathsf{TrCl}(x)^{>\mathbf{r}}$ so that $\mathbf{z} \in w$; take such a $\mathbf{w}$ to

be $\subseteq$-minimal in $\mathsf{TrCl}(x)^{>\mathbf{r}}$. Since $\mathsf{rk}(\mathbf{w}) > \mathbf{r}$, there must exist $\mathbf{u} \in \mathbf{w}$ of rank greater than or equal to $\mathbf{r}$. From the minimality of $\mathbf{w}$, $\mathsf{rk}(\mathbf{w} \setminus \{\mathbf{u}\}) = \mathbf{r}$. Since no $y \in \mathsf{TrCl}(x)^{=\mathbf{r}}$ contains $\mathbf{z}$ as element, we have that $\mathbf{w} \setminus \{\mathbf{u}\} \notin \mathsf{TrCl}(x)$. Thus, removing the arc $\mathbf{wu}$ from $G_{\mathsf{TrCl}(x)}$ maintains its extensionality, which is against the slimness of x.

To see that (2) holds as well, proceed again by contradiction and take $\mathbf{r}$ maximum such that $|\mathsf{TrCl}(x)^{=\mathbf{r}}| > |x^{\geqslant \mathbf{r}}|$. Observe that $\mathbf{r} < \mathsf{rk}(x) - 1$, since otherwise $\mathsf{TrCl}(x)^{=\mathsf{rk}(x)-1} = x^{\geqslant \mathsf{rk}(x)-1}$. The choice of $\mathbf{r}$ implies that $|\mathsf{TrCl}(x)^{=\mathbf{r}+1}| \leqslant |x^{\geqslant \mathbf{r}+1}|$. From (1) we thus have

$$|\mathsf{TrCl}(x)^{=\mathbf{r}} \setminus x| \leqslant |\mathsf{TrCl}(x)^{=\mathbf{r}+1}| \leqslant |x^{\geqslant \mathbf{r}+1}|,$$

which brings the desired contradiction, since

$|\mathsf{TrCl}(x)^{=\mathbf{r}}| \leqslant |\mathsf{TrCl}(x)^{=\mathbf{r}} \setminus x| + |x^{=\mathbf{r}}| \leqslant |x^{\geqslant \mathbf{r}+1}| + |x^{=\mathbf{r}}| = |x^{\geqslant \mathbf{r}}|$. □

4 Encoding

Let us begin by considering the set of $\in$-digraphs of slim sets of bounded cardinality and skewness:

$$\mathsf{M}_h^s = \{G_{\mathsf{TrCl}(x)} \mid x \text{ is slim} \wedge |x| \leqslant s \wedge \mathsf{skewness}(x) \leqslant h\}.$$

Point (2) of the slim structure lemma implies that at any rank of $G_{\mathsf{TrCl}(x)} \in \mathsf{M}_h^s$ there are at most s vertices. Point (1) implies that the number of vertices at any given rank r of $G_{\mathsf{TrCl}(x)}$ is non-increasing (for decreasing r), save for at most $|x| \leqslant s$ times, when a bounded number of sources (which are elements of x) appear in $G_{\mathsf{TrCl}(x)}$. Additionally, from the slim discrimination lemma it follows that whenever $x \cap \mathsf{TrCl}(x)^{=r} = \emptyset$ and $|\mathsf{TrCl}(x)^{=r+1}| = |\mathsf{TrCl}(x)^{=r}|$, then each element of $\mathsf{TrCl}(x)^{=r+1}$ has cardinality 1, and thus has exactly one out-going arc towards an element of $\mathsf{TrCl}(x)^{=r}$, and conversely, each element of $\mathsf{TrCl}(x)^{=r}$ is the out-neighbor of exactly one element of $\mathsf{TrCl}(x)^{=r+1}$.

Let us say that a rank r is a *juncture rank* of a slim set x if there is a source at rank r in $G_{\mathsf{TrCl}(x)}$, or there is a change in cardinality when passing from $\mathsf{TrCl}(x)^{=r}$ to $\mathsf{TrCl}(x)^{=r-1}$. For uniformity, we also consider 0 to be a juncture rank. The previous two observations imply that two factors need to be taken into account in order to show that strong immersion is a wqo on M_h^s.

First, the subdigraphs of $G_{\mathsf{TrCl}(x)}$ induced by the vertices at a juncture rank and by their out-neighbors can be encoded by characters drawn from an alphabet $\Sigma_{s,h}$ consisting of all such membership digraphs. On the other hand, in order to manage the directed paths linking these subdigraphs, it suffices to encode their lengths and their endpoints. We will use natural numbers for the first item, while for the second, we will require that the membership digraphs from the alphabet $\Sigma_{s,h}$ have their sinks labeled. Entering into details, $\Sigma_{s,h}$ consists of all pairs (D, λ) satisfying the following properties:

(i) D is a digraph, which is acyclic and *weakly extensional*, in the sense that for any distinct vertices u and v in $V(D)$, if u is not a sink, then $N^+(u) \neq N^+(v)$;

(ii) removing any arc from D either increases the number of sinks of D or makes it no longer weakly extensional;
(iii) D has at most s sources;
(iv) for any $u, v \in V(D)$ with $v \in N^+(u)$ it holds that $\mathsf{rk}(u) - \mathsf{rk}(v) \leqslant h$;
(v) if all vertices of D at some rank r have exactly one out-neighbor, then either
  (a) there is a source of D of rank r, or
  (b) there exist $u \in V(D)$ and $v \in N^+(u)$ such that $\mathsf{rk}(v) < r < \mathsf{rk}(u)$;
(vi) denoting by T the set of sinks of D, λ is a label-assigning bijection $\lambda : T \to \{1, \ldots, |T|\}$.

Given a pair $(\mathcal{D}, \lambda) \in \Sigma_{s,h}$, we can define an Ackermann-line order on the vertices of $\mathcal{D}$ in the following recursive way:

- for any sinks u, v of $\mathcal{D}$, let $u \prec_{(\mathcal{D},\lambda)} v \Leftrightarrow_{\text{def}} \lambda(u) < \lambda(v)$;
- for any sink u of $\mathcal{D}$ any any other non-sink vertex v of $\mathcal{D}$, set $u \prec_{(\mathcal{D},\lambda)} v$;
- for any non-sink vertices u, v of $\mathcal{D}$, let $u \prec_{(\mathcal{D},\lambda)} v \Leftrightarrow_{\text{def}} \max_{\prec_{(\mathcal{D},\lambda)}}(u \backslash v) \prec_{(\mathcal{D},\lambda)} \max_{\prec_{(\mathcal{D},\lambda)}}(v \setminus u)$.

Given a slim set x having $|x| \leqslant s$ and $\mathsf{skewness}(x) \leqslant h$, we give two encodings for it: $\sigma(x)$, a string over the alphabet $\Sigma_{s,h}$, and $\delta(x)$, a string of positive integers. They are obtained in the following algorithmic way. First, compute Ackermann's order $\prec_{\mathsf{A}}$ on the elements of $\mathsf{TrCl}(x)$. Put also $\varrho = 1 + \max\{\mathsf{rk}(y) \mid y \in \mathsf{TrCl}(x)\}$.

The ith ($i = 1, 2, ...$) characters of $\sigma(x)$ and $\delta(x)$ are obtained as follows. Let r be the greatest juncture rank of x satisfying $r < \varrho$, and let r' be the smallest rank such that $r' \leqslant r$, and the subdigraph induced by the vertices in $\mathsf{TrCl}(x)^{=r} \cup \mathsf{TrCl}(x)^{=r-1} \cup \cdots \cup \mathsf{TrCl}(x)^{=r'}$, which we denote by D_i, is isomorphic to a digraph appearing in a pair of the alphabet $\Sigma_{s,h}$. Then, the ith character of $\sigma(x)$ is that pair $(\mathcal{D}, \lambda)$ such that there exists an isomorphism $f : D_i \to \mathcal{D}$ with the additional property that for any sinks $u, v \in V(D_i)$ it holds $u \prec_{\mathsf{A}} v \Leftrightarrow \lambda(f(u)) < \lambda(f(v))$. It thus holds that for any u, v sources of D_i, we have $u \prec_{\mathsf{A}} v \Leftrightarrow f(u) \prec_{(\mathcal{D},\lambda)} f(v)$.

Moreover, set the ith character of $\delta(x)$ to be $\varrho - r$. Next, update ϱ to be r' and go on to computing the $(i+1)$th characters of $\sigma(x)$ and $\delta(x)$. Observe that the strings $\sigma(x)$ and $\delta(x)$ have the same lengths, and that, for uniformity, we have chosen to always set the first character of $\delta(x)$ as 1.

An example of a membership digraph of a slim set x is given in Fig. 5, where the digraphs appearing in the characters of $\sigma(x)$ are marked in gray.

5 Main Result

The proof that strong immersion if a wqo on M_h^s proceeds as follows. First, we argue that for any such slim set x of bounded cardinality and skewness, the length of the string $\sigma(x)$ is bounded. Thus, given an infinite sequence $(x_i)_{i=1,2,\ldots}$ of such sets, we can find an infinite subsequence $(x_{i_j})_{j=1,2,\ldots}$ such that $\sigma(x_{i_1}) = \sigma(x_{i_2}) = \ldots$. This implies that between the membership digraphs corresponding to two consecutive characters of $\sigma(x_{i_j})$, for any j, we have the same number of directed paths. Observe also that $|\delta(x_{i_1})| = |\delta(x_{i_2})| = \ldots$.

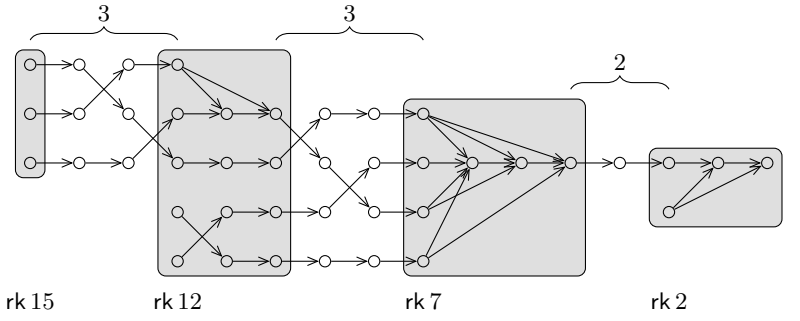


Fig. 5. The membership digraph of a slim set x of cardinality 6 and skewness 3; we have $\delta(x) = 1\ 3\ 3\ 2$

However, this not yet sufficient, since the directed paths between these digraphs can have arbitrary lengths. For this reason, we use a basic fact from the theory of wqo's (see e.g. [7]). If $(Q, \preccurlyeq)$ is a wqo, then so is the set of fixed length sequences over Q, componentwise ordered by $\preccurlyeq$. That is, the pair $(Q^\ell, \preccurlyeq^\ell)$ is a wqo, where for any $(x_1, \ldots, x_\ell), (y_1, \ldots, y_\ell) \in Q^\ell$ we have $(x_1, \ldots, x_\ell) \preccurlyeq^\ell (y_1, \ldots, y_\ell) \Leftrightarrow_{\text{def}} x_i \preccurlyeq y_i$, for all $1 \leqslant i \leqslant \ell$. Since $(\mathbb{N}, \leqslant)$ is a wqo, this implies that we can find x_{i_j} and x_{i_k} in the aforementioned infinite sequence such that taking $\ell = |\delta(x_{i_1})|$ we have $\delta(x_{i_j}) \leqslant^\ell \delta(x_{i_k})$. To ensure that these directed paths also have corresponding endpoints, we will finally make use of the labeling of the sinks, obtained by Ackermann's order.

Lemma 5. *For any $s, h \geqslant 1$, there exists a computable function $g(s,h)$ such that for any slim set x, with $|x| \leqslant s$ and $\mathsf{skewness}(x) \leqslant h$, it holds $|\sigma(x)| \leqslant g(s,h)$.*

Proof. We first argue that the cardinality of $\Sigma_{s,h}$ is finite. Given $(\mathcal{D}, \lambda) \in \Sigma_{s,h}$, observe that point (2) of the slim structure lemma entails that the number of vertices of $\mathcal{D}$ at a given rank is bounded by s. Then, for any t, $1 \leqslant t \leqslant s$, there are at most sh ranks r with the property that $\mathcal{D}$ has exactly t vertices of rank r. Indeed, (ii), (iv) and the slim structure lemma imply that there can be at most h consecutive ranks at which there are exactly t vertices of $\mathcal{D}$. Moreover, this cardinality t of vertices at the same rank can repeat itself for at most s non-consecutive ranks, the culprits being the sources of $\mathcal{D}$, which are at most s. To conclude, $\mathcal{D}$ can have at most $sh \sum_{t=1}^{s} t$ vertices, and hence the cardinality of $\Sigma_{s,h}$ is bounded by $s! 3^{\binom{\frac{1}{2}s^2(s+1)h}{2}}$, since there can be at most s sinks in a character of $\Sigma_{s,h}$, and $s!$ distinct ways to label them.

Observe now that a character inside the encoding $\sigma(x)$ of a slim set x can appear more than once, but only because of a source of $G_{\mathsf{TrCl}(x)}$ (that is, one of the elements of x). Since the cardinality of x is bounded by s, then $g(s,h)$ can be taken to be $s|\Sigma_{s,h}|$. □

Theorem 1. *The pair $(\mathsf{M}_h^s, \preccurlyeq_i^s)$ is a wqo.*

Proof. Let $(G_{\mathsf{TrCl}(x_i)})_{i=1,2,\ldots}$ be an infinite sequence of membership digraphs belonging to M^s_h. By Lemma 5, there exists an infinite subsequence of it, $(G_{\mathsf{TrCl}(x_{i_j})})_{j=1,2,\ldots}$, such that $\sigma(x_{i_1}) = \sigma(x_{i_2}) = \ldots$. As noted before, $|\sigma(x_{i_j})| = |\delta(x_{i_j})|$ holds for any j, and if we denote their length by ℓ, we have that there exists $1 \leqslant j < k$ such that for $\mathbf{y} =_{\text{def}} x_{i_j}$ and $\mathbf{z} =_{\text{def}} x_{i_k}$, we have $\delta(\mathbf{y}) \leqslant^{\ell} \delta(\mathbf{z})$.

Given a string σ, denote by $\sigma[m]$ its mth character, $1 \leqslant m \leqslant |\sigma|$. Let $(\mathcal{D}_m, \lambda_m) =_{\text{def}} \sigma(\mathbf{y})[m]$, for any $1 \leqslant m \leqslant \ell$. Let also $r(m)$ be the juncture rank of $\mathbf{y}$ responsible for the introduction of the mth character of $\sigma(\mathbf{y})$. Recall that $r(m) > r(m+1)$, for any $1 \leqslant m < \ell$. We will show by induction on t, $0 \leqslant t < \ell$, that we have a strong immersion of the subdigraph of $G_{\mathsf{TrCl}(\mathbf{y})}$ induced by the elements of $\mathsf{TrCl}(\mathbf{y})^{\leqslant r(\ell-t)}$ into $G_{\mathsf{TrCl}(\mathbf{z})}$.

The subdigraph induced by $\mathsf{TrCl}(\mathbf{y})^{\leqslant r(\ell)}$ is isomorphic to $\mathcal{D}_\ell$, and hence isomorphic to $\mathsf{TrCl}(\mathbf{z})^{\leqslant r(\ell)}$, which proves our claim for $t = 0$.

Assume now that η is a strong immersion of $\mathsf{TrCl}(\mathbf{y})^{\leqslant r(\ell-t)}$ into $G_{\mathsf{TrCl}(\mathbf{z})}$, for $t < \ell - 1$. Construct the strong immersion η' of $\mathsf{TrCl}(\mathbf{y})^{\leqslant r(\ell-(t+1))}$ into $G_{\mathsf{TrCl}(\mathbf{z})}$ as follows. For any vertex in $\mathsf{TrCl}(\mathbf{y})^{\leqslant r(\ell-t)}$ and any arc between two such vertices, let η' coincide with η. The elements of $\mathsf{TrCl}(\mathbf{y})^{\leqslant r(\ell-(t+1))} \setminus \mathsf{TrCl}(\mathbf{y})^{\leqslant r(\ell-t)}$ can be divided into those vertices forming the subdigraph $D^{\mathbf{y}}$ of $G_{\mathsf{TrCl}(\mathbf{y})}$ isomorphic to $\mathcal{D}_{\ell-(t+1)}$, and those vertices forming the subdigraph $P^{\mathbf{y}}$ of $G_{\mathsf{TrCl}(\mathbf{y})}$ consisting of vertex-disjoint directed paths linking the sinks of $D^{\mathbf{y}}$ to vertices at juncture rank $r(\ell - t)$.

Map the vertices of $D^{\mathbf{y}}$ to those vertices of $G_{\mathsf{TrCl}(\mathbf{z})}$ to which they are mapped according to the isomorphisms which gave $\sigma(\mathbf{y})[\ell - (t+1)] = \sigma(\mathbf{z})[\ell - (t+1)]$. To be more precise, denote by $D^{\mathbf{z}}$ the subdigraph of $G_{\mathsf{TrCl}(\mathbf{z})}$ responsible for the choice of $(\mathcal{D}_{\ell-(t+1)}, \lambda_{\ell-(t+1)})$ as the $(\ell - (t+1))$th character of $\sigma(\mathbf{z})$. Denote also by $f_{\mathbf{y}} : D^{\mathbf{y}} \to \mathcal{D}_{\ell-(t+1)}$ and by $f_{\mathbf{z}} : D^{\mathbf{z}} \to \mathcal{D}_{\ell-(t+1)}$ the corresponding label-preserving isomorphisms. Then, let $\eta'(v) = f_{\mathbf{z}}^{-1}(f_{\mathbf{y}}(v))$ and $\eta'(uv) = (f_{\mathbf{z}}^{-1}(f_{\mathbf{y}}(u)), f_{\mathbf{z}}^{-1}(f_{\mathbf{y}}(v)))$ for any $v, u \in V(D^{\mathbf{y}})$.

Let now $P^{\mathbf{z}}$ be the induced subdigraph of $G_{\mathsf{TrCl}(\mathbf{z})}$ consisting of those vertex-disjoint paths between the sinks of $D^{\mathbf{z}}$ to the vertices of $G_{\mathsf{TrCl}(\mathbf{z})}$ at that juncture rank responsible for the introduction of the $(\ell - (t+1))$th character of $\sigma(\mathbf{z})$. Clearly, $P^{\mathbf{y}}$ and $P^{\mathbf{z}}$ consist of the same number of vertex disjoint paths, since $\sigma(\mathbf{y}) = \sigma(\mathbf{z})$. Moreover, all paths of $P^{\mathbf{y}}$ and $P^{\mathbf{z}}$, have the same lengths, respectively. Additionally, since $\delta(\mathbf{y}) \leqslant^{\ell} \delta(\mathbf{z})$, we have that all paths of $P^{\mathbf{y}}$ have length less than, or equal, to all paths of $P^{\mathbf{z}}$. It remains to show that their endpoints correspond.

From the way $\sigma(\mathbf{y})$ and $\sigma(\mathbf{z})$ were constructed, for any sinks u, v of $D^{\mathbf{y}}$ it holds that $u \prec_{\mathsf{A}} v \Leftrightarrow \eta'(u) \prec_{\mathsf{A}} \eta'(v)$. Similarly, for any vertices u, v of $G_{\mathsf{TrCl}(\mathbf{y})}$ at juncture rank $r(\ell - t)$, $u \prec_{\mathsf{A}} v \Leftrightarrow \eta'(u) \prec_{\mathsf{A}} \eta'(v)$ also holds. Consider now a directed path $p^{\mathbf{y}}$ of $P^{\mathbf{y}}$ with endpoints v, a sink of $D_{\mathbf{y}}$, and w, a vertex of rank $r(\ell - t)$. From the above two observations and the definition of Ackermann's order, it follows that in $P^{\mathbf{z}}$ there is a directed path $p^{\mathbf{z}}$ with endpoints $\eta'(v)$ and $\eta'(w)$. Since $|p^{\mathbf{y}}| \leqslant |p^{\mathbf{z}}|$, $p^{\mathbf{y}}$ can thus be strongly immersed into $p^{\mathbf{z}}$ by an extension of η'. □

6 Conclusion

We consider the results of this paper a first step in studying digraph immersion and well-quasi-orders for various classes of sets. It is thus of particular interest to what extent the constraints considered here can be weakened. Observe, however, that dropping the bounds on cardinality and on skewness at the same time no longer ensures the wqo property for slim sets, since the membership digraphs F_n in Fig. 4, which have unbounded cardinality and skewness, can be easily rendered slim (it suffices to add 4 vertices with sets of out-neighbors $\{a_n\}$, $\{b'_n\}$, $\{a_n, b'_n\}$, and $\{a_1, b'_n\}$, respectively).

It would be interesting to know if the same strategy employed in this paper could be extended to prove that the collection of hereditarily finite slim sets of bounded skewness (with no bound on the cardinality) admits a wqo. In order to do this a generalization of the results in Sec. 5 dealing with alphabets $\Sigma_{s,h}$ for various values of s would be sufficient.

References

1. Abdulla, P., Delzanno, G., Van Begin, L.: A classification of the expressive power of well-structured transition systems. Information and Computation (in Press)
2. Ackermann, W.: Die Widerspruchfreiheit der allgemeinen Mengenlehre. Mathematische Annalen 114, 305–315 (1937)
3. Bang-Jensen, J., Gutin, G.: Digraphs Theory, Algorithms and Applications, 1st edn. Springer, Berlin (2000)
4. Bollobás, B.: Combinatorics. Cambridge University Press, Cambridge (1986)
5. Chudnovsky, M., Seymour, P.D.: A well-quasi-order for tournaments. J. Comb. Theory, Ser. B 101(1), 47–53 (2011)
6. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere? Theoretical Computer Science 256(1-2), 63–92 (2001)
7. Higman, G.: Ordering by divisibility in abstract algebras. Proc. London Math. Soc. 3(2), 326–336 (1952)
8. Kruskal, J.B.: Well-quasi ordering, the tree theorem and Vászonyi's conjecture. Trans. Amer. Math. Soc. 95, 210–225 (1960)
9. Levy, A.: Basic Set Theory. Springer, Berlin (1979)
10. Nash-Williams, C.S.J.A.: On well-quasi-ordering trees. In: Theory of Graphs and Its Applications (Proc. Symp. Smolenice, 1963), pp. 83–84. Publ. House Czechoslovak Acad. Sci. (1964)
11. Omodeo, E.G., Policriti, A.: The Bernays-Schönfinkel-Ramsey class for set theory: semidecidability. J. Symb. Log. 75(2), 459–480 (2010)
12. Parlamento, F., Policriti, A., Rao, K.: Witnessing Differences Without Redundancies. Proc. of the American Mathematical Society 125(2), 587–594 (1997)
13. Robertson, N., Seymour, P.: Graph minors. XX. Wagner's conjecture. J. Combin. Theory, Ser. B 92, 325–357 (2004)
14. Robertson, N., Seymour, P.: Graph minors. XXIII. Nash–Williams's immersion conjecture. J. Combin. Theory, Ser. B 100, 181–205 (2010)

On the Interval-Bound Problem for Weighted Timed Automata

Karin Quaas

Institut für Informatik, Universität Leipzig
04009 Leipzig, Germany

Abstract. A weighted timed automaton is a timed automaton equipped with weights on transitions and weight rates on locations. These weights may be positive or negative, corresponding to the production and consumption of some resources. We consider the interval-bound problem: does there exist an infinite run such that the accumulated weight for each prefix of the run is within some given bounds? We show that this problem is undecidable if the weighted timed automaton has more than one clock and more than one weight variable. We further prove that the problem is PSPACE-complete if we restrict the time domain to the natural numbers.

1 Introduction

During the last years, *weighted timed automata* [2,3] have received much attention in the real-time community. A weighted timed automaton is a timed automaton extended with weight variables, whose values may grow or fall linearly with time while being in a state, or discontinuously grow or fall during a state change. In this way, weighted timed automata can be used to model both continuous and discrete production and consumption of resources, like energy, bandwidth or money. This allows for interesting applications e.g. in operations research, in particular, *optimal* scheduling.Recently [5], three interesting resource scheduling problems for weighted timed automata were introduced: the existence of an infinite run during which the values of the weight variables never fall below zero (*lower-bound*), never fall below zero and never exceed a given upper bound (*interval-bound*), and never fall below zero and never exceed a given *weak* upper bound, meaning that when the weak upper bound is reached, the value of the weight variable is not increased but maintained at this level (*lower-weak-upper-bound*). For weighted timed automata with a single clock and a single weight variable, the lower-bound-problem and the lower-weak-upper-bound-problem are decidable in polynomial time [5], albeit with the restriction that the weighted timed automaton does not allow for discontinuous updates of the weight variables during state changes. In [4] it is proved that the lower-bound-problem is also decidable if this restriction is lifted and if the values of the weight variables may not only change linearly but also exponentially in time. The interval-bound-problem is undecidable for weighted timed automata with a single clock and a

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 452–464, 2011.

single weight variable *in a game setting*; in the corresponding existential setting, however, the problem is open to the best of our knowledge.

In this paper, we show that the interval-bound-problem is undecidable for weighted timed automata with two clocks and two weight variables (or more). The proof is a reduction from the infinite computation problem for two-counter machines [8]. The undecidability proof does not require discrete updates of the weight variables in the automaton. As a second main result, we show that the interval-bound-problem is PSPACE-complete if we restrict the time domain to the natural numbers. This result is irrespective of the number of clocks and weight variables. The proof for PSPACE-membership is based on a polynomial-time reduction to the recurrent reachability problem for timed automata [7,11]. These two results are a big step towards the precise decidability border for the interval-bound problem for weighted timed automata.

2 Preliminaries

We let $\mathbb{Z}$ and $\mathbb{R}$ denote the set of integers and reals, respectively, and we let $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ be the set of *positive* integers and reals, respectively. We let $\mathbb{T} \in \{\mathbb{Z}, \mathbb{R}\}$, and define $\mathbb{T}_{\geq 0}$ to be the set $\{x \in \mathbb{T} \mid x \geq 0\}$.

Let $\mathcal{X}$ be a finite set of *clock variables* ranging over $\mathbb{T}_{\geq 0}$. We define *clock constraints* ϕ over $\mathcal{X}$ to be conjunctions of formulas of the form $x \sim c$, where $c \in \mathbb{N}$, $x \in \mathcal{X}$ and $\sim \in \{<, \leq, =, \geq, >\}$. Let $\Phi(\mathcal{X})$ be the set of all clock constraints over $\mathcal{X}$. A *clock valuation* $\nu : \mathcal{X} \to \mathbb{T}_{\geq 0}$ is a function that assigns a value to each clock variable. A clock valuation ν satisfies a clock constraint ϕ, written $\nu \models \phi$, if ϕ evaluates to true according to the values given by ν. For $\delta \in \mathbb{T}_{\geq 0}$ and $\lambda \subseteq \mathcal{X}$, we define the clock valuation $\nu + \delta$ to be $(\nu + \delta)(x) = \nu(x) + \delta$ for each $x \in \mathcal{X}$ and the clock valuation $\nu[\lambda := 0]$ by $(\nu[\lambda := 0])(x) = 0$ if $x \in \lambda$ and $(\nu[\lambda := 0])(x) = \nu(x)$ otherwise.

Let $\mathcal{W}$ be a finite set of *weight variables* ranging over $\mathbb{T}$. A *weight valuation* $\mu : \mathcal{W} \to \mathbb{T}$ is a function that assigns a value to each weight variable.

A *weighted timed automaton* over time domain $\mathbb{T}$ is a tuple

$$\mathcal{A} = (\mathcal{L}, l_0, \mathcal{X}, \mathcal{W}, E, \iota, \mathsf{ewt}, \mathsf{lwt}),$$

where

- $\mathcal{L}$ is a finite set of locations,
- $l_0 \in \mathcal{L}$ is an initial location,
- $\mathcal{X}$ is a finite set of clock variables,
- $\mathcal{W}$ is a finite set of weight variables,
- $E \subseteq \mathcal{L} \times \Phi(X) \times 2^{\mathcal{X}} \times \mathcal{L}$ is a finite set of edges,
- $\iota \in \mathbb{N}^{\mathcal{W}}$ is an initial weight function,
- $\mathsf{ewt} : E \to \mathbb{Z}^{\mathcal{W}}$ is a function assigning a weight function to each edge,
- $\mathsf{lwt} : \mathcal{L} \to \mathbb{Z}^{\mathcal{W}}$ is a function assigning a weight rate function to each location.

A weighted timed automaton does not allow for discrete updates of its weight variables if $\mathsf{ewt}(e) = \{0\}^{\mathcal{W}}$ for each $e \in E$. A *timed automaton* over $\mathbb{T}$ is a weighted timed automaton $\mathcal{A}$ over $\mathbb{T}$ where the set of weight variables is empty.

A *state* of a weighted timed automaton is a triple $(l, \nu, \mu) \in \mathcal{L} \times \mathbb{T}^{\mathcal{X}}_{\geq 0} \times \mathbb{T}^{\mathcal{W}}$. Between two states (l, ν, μ) and (l', ν', μ') there is a *timed* transition, written $(l, \nu, \mu) \xrightarrow{\delta} (l, \nu', \mu')$, if there is some $\delta \in \mathbb{T}_{\geq 0}$ such that $l' = l$, $\nu' = \nu + \delta$ and $\mu'(w) = \mu(w) + \mathsf{lwt}(l)(w) \cdot \delta$ for each $w \in \mathcal{W}$. Between two states (l, ν, μ) and (l', ν', μ') there is a *discrete* transition, written $(l, \nu, \mu) \xrightarrow{e} (l, \nu', \mu')$, if there is some $e = (l, \phi, \lambda, l') \in E$ such that $\nu \models \phi$, $\nu' = \nu[\lambda := 0]$ and $\mu'(w) = \mu(w) + \mathsf{ewt}(e)(w)$ for each $w \in \mathcal{W}$. A *finite run* in $\mathcal{A}$ is a sequence $\prod_{i=1}^{n} (l_{i-1}, \nu_{i-1}, \mu_{i-1}) \xrightarrow{\delta_i} (l_{i-1}, \nu'_i, \mu'_i) \xrightarrow{e_i} (l_i, \nu_i, \mu_i)$ of alternating timed and discrete transitions for some $n \in \mathbb{N}$. An *infinite run* in $\mathcal{A}$ is an infinite sequence $\prod_{i=0}^{\infty} (l_{i-1}, \nu_{i-1}, \mu_{i-1}) \xrightarrow{\delta_i} (l_{i-1}, \nu'_i, \mu'_i) \xrightarrow{e_i} (l_i, \nu_i, \mu_i)$ of alternating timed and discrete transitions. We say that a run is *initialized* if $\nu_0(x) = 0$ for each $x \in \mathcal{X}$ and $\mu_0 = \iota$. For $b \in \mathbb{Z}^{\mathcal{W}}$, we say that a finite (or infinite, respectively) run is *b-feasible* if the run satisfies $0 \leq \mu_i(w), \mu'_i(w) \leq b(w)$ for each $w \in \mathcal{W}$ and each $i \in \{0, ...n\}$ ($i \geq 0$, respectively). We say that a state (l, ν, μ) is *reachable in* $\mathcal{A}$ if there is an initialized finite run in $\mathcal{A}$ ending in (l, ν, μ). Given a set $F \subseteq \mathcal{L}$, we say that an infinite run is *F-accepting* if some $l \in F$ occurs on it infinitely often (standard Büchi acceptance condition).

In this paper, we are interested in the following resource scheduling problem, introduced in [5].

The interval-bound problem

INPUT: A weighted timed automaton $\mathcal{A}$ over $\mathbb{T}$ with the set $\mathcal{X}$ of clock variables, the set $\mathcal{W}$ of weight variables, the edge cost function ewt and initial weight function ι, and an upper bound function $b \in \mathbb{Z}^{\mathcal{W}}$ satisfying $\iota(w) \leq b(w)$ for each $w \in \mathcal{W}$.

QUESTION: Does there exist an initialized infinite b-feasible run in $\mathcal{A}$?

3 Undecidability Result

Theorem 1. *The interval-bound problem for weighted timed automata over* $\mathbb{T}$ *is undecidable if* $\mathbb{T} = \mathbb{R}$, $|\mathcal{X}| \geq 2$, $|\mathcal{W}| \geq 2$, *and* ewt *satisfies* $\mathsf{ewt}(e) = \{0\}^{\mathcal{W}}$ *for each* $e \in E$.

Proof. The proof is a reduction from the infinite computation problem for two-counter machines [8]. A two-counter machine $\mathcal{M}$ is a finite sequence $(I_j)_{j=1}^{n}$ of instructions operating on two counters denoted by C_1 and C_2, where I_j is one of the following instructions (with $i \in \{1, 2\}$ and $j, k, m \in \{1, ..., n\}$):

increment	I_j : $C_i := C_i + 1$; `go to` I_k
zero test/decrement	I_j : `if` $C_i = 0$ `then go to` I_k `else` $C_i := C_i - 1$; `go to` I_m
stop	I_j : `stop`

A *configuration* of a two-counter machine $\mathcal{M}$ is a triple $\gamma = (J, c_1, c_2) \in \{I_1, ..., I_n\} \times \mathbb{N} \times \mathbb{N}$, where J indicates the current instruction, and c_1 and c_2 are the current values of the counters C_1 and C_2, respectively. A *computation* of $\mathcal{M}$ is a finite or infinite sequence $(\gamma_i)_{i \geq 0}$ of configurations, such that $\gamma_0 = (I_1, 0, 0)$

and γ_{i+1} is the result of executing the instruction of γ_i on γ_i for each $i \geq 0$. The infinite computation problem for two-counter machines asks, given a two-counter machine $\mathcal{M}$, whether there is an infinite computation of $\mathcal{M}$. This problem is undecidable [8].

We simulate a two-counter machine $\mathcal{M}$ by a weighted timed automaton $\mathcal{A}_{\mathcal{M}}$ with the set $\mathcal{X} = \{x, y\}$ of clock variables and the set $\mathcal{W} = \{w_1, w_2\}$ of weight variables, and an upper bound function $b \in \mathbb{Z}^{\mathcal{W}}$ with $b(w_1) = b(w_2) = 5$, and show that there is an infinite computation of $\mathcal{M}$ if and only if there exists an initialized infinite b-feasible run in $\mathcal{A}_{\mathcal{M}}$.

We use the first weight variable w_1 to encode the values of the two counters of $\mathcal{M}$. If c_1 and c_2 are the values of the two counters, then the value of w_1 is $5 - \frac{1}{2^{c_1} 3^{c_2}}$. We let $\iota(w_1) = 4$ to model that both C_1 and C_2 are initialized to 0. The second weight variable w_2 is needed for encoding the zero test instruction of $\mathcal{M}$. We let $\iota(w_2) = 0$.

The instructions of $\mathcal{M}$ are encoded as weighted timed automata widgets. We label a location l with α, β to denote that the weight rate function $\mathsf{lwt}(l)$ is defined by $\mathsf{lwt}(l)(w_1) = \alpha$ and $\mathsf{lwt}(l)(w_2) = \beta$ for some $\alpha, \beta \in \mathbb{Z}$. In the following, we let $0 < e = \frac{1}{2^{c_1} 3^{c_2}} \leq 1$ for some $c_1, c_2 \in \mathbb{N}$.

Incrementing the value of C_1. The widget for incrementing the value of C_1 is shown in Fig. 1. The idea (adopted from [5]) is to use the interval-bound of

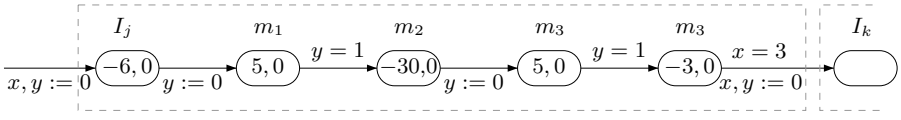


Fig. 1. Widget for the instruction $I_j : C_1 := C_1 + 1$; `go to` I_k

the value of w_1 to force the weighted timed automaton to pass exactly as much time as is needed to obtain a new value of w_1 corresponding to the increment of C_1. Note that incrementing C_1 corresponds to dividing e by 2.

Lemma 1. *Let r be a finite b-feasible run from (I_j, ν, μ) to (I_k, ν', μ') with $\mu(w_1) = 5 - e$ and $\mu(w_2) = 0$. Then $\mu'(w_1) = 5 - e/2$ and $\mu'(w_2) = 0$.*

Testing the value of C_1 against zero and decrement it. In this widget we have to check whether $c_1 = 0$, i.e., whether $e = 3^{-c_2}$ for some $c_2 \in \mathbb{N}$. The idea for the construction of this widget is adopted from [6]. We first transfer the value of e into the clock variable x and then multiply the value of x by 3. If at some point the value of x equals 1, we know that $e = 3^{-c_2}$ for some $c_2 \in \mathbb{N}$, and thus the value of C_1 is zero. If on the other hand the value of x exceeds 1, we can conclude that $e \neq 3^{-c_2}$ for all $c_2 \in \mathbb{N}$ and thus C_1 is not zero. We then reconstruct the original values of x and, depending on whether C_1 is zero or not, we either go to I_k, or we multiply the original value of x (i.e., e) by 2 and go to I_m. We use the interval-bounds of the weight variables to force the weighted

timed automaton to spend as much time as is needed for the construction. The widget is shown in Fig. 2. We remark that in the widget the clock variable y is reset at every edge. Due to lack of space, we do not display this in the picture.

Lemma 2. *Let r be a finite b-feasible run from (I_j, ν, μ) to (I_k, ν', μ') with $\mu(w_1) = 5 - e$ and $\mu(w_2) = 0$. Then $\mu'(w_1) = 5 - e$ and $e = 3^{-n}$ for some $n \in \mathbb{N}$, and $\mu'(w_2) = 0$.*

Lemma 3. *Let r be a finite b-feasible run from (I_j, ν, μ) to (I_m, ν', μ') with $\mu(w_1) = 5 - e$ and $\mu(w_2) = 0$. Then $\mu'(w_1) = 5 - 2e$ and $e \neq 3^{-n}$ for each $n \in \mathbb{N}$, and $\mu'(w_2) = 0$.*

C_2. Incrementing C_2 and testing the value of C_2 against zero and decrement it can be done in a similar way, but is not explained here due to lack of space.

Stop. The widget for the stop instruction consists of a single location l_f with no outgoing edges.

Simulation of the two-counter machine. Let $\mathcal{M} = (I_1, ..., I_n)$ for some $n \in \mathbb{N}$ be a two-counter machine. For each $i \in \{1, ..., n\}$, let $\mathcal{A}_i$ be the weighted timed automaton widget corresponding to I_i according to the constructions presented above. Now, let $\mathcal{A}_\mathcal{M}$ be the weighted timed automaton obtained by plugging the corresponding $\mathcal{A}_i$ with each other. Then the following lemma holds.

Lemma 4. *There is an infinite computation of $\mathcal{M}$ if and only if there is an initialized infinite $(5, 5)$-feasible run in $\mathcal{A}_\mathcal{M}$.*

4 Decidability Result

Theorem 2. *The interval-bound problem for weighted timed automata $\mathcal{A}$ over $\mathbb{T}$ is* PSPACE*-complete if $\mathbb{T} = \mathbb{Z}$.*

The lower bound of this problem follows immediately from the PSPACE-completeness of the *recurrent reachability problem* for timed automata over $\mathbb{Z}$ with three or more clock variables [7,1]: given a timed automaton with a designated set F of locations, is there an initialized F-accepting run? For showing that the problem is in PSPACE, we give a polynomial-time translation to the recurrent reachability problem. Let $\mathcal{A} = (\mathcal{L}, l_0, \mathcal{X}, \mathcal{W}, E, \iota, \mathsf{ewt}, \mathsf{lwt})$ be a weighted timed automaton and let $b \in \mathbb{Z}^\mathcal{W}$ be an upper bound function such that $\iota(w) \leq b(w)$ for each $w \in \mathcal{W}$. We define a timed automaton $\mathcal{A}_b$ with location set $\mathcal{L}_b \supseteq \mathcal{L}$ such that the following lemma holds:

Lemma 5. *There is an initialized infinite b-feasible run in $\mathcal{A}$ if and only if there is an initialized $\mathcal{L}$-accepting run in $\mathcal{A}_b$.*

The timed automaton $\mathcal{A}_b$ is composed of timed automata modelling the edges of $\mathcal{A}$. Let $e = (l, \phi, \lambda, l') \in E$. We define the timed automaton $\mathcal{A}_e$ to be a combination of timed automata widgets (see Fig. 5 on p.11 for an example). The main widgets are the *addition widget* and the *subtraction widget*, which may be

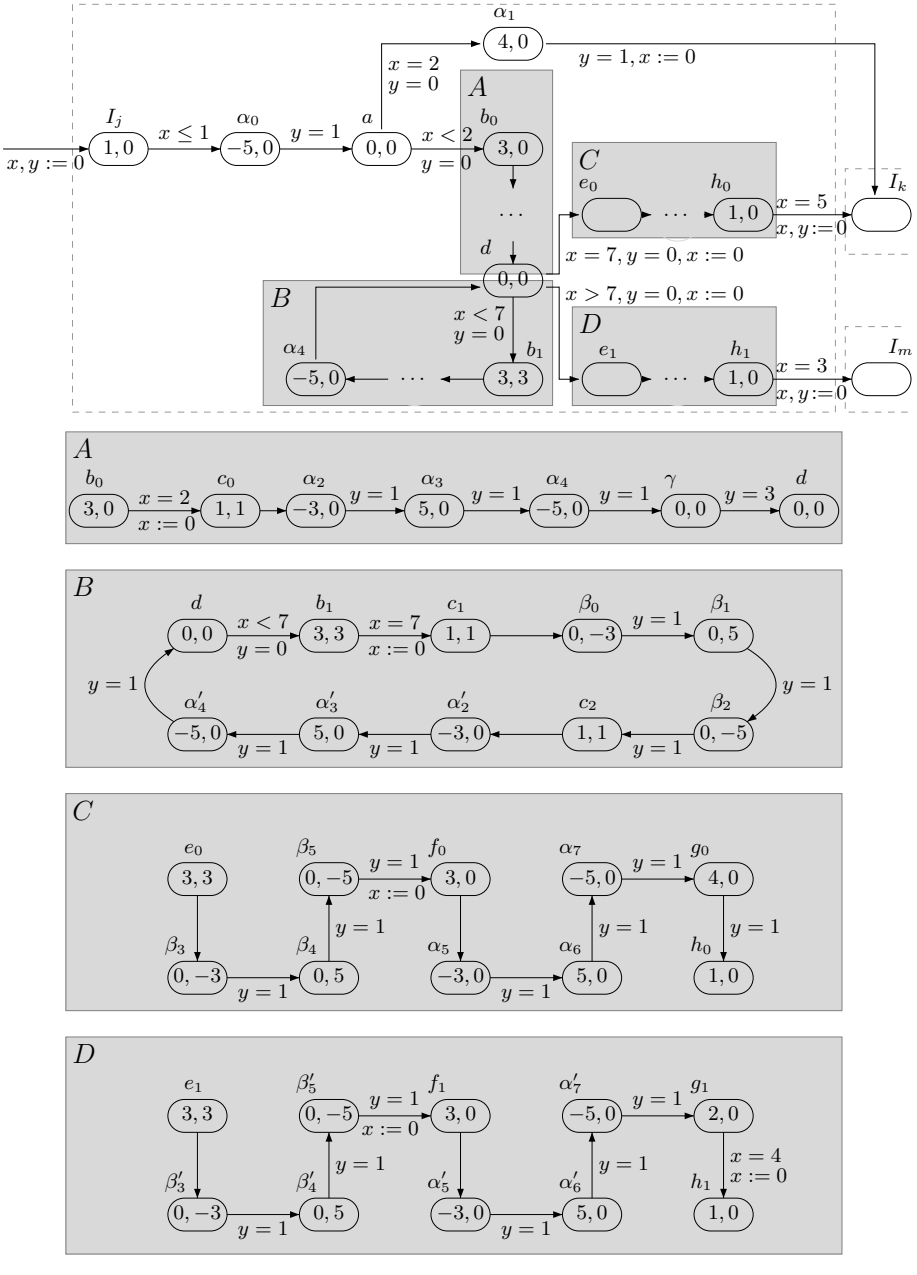


Fig. 2. Widget for the instruction I_j : `if` $C_1 = 0$ `then go to` I_k `else` $C_1 := C_1 - 1$; `go to` I_m. The clock y is reset at each edge. This is not displayed due to lack of space.

executed several times. In short, the idea of $\mathcal{A}_e$ is as follows. For each clock variable x in $\mathcal{A}$, we introduce a clock variable x' in $\mathcal{A}_e$. Likewise, for each weight variable w in $\mathcal{A}$, we introduce a clock variable w' in $\mathcal{A}_e$. We further let d, d' be clock variables representing the time delay and its copy, respectively, and y be an auxiliary clock variable. The purpose of $\mathcal{A}_e$ is to model the elapse of time in the source location of the edge followed by the execution of e in $\mathcal{A}$. If in $\mathcal{A}_e$ we let δ time units pass in l, the values of both the clock variables x' and w' are increased by δ. However, since w' represents the weight variable w, we want its value to be the sum of its old value and $\mathsf{lwt}(l)(w) \cdot \delta$. For positive weight rates, this is done in the addition widget: We read out the value of δ (which is stored in the clock variable d), and add the product of $\mathsf{lwt}(l)(w) - 1$ and δ to the value of w', so that finally the value of w' is as wanted. Negative weight rates are treated similarly using the subtraction widget. Thereafter, we add the edge cost $\mathsf{ewt}(e)(w)$ to the value of w' using another instance of the addition widget or the subtraction widget, respectively. In between, we check whether the value of w' satisfies the interval-bound. If this is the case, we reset the values of the clock variables according to λ and go to l'. If not, the timed automaton is stuck in a dead end.

In all widgets, we have to read out and change the values of some clock variables. One difficulty is to maintain the original values of the other clock variables. For the widgets, we need several bounds on the values of clock variables. The following lemma claims that we can assume that all values of clock variables in $\mathcal{A}$ are bounded above. The proof of the lemma is similar to the proof of Theorem 2 in [3].

Lemma 6. *Let $\mathcal{A}$ be a weighted timed automaton over $\mathbb{Z}$ with weight variables $\mathcal{W}$ and initial weight function ι, and let $b \in \mathbb{Z}^{\mathcal{W}}$ be an upper bound function satisfying $\iota(w) \leq b(w)$ for each $w \in \mathcal{W}$. Then there is a weighted timed automaton $\mathcal{A}'$ over $\mathbb{Z}$ such that for each reachable state (l, ν, μ) in $\mathcal{A}'$ we have $\nu(x) < u$ for some $u \in \mathbb{N}$ and each $x \in \mathcal{X}$, and there is an initialized b-feasible run in $\mathcal{A}$ if and only if there is an initialized b-feasible run in $\mathcal{A}'$.*

Let $u \in \mathbb{N}$ be the upper bound referred to in Lemma 6. In the remaining paper, we let $n \in \mathbb{N}$ be the smallest number $\alpha > 1$ such that $2^\alpha > \max(\{b(w) \mid w \in \mathcal{W}\} \cup \{u\}) + u$. In the following, we explain the addition widget and thereafter the other widgets needed. Finally, we explain how the widgets are combined to make up $\mathcal{A}_e$.

4.1 The Addition Widget

The addition widget is shown in Fig. 3. It is called by $\mathsf{ADD}\langle z, d, i, \mathcal{U}\rangle$, where z and d are clock variables, $i \in \mathbb{N}$ and $\mathcal{U}$ is a set of clock variables. We assume that the values of all clock variables are strictly less than 2^n. In the widget, the value of the clock variable z is increased by the product of the value of the clock variable d and 2^i. The entering values of the clock variables $x \in \mathcal{U}$ are maintained. This is done as follows.

First, we read out the value of the clock variable d. The idea is adopted from [7]. The value of d is at most $2^n - 1$. The widget consists of n phases.

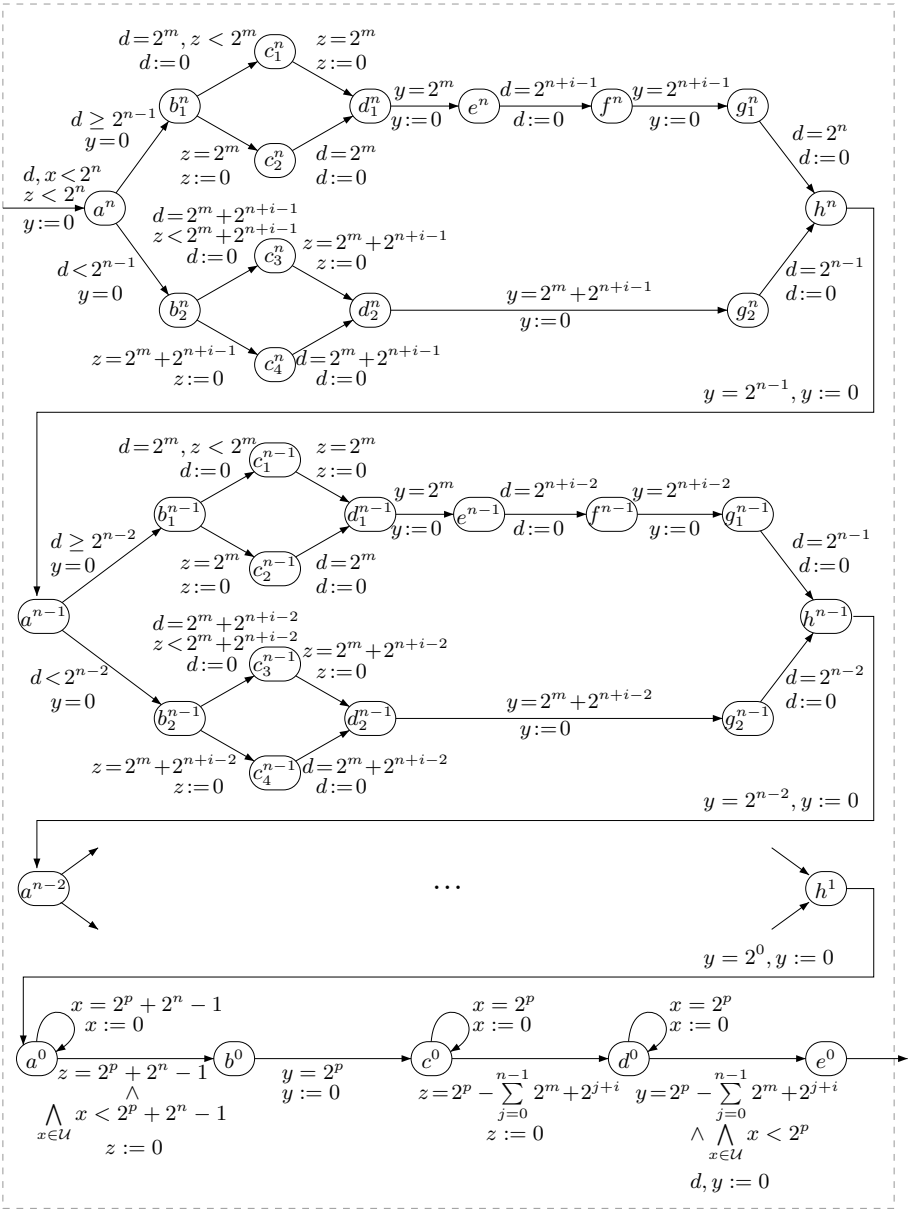


Fig. 3. The addition widget $\mathsf{ADD}\langle d, z, i, \mathcal{U}\rangle$. Edges labeled with, e.g., $x = 2^p$, indicate that for each $x \in \mathcal{U}$ there is an edge with clock constraint $x = 2^p$.

The first phase is called phase n, the second $n-1$, and so forth, until we reach the last phase called phase 1. For each $j \in \{n, ..., 1\}$, we check whether in phase j the j^{th} bit of the value of d is set or not by using the clock constraint $d \geq 2^{j-1}$ and $d < 2^{j-1}$, respectively.

If the j^{th} bit is set, we take one of the the upper paths in phase j. Here, we add 2^{j+i-1} to the value of z. This is done by letting exactly 2^{j+i-1} time units elapse between locations e^j and g_1^j. Doing so, the value of z is increased by 2^{j+i-1}. Note that the value of d is not increased by 2^{j+i-1} because it is reset to zero when its value equals 2^{j+i-1}. After that we set the j^{th} bit of d to zero in order to be able in the next phase $j-1$ to check whether the $(j-1)^{\text{th}}$ bit is set or not. This is done between locations g_1^j and a^{j-1}.

If the j^{th} bit is not set, we take one of the lower paths in phase j. In this case, we neither need to change the value of z nor the value of d. However, we want the time that is spent between locations a^j and a^{j-1} to be the same, independent on whether we take the upper or lower path, in order to be able to reconstruct the original values of $x \in \mathcal{U}$ after phase 1. In the upper paths, between e^j and a^{j-1}, exactly $2^{j+i-1} + 2^{j-1}$ time units elapse. This sum is added to the values of z and $x \in \mathcal{U}$. Hence, we want $2^{j+i-1} + 2^{j-1}$ to be added to the values of $x \in \mathcal{U}$ also in the lower paths. In contrast to the upper path, here we do not want to add 2^{j+i-1} to the value of z. Thus, we have to reset the value of z to zero if its value equals 2^{j+i-1} before we compare y against 2^{j+i-1}. However, the value of z may be greater than 2^{j+i-1}, and we thus need to use another upper bound for testing against the value of z. We let $m \in \mathbb{N}$ be the smallest number α such that $2^\alpha \geq 2^{i+n} + \sum_{j=1}^{n} 2^j$. We use the two upper bounds 2^n and 2^m in the clock constraints in the widget to let exactly the same time $2^m + 2^{j+i-1} + 2^{j-1}$ elapse between a^j and a^{j-1}, no matter whether the j^{th} bit is set or not. For this reason, in the following lemma the values of the clock variables contain some "unwanted" summands.

Lemma 7. *Let $i \in \mathbb{N}$, and let $\nu^n \in \mathbb{N}^{\mathcal{U} \cup \{d,y,z\}}$ be a clock valuation satisfying $\nu^n(d), \nu^n(z), \nu^n(x) < 2^n$ for each $x \in \mathcal{U}$ and $\nu^n(y) = 0$. Then there is a unique run from (a^n, ν^n) to (a^0, ν^0), and ν^0 satisfies*

- $\nu^0(x) = \nu^n(x) + (\sum_{j=0}^{n-1} 2^m + 2^{j+i}) + (2^n - 1)$ *for each* $x \in \mathcal{U}$,
- $\nu^0(y) = 0$,
- $\nu^0(z) = \nu^n(z) + \nu^n(d) \cdot 2^i + (2^n - 1)$.

We observe that in location a^0, the values of the clock variables z and $x \in \mathcal{U}$ share the same summand $2^n - 1$. We get rid of this summand between locations a^0 and c^0 in the product widget. In the following, we explain how this is done.

We first determine a new upper bound of the values of the clock variables when entering a^0. Let $p \in \mathbb{N}$ be the smallest number α such that $2^\alpha \geq 2^n + \sum_{j=0}^{n-1} 2^m + 2^{j+i}$. Then we have $\nu^0(c) < 2^p + (2^n - 1)$ for each $c \in \mathcal{U} \cup \{y, z\}$. We use the clock variable y to let exactly 2^p time units pass between entering a^0 and entering c^0. In between, the values of z and $x \in \mathcal{U}$ are reset to zero when their values equal $2^p + (2^n - 1)$. As a result, the values of z and $x \in \mathcal{U}$ are decreased by $2^n - 1$. In particular, the value of z equals $\nu^n(z) + \nu^n(d) \cdot 2^i$

when the automaton enters c^0. Note that $\nu^0(z) < \nu^0(x)$ for each $x \in \mathcal{U}$, and thus the clock constraint $\bigwedge_{x\in\mathcal{U}} x < 2^p + (2^n - 1)$ guarantees that each clock variable $x \in \mathcal{U}$ is reset before we take the edge to c^0. Using similar ideas we get rid of the common summand $\sum_{j=0}^{n-1} 2^m + 2^{j+i}$ in the values of $x \in \mathcal{U}$, while maintaining the value of z. Altogether, we obtain the following lemma.

Lemma 8. *If the addition widget is called by* $\mathsf{ADD}\langle z, d, i, \mathcal{U}\rangle$ *for some* $i \in \mathbb{N}$*, and the values of* d, z *and* $x \in \mathcal{U}$ *are given by the clock valuation* ν*, then we leave the subtraction widget if and only if* ν *satisfies* $\nu(d), \nu(z), \nu(x) < 2^n$ *for each* $x \in \mathcal{U}$*. When leaving the widget, the new values of* d, z *and* $x \in \mathcal{U}$ *are given by the clock valuation* ν' *satisfying* $\nu'(x) = \nu(x)$ *for each* $x \in \mathcal{U}$*,* $\nu'(z) = \nu(z) + \nu(d) \cdot 2^i$ *and* $\nu'(d) = 0$*.*

4.2 Other Widgets

For dealing with negative edge weights and weight rates, respectively, we use the subtraction widget, see Fig. 4. It is called by $\mathsf{SUB}\langle z, d, i, \mathcal{U}\rangle$, where z, d, i and $\mathcal{U}$ are as above. In the widget, the value of the clock variable z is decreased by the product of the value of the clock variable d and 2^i. The entering values of the clock variables $x \in \mathcal{U}$ are maintained. For copying the value of one

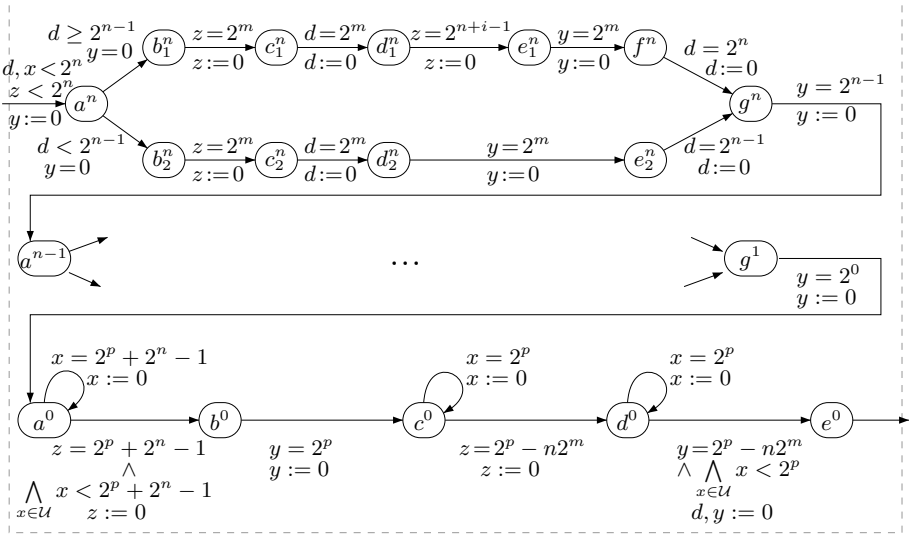


Fig. 4. The subtraction widget $\mathsf{SUB}\langle d, z, i, \mathcal{U}\rangle$. Edges labeled with, e.g., $x = 2^p$, indicate that for each $x \in \mathcal{U}$ there is an edge with clock constraint $x = 2^p$.

clock variable d to another clock variable z, we use instances of the copy widget, called by $\mathsf{COPY}\langle z, d, \mathcal{U}\rangle$. Moreover, we can assign some constant i to a clock variable z, while maintaining the values of some set $\mathcal{U}$ of clock variables, using

instances of the assignment widget, called by $\mathsf{ASGN}\langle z, i, \mathcal{U}\rangle$. Both the copy and the assignment widget are constructed using ideas explained in the preceding subsection.

4.3 Arrangement of the Widgets

For the edge $e = (l, \phi, \lambda, l')$ with $\mathsf{lwt}(l)(w_i) = k_i$ and $\mathsf{ewt}(e)(w_i) = k'_i$ for each $w_i \in \mathcal{W}$, the widget $\mathcal{A}_e$ is composed from instances of the addition, subtraction, copy and assignment widget as follows. Recall that the clock variables of $\mathcal{A}_e$ are $\mathcal{X}_e = \{x' \mid x \in \mathcal{X}\} \cup \{w' \mid w \in \mathcal{W}\} \cup \{d, d', y\}$.

The clock variables d and d' represent the time delay and its copy, respectively. The initial location is l, and d and d' are initialized to zero. From l there is an edge to some location l_1 labeled with the clock constraint ϕ and the reset set $\{y\}$. Note that when the automaton enters l_1 the values of all clock variables are increased by the time δ that is spent in l. In particular, the values of d and d' equal δ. We treat the weight variables consecutively and start with w_1. If $k_1 = 0$, no weight is supposed to be added to the value of w'_1 and thus the delay δ must be subtracted from the value of w'_1 again. For this, we call the subtraction widget $\mathsf{SUB}\langle w'_1, d, 0, \mathcal{X}_e \backslash \{w'_1, d\}\rangle$. If $k_1 \neq 0$, we let $\rho = k - 1$ and repeatedly add widgets to $\mathcal{A}_e$ until $\rho = 0$ as follows. Assume $|\rho| = 2^i$ for some $i \in \mathbb{N}\backslash\{0\}$. If $k_1 > 0$, we have to add the value of d 2^i times to the value of w'_1. For this, call the addition widget $\mathsf{ADD}\langle w'_1, d, i, \mathcal{X}_e \backslash \{w'_1, d\}\rangle$, and put $\rho = 0$. Analogously, if $k_1 < 0$, we have to subtract the value of d exactly 2^i times from the value of w'_1. In this case, call the subtraction widget $\mathsf{SUB}\langle w'_1, d, i, \mathcal{X}_e \backslash \{w'_1, d\}\rangle$, and put $\rho = 0$. If $|\rho| \neq 2^i$ for all $i \in \mathbb{N}$, let $j \in \mathbb{N}$ be the greatest number α such that $2^\alpha < |\rho|$. If $k_1 > 0$, call the addition widget $\mathsf{ADD}\langle w'_1, d, j, \mathcal{X}_e \backslash \{w'_1, d\}\rangle$, and decrement ρ by 2^j. Otherwise, call the subtraction widget $\mathsf{SUB}\langle w'_1, d, j, \mathcal{X}_e \backslash \{w'_1, d\}\rangle$, reset the value of d by calling the copy widget $\mathsf{COPY}\langle d, d', \mathcal{X}_e \backslash \{d, d'\}\rangle$, and increment ρ by 2^j. If eventually $\rho = 0$, the value of w'_1 equals the value of w_1 after the time delay in l. We check whether this value is within the interval-bound using an edge labelled with the clock constraint $0 \leq w'_1 \leq b(w_1)$. Then, we add the

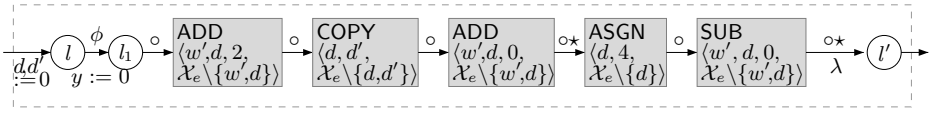


Fig. 5. Widget $\mathcal{A}_e$ for $e = (l, \phi, \lambda, l')$, $\mathsf{lwt}(l)(w) = 6$ and $\mathsf{ewt}(e)(w) = -4$ ($\star$ stands for the clock constraint $0 \leq w' \leq b(w)$ and $\circ$ stands for the clock constraint $y = 0$).

edgecost k'_1 (if unequal to 0) to the value of w'_1 using the addition or subtraction widget, respectively. For this, the absolute value k'_1 is assigned to a clock variable by calling the assignment widget $\mathsf{ASGN}\langle d, |k'_1|, \mathcal{X}_e \backslash \{d\}\rangle$. After the addition or subtraction widget, respectively, we call the copy widget again, to reset the value of d to δ. From there, we add an edge where we check whether the value of w'_1 is within the interval-bound using the clock constraint $0 \leq w'_1 \leq b(w_1)$.

We remark that the edges between all widget instances must be labelled with a clock constraint $y = 0$ in order to avoid that time elapses. We proceed in the same manner with the remaining weight variables. Finally, we add an edge to the final location l'. At this edge we reset all clock variables in λ according to the edge e.

Lemma 9. *Let $\nu \in \mathbb{N}^{\mathcal{X}}, \mu \in \mathbb{Z}^{\mathcal{W}}$ and $\nu_e \in \mathbb{N}^{\mathcal{X}_e}$ such that $\nu_e(x') = \nu(x)$ for each $x \in \mathcal{X}$, $\nu_e(w') = \mu(w)$ and $0 \leq \mu(w) \leq b(w)$ for each $w \in \mathcal{W}$, and $\delta \in \mathbb{N}$. Then $(l, \nu, \mu) \xrightarrow{\delta} (l, \nu'', \mu'') \xrightarrow{e} (l', \nu', \mu')$ is a finite b-feasible run in $\mathcal{A}$ if and only if there is a unique run from (l, ν_e) to (l', ν'_e) in $\mathcal{A}_e$, where $\nu'_e(x') = \nu'(x)$ for each $x \in \mathcal{X}$ and $\nu'_e(w') = \mu'(w)$ for each $w \in \mathcal{W}$.*

5 Open Problems

In this paper, we move towards a precise decidability border for the interval-bound problem for weighted timed automata. The decidability of the interval-bound problem for weighted timed automata over $\mathbb{R}$ with exactly one clock or one weight variable is still an open problem. Also, it is interesting to consider the lower-bound-problem and the lower-weak-upper-bound-problem mentioned in the introduction for weighted timed automata over time domain $\mathbb{R}$ with two clock or weight variables. Last but not least, all three resource scheduling problems are practically interesting also for weighted timed automata over different weight structures. One step into this direction was recently [4] done for weighted timed automata with exponentially growing value of the weight variable.

Acknowledgements

I would like to thank Uli Fahrenberg, Stefan Göller, Nicolas Markey and anonymous referees very much for their helpful discussions and comments.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. Theoretical Computer Science 318, 297–322 (2004)
3. Behrmann, G., Fehnker, A., Hune, T., Larsen, K., Pettersson, P., Romijn, J., Vaandrager, F.: Minimum-cost reachability for priced timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001)
4. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N.: Timed automata with observers under energy constraints. In: Johansson, K.H., Yi, W. (eds.) Proceedings of the 13th International Conference on Hybrid Systems: Computation and Control (HSCC 2010), pp. 61–70. ACM Press, Stockholm (2010)
5. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Srba, J.: Infinite runs in weighted timed automata with energy constraints. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 33–47. Springer, Heidelberg (2008)

6. Bouyer, P., Markey, N.: Costs are expensive! In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 53–68. Springer, Heidelberg (2007)
7. Courcoubetis, C., Yannakakis, M.: Minimum and maximum delay problems in real-time systems. Formal Methods in System Design 1, 385–415 (1992)
8. Minsky, M.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)

Finding Shuffle Words That Represent Optimal Scheduling of Shared Memory Access

Daniel Reidenbach and Markus L. Schmid*

Department of Computer Science, Loughborough University,
Loughborough, Leicestershire, LE11 3TU, UK
{D.Reidenbach,M.Schmid}@lboro.ac.uk

Abstract. In the present paper, we introduce and study the problem of computing, for any given finite set of words, a shuffle word with a minimum so-called scope coincidence degree. The scope coincidence degree is the maximum number of different symbols that parenthesise any position in the shuffle word. This problem is motivated by an application of a new automaton model and can be regarded as the problem of scheduling shared memory accesses of some parallel processes in a way that minimises the number of memory cells required. We investigate the complexity of this problem and show that it can be solved in polynomial time.

Keywords: String Algorithms, Shuffle, Memory Access Scheduling.

1 Introduction

A *shuffle word* of two words u and v is any word w that can be produced by inserting all symbols of u somewhere into v, in such a way that their relative order, given by u, is preserved. Thus, w comprises both u and v as a (scattered) subword, and each of its letters corresponds to exactly one letter of either u or v. Shuffle words of more than two words are constructed iteratively.

In the present paper, we wish to propose and study a question on shuffle words that is mainly motivated by the following problem on scheduling of memory accesses: Let us assume we have k processes and m values stored in memory cells, and all these processes need to access the stored values at some points during their execution. A process does not necessarily need all the m values at the same time, so a process might get along with less than m memory cells by, for example, first using a memory cell for a value x and then, as soon as x is not needed anymore, using the same cell for another, and previously unneeded, value y. As an example, we assume that process w_1 uses the values $\mathtt{a}, \mathtt{b}$ and $\mathtt{c}$ in the order $\mathtt{abacbc}$. This process only needs two memory cells: In the first cell, $\mathtt{b}$ is permanently stored, and the second cell first stores $\mathtt{a}$ until it is not required anymore and then stores value $\mathtt{c}$. This is possible, since the part of w_1 where $\mathtt{a}$ occurs and the part where $\mathtt{c}$ occurs can be completely separated

* Corresponding author.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 465–476, 2011.

from each other. If we now assume that the k processes cannot access the shared memory simultaneously, then the question arises how we can sequentially arrange all memory accesses such that a minimum overall number of memory cells is required. For example, if we assume that, in addition to process $w_1 = \mathtt{abacbc}$, there is another process $w_2 := \mathtt{abc}$, then we can of course first execute w_1 and afterwards w_2, which results in the memory access sequence $\mathtt{abacbcabc}$. It is easy to see that this requires a memory cell for each value $\mathtt{a}, \mathtt{b}$ and $\mathtt{c}$. On the other hand, we can first execute $\mathtt{aba}$ of process w_1, then process $w_2 = \mathtt{abc}$, and finally the remaining part $\mathtt{cbc}$ of w_1. This results in $\mathtt{abaabccbc}$, which allows us to use a single memory cell for both values $\mathtt{a}$ and $\mathtt{c}$ as before.

This scheduling problem can directly be formalised as a question on shuffle words. To this end, we merely have to interpret each of the k processes as a word over an alphabet of cardinality m, where m is the number of different values to be stored. Hence, our problem of finding the best way to organise the memory accesses of all processes directly translates into computing a shuffle word of the k processes that minimises the parameter determining the number of memory cells required. Unfortunately, even for $k = 2$, there is an exponential number of possible ways to schedule the memory accesses. However, we can present an algorithm solving this problem for arbitrary input words and a fixed alphabet size in polynomial time.

The above described problem is similar to the task of *register allocation* (see, e. g., [4,6]), which plays an important role in compiler optimisation. However, in register allocation, the problem is to allocate a number of m values accessed by a process to a fixed number of k registers, where $k < m$, with the possibility to temporarily move values from a register into the main memory. Since accessing the main memory is a much more expensive CPU operation, the optimisation objective is to find an allocation such that the number of memory accesses is minimised. The main differences to the problem investigated in this work are that the number of registers is fixed, the periods during which the values must be accessible in registers can be arbitrarily changed by storing them in the main memory, and there is usually not the problem of sequentialising several processes.

Our practical motivation and the definition of the above introduced problem result from an application of a new automaton model with two input heads [7]. In our application, these two input heads need to travel over factors of the input word; to this end, they need to know the lengths of these factors. Thus, each input head movement can be interpreted as a process that needs to access lengths of factors in a certain order. Within the scope of [7], the overall number of values that need to be stored simultaneously does not only affect the memory usage of the automaton; it also has a significant impact on the runtime of its computations. Thus, our problem on shuffle words is crucial in this context. Although we consider this nontrivial problem fundamental and believe that it might occur in other practical situations as well, it is not covered by any literature on scheduling (see, e. g., [1,3]) we are aware of, and the same holds for the research on the related *common supersequence problems* (see, e. g., [5]).

2 Basic Definitions

In the following, let Σ be a finite alphabet. A *word* (*over* Σ) is a finite sequence of symbols from Σ, and ε stands for the *empty word*. The symbol Σ^+ denotes the set of all nonempty words over Σ, and $\Sigma^* := \Sigma^+ \cup \{\varepsilon\}$. For the *concatenation* of two strings w_1, w_2 we write $w_1 \cdot w_2$ or simply $w_1 w_2$. We say that a string $v \in \Sigma^*$ is a *factor* of a string $w \in \Sigma^*$ if there are $u_1, u_2 \in \Sigma^*$ such that $w = u_1 \cdot v \cdot u_2$. If $u_1 = \varepsilon$ (or $u_2 = \varepsilon$), then v is a *prefix* of w (or a *suffix*, respectively). The notation $|K|$ stands for the size of a set K or the length of a string K. The term $\mathrm{alph}(w)$ denotes the set of all symbols occurring in w and, for each $\mathtt{a} \in \mathrm{alph}(w)$, $|w|_{\mathtt{a}}$ refers to the number of occurrences of $\mathtt{a}$ in w. If we wish to refer to the symbol at a certain position j, $1 \leq j \leq n$, in a word $w = \mathtt{a}_1 \cdot \mathtt{a}_2 \cdot \cdots \cdot \mathtt{a}_n$, $\mathtt{a}_i \in \Sigma$, $1 \leq i \leq n$, we use $w[j] := \mathtt{a}_j$. Furthermore, for each j, j', $1 \leq j < j' \leq |w|$, let $w[j, j'] := \mathtt{a}_j \cdot \mathtt{a}_{j+1} \cdot \cdots \cdot \mathtt{a}_{j'}$ and $w[j, -] := w[j, |w|]$. In case that $j > |w|$, we define $w[j, -] = \varepsilon$.

We now formally introduce the notion of a shuffle word. The *shuffle operation*, denoted by $\sqcup$, is a binary operation on words, defined inductively by

- $u \sqcup \varepsilon = \varepsilon \sqcup u = \{u\}$, for each $u \in \Sigma^*$,
- $\mathtt{a} \cdot u \sqcup \mathtt{b} \cdot v = \mathtt{a} \cdot (u \sqcup \mathtt{b} \cdot v) \cup \mathtt{b} \cdot (\mathtt{a} \cdot u \sqcup v)$, for all $u, v \in \Sigma^*$ and $\mathtt{a}, \mathtt{b} \in \Sigma$.

We extend the definition of the shuffle operation to the case of more than two words in the obvious way. Furthermore, for arbitrary words $w_1, w_2, \ldots, w_k \in \Sigma^*$, we call $\Gamma := w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ the *shuffle* of $w_1, \ldots, w_k$ and each word $w \in \Gamma$ is a *shuffle word* of $w_1, \ldots, w_k$. For example, $\mathtt{bcaabac} \in \mathtt{abc} \sqcup \mathtt{ba} \sqcup \mathtt{ca}$.

Finally, we introduce a special property of words that is important for our central problem. For an arbitrary $w \in \Sigma^*$ and any $\mathtt{b} \in \mathrm{alph}(w)$ let l, r, $1 \leq l, r \leq |w|$, be chosen such that $w[l] = w[r] = \mathtt{b}$ and there exists no k, $k < l$, with $w[k] = \mathtt{b}$ and no k', $r < k'$, with $w[k'] = \mathtt{b}$. Then the *scope of* $\mathtt{b}$ *in* w ($\mathrm{sc}_w(\mathtt{b})$ for short) is defined by $\mathrm{sc}_w(\mathtt{b}) := (l, r)$. Note that in the case that for some word w we have $w[j] = \mathtt{b}$ and $|w|_{\mathtt{b}} = 1$, the scope of $\mathtt{b}$ in w is (j, j). Now we are ready to define the so called *scope coincidence degree*: Let $w \in \Sigma^*$ be an arbitrary word and, for each i, $1 \leq i \leq |w|$, let

$$\mathrm{scd}_i(w) := |\{\mathtt{b} \in \Sigma \mid \mathtt{b} \neq w[i], \mathrm{sc}_w(\mathtt{b}) = (l, r) \text{ and } l < i < r\}| \ .$$

We call $\mathrm{scd}_i(w)$ the scope coincidence degree of position i in w. Furthermore, the scope coincidence degree of the word w is defined by

$$\mathrm{scd}(w) := \max\{\mathrm{scd}_i(w) \mid 1 \leq i \leq |w|\} \ .$$

As an example, we now consider the word $w := \mathtt{acacbbdeabcedefdeff}$. It can easily be verified that $\mathrm{scd}_8(w) = \mathrm{scd}_9(w) = 4$ and $\mathrm{scd}_i(w) < 4$ if $i \notin \{8, 9\}$. Hence, $\mathrm{scd}(w) = 4$.

3 The Problem of Computing Shuffle Words with Minimum Scope Coincidence Degree

In our practical motivation given in the introduction, we state that we wish to sequentially arrange parallel sequences of memory accesses. These sequences shall be modelled by words and the procedure of sequentially arranging them is described by the shuffle operation. Furthermore, our goal is to construct a shuffle word such that, for any memory access in the shuffle word, the maximum number of values that already have been accessed and shall again be accessed later on is minimal. For instance, in the shuffle word `abaabccbc` of `abacbc` and `abc`, for each position i, $1 \leq i \leq 9$, there exists at most one other symbol that has an occurrence to either side of position i. On the other hand, with respect to the shuffle word `abacbcabc` we observe that at position 4 symbol `c` occurs while both symbols `a` and `b` have an occurrence to either side of position 4. This number of symbols occurring to both sides of an occurrence of another symbol is precisely the scope coincidence degree. Hence, our central problem is the problem of finding, for any given set of words, a shuffle word with a minimum scope coincidence degree.

Problem 1. For an arbitrary alphabet Σ, let the problem SWminSCD$_\Sigma$ be the problem of finding, for given $w_i \in \Sigma^*$, $1 \leq i \leq k$, a shuffle word $w \in w_1 \sqcup \ldots \sqcup w_k$ with minimum scope coincidence degree.

Note that in the definition of SWminSCD$_\Sigma$, the alphabet Σ is constant and not part of the input; hence, for each alphabet Σ, inputs for the problem SWminSCD$_\Sigma$ have to consist of words over the alphabet Σ exclusively. This shall be important for complexity considerations.

A naive approach to solving SWminSCD$_\Sigma$ on input $(w_1, w_2, \ldots, w_k)$ would be to enumerate all elements in $w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ in order to find one with minimum scope coincidence degree. However, the size of this search space is too large, as the cardinality of the shuffle $w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ is, in the worst case, given by the multinomial coefficient [2]. More precisely,

$$|w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k| \leq \binom{n}{|w_1|, |w_2|, \ldots, |w_k|} = \frac{n!}{|w_1|! \times |w_2|! \times \ldots \times |w_k|!},$$

where $n := \sum_{i=1}^{k} |w_i|$, and $x!$ denotes the factorial of an integer x. This demonstrates that the search space of a naive algorithm can be exponentially large. Therefore, a polynomial time algorithm cannot simply search the whole shuffle $u_1 \sqcup u_2 \sqcup \ldots \sqcup u_k$, which implies that a more sophisticated strategy is required.

Before we present a successful approach to SWminSCD$_\Sigma$ in the next section, we discuss some simple observations. First, we note that solving SWminSCD$_\Sigma$ on input $w_1, w_2, \ldots, w_k$ by first computing a minimal shuffle word w of w_1 and w_2 (ignoring $w_3, \ldots, w_n$) and then solving SWminSCD$_\Sigma$ on the smaller input $w, w_3 \ldots, w_n$ and so on is not possible. This can be easily comprehended by considering the words $w_1 :=$ `aa` and $w_2 :=$ `bb` and observe that $w :=$ `aabb` is a shuffle word of w_1 and w_2 that is optimal, since $\mathrm{scd}(w) = 0$. Now, it is not

possible to shuffle w with $w_3 := \texttt{ba}$ in such a way that the resulting shuffle word has a scope coincidence degree of 0; however, $w' := \texttt{bbbaaa} \in w_1 \sqcup w_2 \sqcup w_3$ and $\mathrm{scd}(w') = 0$.

Intuitively, it seems obvious that the scope coincidence degree only depends on the leftmost and rightmost occurrences of the symbols. In other words, removing a symbol from a word that does not constitute a leftmost or rightmost occurrence should not change the scope coincidence degree of that word. For instance, if we consider a word $w := \alpha \cdot c \cdot \beta$, where c is a symbol occurring in α and β, then all symbols in the word w that are in the scope of c are still in the scope of c with respect to the word $\alpha \cdot \beta$.

Consequently, we can first remove all occurrences of symbols that are neither leftmost nor rightmost occurrences, then solve $\mathrm{SWminSCD}_\Sigma$ on these reduced words and finally insert the removed occurrences into the shuffle word in such a way that the scope coincidence degree does not increase. A formalisation and proof of correctness of this approach is omitted. This reduction of the input words results in a smaller, but still exponentially large search space. Hence, this approach does not seem to help us solving $\mathrm{SWminSCD}_\Sigma$ in polynomial time.

In the following section, we shall establish basic results about the scope coincidence degree of words. These results shall then be applied later on in order to analyse the scope coincidence degree of shuffle words.

4 Further Properties of the Scope Coincidence Degree

In this section, we take a closer look at the scope coincidence degree. We are particularly interested in how words can be transformed without increasing their scope coincidence degree. First, we consider a proposition which describes a very basic property of the scope coincidence degree that directly follows from its definition. It can roughly be stated by saying that the scope coincidence degree of a certain position i does not depend on the order of the symbols occurring to the left and to the right of i.

Proposition 1. *Let $u, v \in \Sigma^*$ with $|u| = |v|$. If, for some i, $1 \leq i \leq |u|$, $u[i] = v[i]$ and $u[1, i-1]$ is a permutation of $v[1, i-1]$ and $u[i+1, -]$ is a permutation of $v[i+1, -]$, then $\mathrm{scd}_i(u) = \mathrm{scd}_i(v)$.*

Hence, for every position in a word we can permute the part to the left or to the right of this position without changing its scope coincidence degree. The scope coincidence degree of the positions in the parts that are permuted is not necessarily stable, and thus the scope coincidence degree of the whole word may change. However, if a factor of a word w satisfies a certain property, i. e., it contains no leftmost occurrence of a symbol with respect to w (it may, however, contain rightmost occurrences of symbols), then we can arbitrarily permute this factor without changing the scope coincidence degree of the whole word:

Lemma 1. *Let α, β, π, $\pi' \in \Sigma^*$, where π is a permutation of π' and $\mathrm{alph}(\pi) \subseteq \mathrm{alph}(\alpha)$. Then $\mathrm{scd}(\alpha \cdot \pi \cdot \beta) = \mathrm{scd}(\alpha \cdot \pi' \cdot \beta)$.*

The next two lemmas show that if certain conditions hold, then we can move one or several symbols in a word to the left without increasing the scope coincidence degree. The first result of that kind is related to the situation where only one symbol is moved, and the second lemma describes the case where several symbols are moved and therefore makes use of the first lemma.

We can informally summarise the first lemma in the following way. We assume that at position i in a word w a certain symbol $\mathtt{b}$ occurs and, furthermore, this is not the leftmost occurrence of $\mathtt{b}$. Then we can move this symbol to the left without increasing the scope coincidence degree of w as long as it is not moved to the left of the leftmost occurrence of a $\mathtt{b}$ in w. This seems plausible, as such an operation shortens the scope of symbol $\mathtt{b}$ or leaves it unchanged. However, we might move this certain $\mathtt{b}$ into a region of the word where many scopes coincide; thus, it is possible that the scope coincidence degree of the new position of $\mathtt{b}$ increases compared to its old position. We can show that this increase of the scope coincidence degree of that certain position does not affect the scope coincidence degree of the whole word:

Lemma 2. *For all* $\alpha, \beta, \gamma \in \Sigma^*$ *and for each* $\mathtt{b} \in \Sigma$ *with* $\mathtt{b} \in \mathrm{alph}(\alpha)$,

$$\mathrm{scd}(\alpha \cdot \mathtt{b} \cdot \beta \cdot \gamma) \leq \mathrm{scd}(\alpha \cdot \beta \cdot \mathtt{b} \cdot \gamma)\,.$$

Obviously, if for some word w the condition of Lemma 2 is satisfied not only for one symbol $\mathtt{b}$ but for several symbols $\mathtt{d}_1, \mathtt{d}_2, \ldots, \mathtt{d}_n$, then we can separately move each of these $\mathtt{d}_i$, $1 \leq i \leq n$, to the left and conclude that the scope coincidence degree of the resulting word does not increase compared to w. This observation is described by the following lemma.

Lemma 3. *Let* α, γ, $\beta_i \in \Sigma^*$, $0 \leq i \leq n$, $n \in \mathbb{N}$, *and let* $\mathtt{d}_i \in \Sigma$, $1 \leq i \leq n$, *such that* $\mathtt{d}_i \in \mathrm{alph}(\alpha)$, $1 \leq i \leq n$. *Then*

$$\mathrm{scd}(\alpha \cdot \mathtt{d}_1 \cdot \mathtt{d}_2 \cdot \cdots \cdot \mathtt{d}_n \cdot \beta_1 \cdot \beta_2 \cdot \cdots \cdot \beta_n \cdot \gamma) \leq \mathrm{scd}(\alpha \cdot \beta_1 \cdot \mathtt{d}_1 \cdot \beta_2 \cdot \mathtt{d}_2 \cdot \cdots \cdot \beta_n \cdot \mathtt{d}_n \cdot \gamma)\,.$$

Concerning the previous lemma, we observe that we can as well position the symbols $\mathtt{d}_i$, $1 \leq i \leq n$, in any other order than $d_1 \cdot d_2 \cdot \cdots \cdot d_n$ and would still obtain a word with a scope coincidence degree that has not increased. Furthermore, with Lemma 1, we can conclude that the scope coincidence degree is exactly the same, no matter in which order the symbols $\mathtt{d}_i$, $1 \leq i \leq n$, occur between α and β_1.

5 Solving the Problem SWminSCD$_\Sigma$

In this section, we present an efficient way to solve SWminSCD$_\Sigma$. Our approach is established by identifying a certain set of well-formed shuffle words which contains at least one shuffle word with minimum scope coincidence degree and, moreover, is considerably smaller than the set of all shuffle words. To this end, we shall first introduce a general concept for constructing shuffle words, and then a simpler and standardised way of constructing shuffle words is defined. By applying the lemmas given in the previous section, we are able to show that there exists a shuffle word with minimum scope coincidence degree that can be constructed in this simple way.

Let $w_1, w_2, \ldots, w_k \in \Sigma^*$ be arbitrary words. We consider these words as stack-like data structures where the leftmost symbol is the topmost stack element. Now we can empty these stacks by successively applying the pop operation and every time we pop a symbol from a stack, we append this symbol to the end of an initially empty word w. Thus, as soon as all stacks are empty, we obtain a word built up of symbols from the stacks, and this word is certainly a shuffle word of $w_1, w_2, \ldots, w_k$.

It seems to be useful to reason about different ways of constructing a shuffle word rather than about actual shuffle words, as this allows us to ignore the fact that in general a shuffle word can be constructed in several completely different ways. In particular the following unpleasant situation seems to complicate the analysis of shuffle words. If we consider a shuffle word w of the words $w_1, w_2, \ldots, w_k$, it might be desirable to know, for a symbol $\mathtt{b}$ on a certain position j, which w_i, $1 \leq i \leq k$, is the origin of that symbol. Obviously, this depends on how the shuffle word has been constructed from the words w_i, $1 \leq i \leq k$, and for different ways of constructing w, the symbol $\mathtt{b}$ on position j may originate from different words w_i, $1 \leq i \leq k$. In particular, if we want to alter shuffle words by moving certain symbols, it is essential to know the origin words w_i, $1 \leq i \leq k$, of the symbols, as this determines how they can be moved without destroying the shuffle properties.

We now formalise the way to construct a shuffle word by utilising the stack analogy introduced above. An arbitrary configuration (of the content) of the stacks corresponding to words w_i, $1 \leq i \leq k$, can be given as a tuple $(v_1, \ldots, v_k)$ of suffixes, i. e. $w_i = u_i \cdot v_i$, $1 \leq i \leq k$. Such a configuration $(v_1, \ldots, v_k)$ is then changed into another configuration $(v_1, \ldots, v_{i-1}, v_i', v_{i+1}, \ldots, v_k)$, by a pop operation, where $v_i = \mathtt{b} \cdot v_i'$ for some i, $1 \leq i \leq k$, and for some $\mathtt{b} \in \Sigma$. Initially, we start with the stack content configuration $(w_1, \ldots, w_k)$ and as soon as all the stacks are empty, which can be represented by $(\varepsilon, \ldots, \varepsilon)$, our shuffle word is complete. Hence, we can represent a way to construct a shuffle word by a sequence of these tuples of stack contents:

Definition 1. *A* construction sequence *for words $w_1, w_2, \ldots, w_k$, $w_i \in \Sigma^*$, $1 \leq i \leq k$, is a sequence $s := (s_0, s_1, \ldots, s_m)$, $m := |w_1 \cdot \cdots \cdot w_k|$ such that*

- *$s_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, $0 \leq i \leq m$, where, for each i, $0 \leq i \leq m$, and for each j, $1 \leq j \leq k$, $v_{i,j}$ is a suffix of w_j ,*
- *$s_0 = (w_1, \ldots, w_k)$ and $s_m = (\varepsilon, \varepsilon, \ldots, \varepsilon)$,*
- *for each i, $0 \leq i \leq m-1$, there exists a j_i, $1 \leq j_i \leq k$, and a $\mathtt{b}_i \in \Sigma$ such that $v_{i,j_i} = \mathtt{b}_i \cdot v_{i+1,j_i}$ and $v_{i,j'} = v_{i+1,j'}$, $j' \neq j_i$.*

The shuffle word $w = \mathtt{b}_0 \cdot \mathtt{b}_1 \cdot \cdots \cdot \mathtt{b}_{m-1}$ is said to correspond *to s. In a step from s_i to s_{i+1}, $0 \leq i \leq m-1$, of s, we say that the symbol $\mathtt{b}_{i+1}$ is* consumed.

To illustrate the definition of construction sequences, we consider an example construction sequence $s := (s_0, s_1, \ldots, s_9)$ corresponding to a shuffle word of the words $w_1 := \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}$ and $w_2 := \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c}$:

$$s := ((\mathtt{a}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}), (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}), (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{b}\cdot\mathtt{c}),$$
$$(\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}), (\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}), (\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \varepsilon), (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \varepsilon),$$
$$(\mathtt{b}\cdot\mathtt{c},\, \varepsilon), (\mathtt{c},\, \varepsilon), (\varepsilon,\, \varepsilon))\,.$$

The shuffle word corresponding to s is $w := \mathtt{a}\cdot\mathtt{a}\cdot\mathtt{b}\cdot\mathtt{b}\cdot\mathtt{c}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c}$, and it is easy to see that $\mathrm{scd}(w) = 2$.

In the next definition, we introduce a certain property of construction sequences that can be easily described in an informal way. Recall that in an arbitrary step from s_i to s_{i+1} of a construction sequence s, exactly one symbol $\mathtt{b}$ is consumed. Hence, at each position $s_i = (v_1, \ldots, v_k)$ of a construction sequence, we have a part u of already consumed symbols, which is actually a prefix of the shuffle word we are about to construct and some suffixes $v_1, \ldots, v_k$ that remain to be consumed. A symbol $\mathtt{b}$ that is consumed can be an *old* symbol that already occurs in the part u or it can be a *new* symbol that is consumed for the first time. Now the special property to be introduced next is that this consumption of symbols is greedy with respect to old symbols: Whenever a new symbol $\mathtt{b}$ is consumed in a step from s_i to $s_{i+1} = (v_1, \ldots, v_k)$, we require the construction sequence to first consume as many old symbols as possible from the remaining $v_1, \ldots, v_k$ before another new symbol is consumed. For the sake of uniqueness, this greedy consumption of old symbols shall be defined in a canonical order, i. e. we first consume all the old symbols from v_1, then all the old symbols from v_2 and so on. Obviously, there are still several possible greedy construction sequences for some input words w_i, $1 \leq i \leq k$, as whenever a new symbol is consumed, we have a choice of k possible suffixes to consume this symbol from. We formally define this greedy property of construction sequences.

Definition 2. *Let $w \in w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$, $w_i \in \Sigma^*$, $1 \leq i \leq k$, and let $s := (s_0, s_1, \ldots, s_{|w|})$ with $s_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, $0 \leq i \leq |w|$, be an arbitrary construction sequence for w. An element s_i, $1 \leq i \leq |w| - 1$, of s satisfies the* greedy property *if and only if $w[i] \notin \mathrm{alph}(w[1, i-1])$ implies that for each j, $1 \leq j \leq k$, $s_{i+|u_1 \cdot \ldots \cdot u_j|} = (\overline{v}_{i,1}, \ldots, \overline{v}_{i,j}, v_{i,j+1}, \ldots, v_{i,k})$, where $v_{i,j} = u_j \cdot \overline{v}_{i,j}$ and u_j is the longest prefix of $v_{i,j}$ such that $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(w[1, i])$.*

A construction sequence $s := (s_0, s_1, \ldots, s_{|w|})$ for some $w \in \Sigma^*$ is a greedy construction sequence *if and only if, for each i, $1 \leq i \leq |w| - 1$, s_i satisfies the greedy property. A shuffle word w that corresponds to a greedy construction sequence is a* greedy shuffle word.

As an example, we again consider the words $w_1 = \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c}$ and $w_2 = \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}$. This time, we present a greedy construction sequence $s := (s_0, s_1, \ldots, s_9)$ for w_1 and w_2:

$$s := ((\mathtt{a}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}), (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}),$$
$$(\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{b}\cdot\mathtt{c}), (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}), (\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}),$$
$$(\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \mathtt{c}), (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\, \varepsilon), (\mathtt{b}\cdot\mathtt{c},\, \varepsilon), (\mathtt{c},\, \varepsilon), (\varepsilon,\, \varepsilon))\,.$$

Obviously, the shuffle word $w := \mathtt{a}\cdot\mathtt{a}\cdot\mathtt{b}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c}$ corresponds to the construction sequence s and $\mathrm{scd}(w) = 1$. To show that s is a greedy construction

sequence, it is sufficient to observe that s_1, s_3 and s_6 (the elements where a new symbol is consumed) satisfy the greedy property. We only show that s_3 satisfies the greedy property as s_1 and s_6 can be handled analogously. First, we recall that $s_3 = (\mathtt{b} \cdot \mathtt{a} \cdot \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}, \mathtt{c})$ and note that, in terms of Definition 2, we have $u_1 := \mathtt{b} \cdot \mathtt{a}$, $\overline{v}_{3,1} := \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}$, $u_2 := \varepsilon$ and $\overline{v}_{3,2} := \mathtt{c}$. By definition, s_3 only satisfies the greedy property if $s_{3+|u_1|} = (\overline{v}_{3,1}, v_{3,2})$ and $s_{3+|u_1 \cdot u_2|} = (\overline{v}_{3,1}, \overline{v}_{3,2})$. Since $|u_1| = |u_1 \cdot u_2| = 2$, $\overline{v}_{3,1} = \mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}$, $v_{3,2} = \overline{v}_{3,2} = \mathtt{c}$ and $s_5 = (\mathtt{c} \cdot \mathtt{b} \cdot \mathtt{c}, \mathtt{c})$, this clearly holds.

In the following, we show how we can transform an arbitrary construction sequence $s := (s_0, s_1, \ldots, s_m)$ into a greedy one. Informally speaking, this is done by determining the first element s_i that does not satisfy the greedy property and then we simply redefine all the elements s_j, $i+1 \leq j \leq m$, in a way such that s_i satisfies the greedy property. If we apply this method iteratively, we can obtain a greedy construction sequence. Next, we introduce the formal definition of that transformation and explain it in more detail later on.

Definition 3. *We define an algorithm G that transforms a construction sequence. Let $s := (s_0, s_1, \ldots, s_m)$ with $s_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,k})$, $0 \leq i \leq m$, be an arbitrary construction sequence that corresponds to a shuffle word w. In the case that s is a greedy construction sequence, we define $G(s) := s$. If s is not a greedy construction sequence, then let p, $1 \leq p \leq m$, be the smallest number such that s_p does not satisfy the greedy property. Furthermore, for each j, $1 \leq j \leq k$, let u_j be the longest prefix of $v_{p,j}$ with $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(w[1,p])$ and let $v_{p,j} = u_j \cdot \overline{v}_{p,j}$. For each j, $1 \leq j \leq k$, let $\sigma_j : \Sigma^* \to \Sigma^*$ be a mapping defined by $\sigma_j(x) := \overline{v}_{p,j}$ if $|x| > |\overline{v}_{p,j}|$ and $\sigma_j(x) := x$ otherwise, for each $x \in \Sigma^*$. Furthermore, let the mapping $\sigma : (\Sigma^*)^k \to (\Sigma^*)^k$ be defined by $\sigma((v_1, \ldots, v_k)) := (\sigma_1(v_1), \ldots, \sigma_k(v_k))$, $v_j \in \Sigma^*$, $1 \leq j \leq k$. Finally, we define $G(s) := (s'_0, s'_1, \ldots, s'_{m'})$, where the elements s'_i, $0 \leq i \leq m'$, are defined by the following procedure.*

1: $s'_i := s_i$, $0 \leq i \leq p$
2: ***for all*** j, $1 \leq j \leq k$, ***do***
3: $\quad s'_{p+|u_1 \cdot \cdots \cdot u_j|} := (\overline{v}_{p,1}, \ldots, \overline{v}_{p,j}, v_{p,j+1}, \ldots, v_{p,k})$
4: $\quad$ ***for all*** l_j, $2 \leq l_j \leq |u_j|$, ***do***
5: $\quad\quad s_{p+|u_1 \cdot \cdots \cdot u_{j-1}|+l_j-1} := (\overline{v}_{p,1}, \ldots, \overline{v}_{p,j-1}, u_j[l_j, -] \cdot \overline{v}_{p,j}, v_{p,j+1}, \ldots, v_{p,k})$
6: $\quad$ ***end for***
7: ***end for***
8: $q' \leftarrow p+1$
9: $q'' \leftarrow p + |u_1 \cdot \cdots \cdot u_k| + 1$
10: ***while*** $q' \leq m$ ***do***
11: $\quad$ ***if*** $\sigma(s_{q'-1}) \neq \sigma(s_{q'})$ ***then***
12: $\quad\quad s'_{q''} := \sigma(s_{q'})$
13: $\quad\quad q'' \leftarrow q''+1$
14: $\quad$ ***end if***
15: $\quad q' \leftarrow q'+1$
16: ***end while***

As mentioned above, we explain the previous definition in an informal way and shall later consider an example. Let $s := (s_0, s_1, \ldots, s_m)$ be an arbitrary construction sequence and let p and the u_j, $1 \leq j \leq k$, be defined as in Definition 3. The sequence $s' := (s'_0, s'_1, \ldots, s'_{m'}) := G(s)$ is obtained from s in the following way. We keep the first p elements and then redefine the next $|u_1 \cdot \cdots \cdot u_k|$ elements in such a way that s'_p satisfies the greedy property as described by Definition 2. This is done in lines 1 to 9 of the algorithm. Then, in order to build the rest of s', we modify the elements s_i, $p+1 \leq i \leq m$. First, for each component $v_{i,j}$, $p+1 \leq i \leq m$, $1 \leq j \leq k$, if $|\overline{v}_{p,j}| < |v_{i,j}|$ we know that $v_{i,j} = \overline{u}_j \cdot \overline{v}_{p,j}$, where $\overline{u}_j$ is a suffix of u_j. In s', this part $\overline{u}_j$ has already been consumed by the new elements s'_i, $p+1 \leq i \leq p+|u_1 \cdot \cdots \cdot u_k|$, and is, thus, simply cut off and discarded by the mapping σ in Definition 3. More precisely, if a component $v_{i,j}$, $p+1 \leq i \leq m$, $1 \leq j \leq k$, of an element s_i is longer than $\overline{v}_{p,j}$, then $\sigma_j(v_{i,j}) = \overline{v}_{i,j}$. If on the other hand $|v_{i,j}| \leq |\overline{v}_{p,j}|$, then $\sigma(v_{i,j}) = v_{i,j}$. This is done in lines 10 to 18 of the algorithm.

The following proposition shows that $G(s)$ actually satisfies the conditions to be a proper construction sequence:

Proposition 2. *For each construction sequence s of some words $w_1, \ldots, w_k$, $G(s)$ is also a construction sequence of the words $w_1, \ldots, w_k$.*

Now, as an example for Definition 3, we consider the construction sequence

$$\begin{aligned} s := ((&\mathtt{a}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}), (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}),\\ &(\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{b}\cdot\mathtt{c}), (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{c}), (\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{c}),\\ &(\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \varepsilon), (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \varepsilon), (\mathtt{b}\cdot\mathtt{c},\ \varepsilon), (\mathtt{c},\ \varepsilon), (\varepsilon,\ \varepsilon)) \end{aligned}$$

of the words $w_1 = \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c}$ and $w_2 = \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}$, as given below Definition 1. The shuffle word that corresponds to this construction sequence is $w := \mathtt{a}\cdot\mathtt{a}\cdot\mathtt{b}\cdot\mathtt{b}\cdot\mathtt{c}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c}$. We now illustrate how the construction sequence $s' := (s'_0, s'_1, \ldots, s'_m) := G(s)$ is constructed by the algorithm G. First, we note that $s_3 = (\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{c})$ is the first element that does not satisfy the greedy property, since in the step from s_4 to s_5, the symbol $\mathtt{c}$ is consumed before the leftmost (and old) symbol $\mathtt{a}$ from $v_{4,1}$ is consumed. Thus, $s'_i = s_i$, $1 \leq i \leq 3$. As $w[1,3] = \mathtt{a}\cdot\mathtt{a}\cdot\mathtt{b}$, we conclude that $u_1 := \mathtt{b}\cdot\mathtt{a}$ and $u_2 := \varepsilon$. So the next to elements s'_4 and s'_5 consume the factor u_1 from $\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c}$, hence, $s'_4 = (\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{c})$ and $s'_5 = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{c})$. Now let σ be defined as in Definition 3, thus,

$$\begin{aligned} &\sigma(s_3) = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{c}), \sigma(s_4) = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{c}), \sigma(s_5) = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \varepsilon),\\ &\sigma(s_6) = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \varepsilon), \sigma(s_7) = (\mathtt{b}\cdot\mathtt{c},\ \varepsilon), \sigma(s_8) = (\mathtt{c},\ \varepsilon), \sigma(s_9) = (\varepsilon,\ \varepsilon)\,. \end{aligned}$$

Since $\sigma(s_3) = \sigma(s_4)$ and $\sigma(s_5) = \sigma(s_6)$, we ignore $\sigma(s_4)$ and $\sigma(s_6)$; hence,

$$\begin{aligned} &s'_6 = \sigma(s_5) = (\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \varepsilon), s'_7 = \sigma(s_7) = (\mathtt{b}\cdot\mathtt{c},\ \varepsilon),\\ &s'_8 = \sigma(s_8) = (\mathtt{c},\ \varepsilon), s'_9 = \sigma(s_9) = (\varepsilon,\ \varepsilon)\,. \end{aligned}$$

In conclusion

$$\begin{aligned} s' = ((&\mathtt{a}\cdot\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}),(\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{a}\cdot\mathtt{b}\cdot\mathtt{c}),\\ &(\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{b}\cdot\mathtt{c}),(\mathtt{b}\cdot\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{c}),(\mathtt{a}\cdot\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{c}),\\ &(\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \mathtt{c}),(\mathtt{c}\cdot\mathtt{b}\cdot\mathtt{c},\ \varepsilon),(\mathtt{b}\cdot\mathtt{c},\ \varepsilon),(\mathtt{c},\ \varepsilon),(\varepsilon,\ \varepsilon))\,. \end{aligned}$$

Next, we show that if in a construction sequence $s := (s_0, s_1, \ldots, s_m)$ the element s_p is the first element that does not satisfy the greedy property, then in $G(s) := (s'_0, s'_1, \ldots, s'_m)$ the element s'_p satisfies the greedy property. This follows from Definition 3 and has already been informally explained.

Proposition 3. *Let $s := (s_0, s_1, \ldots, s_m)$ be any construction sequence that is not greedy, and let p, $0 \leq p \leq m$, be the smallest number such that s_p does not satisfy the greedy property. Let $s' := (s'_0, s'_1, \ldots, s'_m) := G(s)$ and, if s' is not greedy, let q, $0 \leq q \leq m$, be the smallest number such that s'_q does not satisfy the greedy property. Then $p < q$.*

More importantly, we can also state that the scope coincidence degree of the shuffle word corresponding to $G(s)$ does not increase compared to the shuffle word that corresponds to s. To this end, we shall employ the lemmas introduced in Section 4.

Lemma 4. *Let s be an arbitrary construction sequence that corresponds to the shuffle word w and let w' be the shuffle word corresponding to $G(s)$. Then* $\mathrm{scd}(w') \leq \mathrm{scd}(w)$.

The previous lemma is very important, as it implies our next result, which can be stated as follows. By iteratively applying the algorithm G, we can transform each construction sequence, including the ones corresponding to shuffle words with minimum scope coincidence degree, into a greedy construction sequence that corresponds to a shuffle word with a scope coincidence degree that is the same or even lower:

Theorem 1. *Let $w \in w_1 \sqcup \ldots \sqcup w_k$, $w_i \in \Sigma^*$, $1 \leq i \leq k$, be an arbitrary shuffle word. There exists a greedy shuffle word w' such that* $\mathrm{scd}(w') \leq \mathrm{scd}(w)$.

This particularly implies that there exists a greedy shuffle word with minimum scope coincidence degree. Hence, $\mathrm{SWminSCD}_\Sigma$ reduces to the problem of finding a greedy shuffle word with minimum scope coincidence degree.

The following algorithm – referred to as SolveSWminSCD – applies the above established way to construct greedy shuffle words and enumerates all possible greedy shuffle words in order to solve $\mathrm{SWminSCD}_\Sigma$.

Next, we state that this algorithm works correctly and establish its time complexity.

Theorem 2. *On an arbitrary input $(w_1, w_2, \ldots, w_k) \in (\Sigma^*)^k$, the algorithm* SolveSWminSCD *computes its output $w \in w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ in time* $\mathrm{O}(|w_1 \cdot \cdots \cdot w_k| \times k^{|\Sigma|})$ *and there exists no $w' \in w_1 \sqcup w_2 \sqcup \ldots \sqcup w_k$ with* $\mathrm{scd}(w') < \mathrm{scd}(w)$.

Algorithm 1. SolveSWminSCD

optShuffle := ε, minscd := $|\Sigma|$, push $(\varepsilon, (w_1, \ldots, w_k))$
while the stack is not empty **do**
  Pop element $(w, (v_1, \ldots, v_k))$
  if $|v_1 \cdot v_2 \cdot \cdots \cdot v_k| = 0$ and $\mathrm{scd}(w) <$ minscd **then**
    optShuffle := w
    minscd := $\mathrm{scd}(w)$
  else
    for all i, $1 \leq i \leq k$, with $v_i \neq \varepsilon$ **do**
      $\mathtt{b} := v_i[1]$
      $v_i := v_i[2, -]$
      Let u_j, $1 \leq j \leq k$, be the longest prefix of v_j with $\mathrm{alph}(u_j) \subseteq \mathrm{alph}(w \cdot \mathtt{b})$
      Push $(w \cdot \mathtt{b} \cdot u_1 \cdot u_2 \cdot \cdots \cdot u_k, (v_1[|u_1|+1, -], v_2[|u_2|+1, -], \ldots, v_k[|u_k|+1, -]))$
    end for
  end if
end while
Output optShuffle

By applying the observation from Section 3 – i. e., we can solve SWminSCD by first deleting all the occurrences of symbols in the input words that are neither leftmost nor rightmost occurrences and then solving SWminSCD for the reduced input words – we can prove the following result about the time complexity of SWminSCD:

Theorem 3. *The problem* SWminSCD *on an arbitrary input* $(w_1, w_2, \ldots, w_k) \in (\Sigma^*)^k$ *can be solved in time* $\mathrm{O}(|\Sigma| \times k^{|\Sigma|+1})$.

References

1. Conway, R.W., Maxwell, W.L., Miller, L.W.: Theory of Scheduling. Addison-Wesley Publishing Company, Reading (1967)
2. Flajolet, P., Gardy, D., Thimonier, L.: Birthday paradox, coupon collectors, caching algorithms and self-organizing search. Discrete Applied Mathematics 39, 207–229 (1992)
3. Graham, R., Lawler, E., Lenstra, J., Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 5, 287–326 (1979)
4. Horwitz, L.P., Karp, R.M., Miller, R.E., Winograd, S.: Index register allocation. Journal of the ACM 13, 43–61 (1966)
5. Maier, D.: The complexity of some problems on subsequences and supersequences. Journal of the ACM 25, 322–336 (1978)
6. Pereira, F.M.Q.: A survey on register allocation (2008), http://compilers.cs.ucla.edu/fernando/publications/drafts/survey.pdf
7. Reidenbach, D., Schmid, M.L.: A polynomial time match test for large classes of extended regular expressions. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 241–250. Springer, Heidelberg (2011)

Syntactic Complexity of Ultimately Periodic Sets of Integers

Michel Rigo and Élise Vandomme*

Institute of Mathematics, University of Liège, Grande Traverse 12 (B 37),
B-4000 Liège, Belgium
{e.vandomme,M.Rigo}@ulg.ac.be

Abstract. We compute the cardinality of the syntactic monoid of the language $0^* \operatorname{rep}_b(m\mathbb{N})$ made of base b expansions of the multiples of the integer m. We also give lower bounds for the syntactic complexity of any (ultimately) periodic set of integers written in base b. We apply our results to some well studied problem: decide whether or not a b-recognizable sets of integers is ultimately periodic.

1 Introduction

Syntactic complexity has received some recent and renewed interest. See for instance [6] for some background and we quote "*in spite of suggestions that syntactic semigroups deserve to be studied further, relatively little has been done on the syntactic complexity of a regular language*". In this paper syntactic complexity is introduced in the framework of numeration systems.

We compute the syntactic complexity of the set $m\mathbb{N}$ written in base b, i.e., the cardinality $M_{b,m}$ of the syntactic monoid of the language $0^* \operatorname{rep}_b(m\mathbb{N})$ made of base b expansions of the multiples of the integer m. A similar problem was solved for the state complexity of the language $0^* \operatorname{rep}_b(m\mathbb{N})$, i.e., the number of states of its minimal automaton. As usual (m, n) denotes the GCD of m and n.

Theorem 1 (B. Alexeev [1]). *Let $b, m \geq 2$ be integers. Let N, M be such that $b^N < m \leq b^{N+1}$ and $(m,1) < (m,b) < \cdots < (m,b^M) = (m,b^{M+1}) = (m,b^{M+2}) = \cdots$. The minimal automaton of $0^* \operatorname{rep}_b(m\mathbb{N})$ has exactly*

$$\frac{m}{(m,b^{N+1})} + \sum_{t=0}^{\inf\{N,M-1\}} \frac{b^t}{(m,b^t)} \text{ states.}$$

For the binary system, the first few values of $M_{2,m}$ are given below. Let $b \geq 2$. In this paper, we obtain an explicit formula for $M_{b,m}$ as a consequence of Theorems 2, 3 and 4 where we discuss three cases: the constant m and the base b are coprime or, m is a power of b or, $m = b^n q$ with $(q, b) = 1$, $q \geq 2$ and $n \geq 1$. Furthermore, we provide lower bounds for the syntactic complexity of any ultimately periodic set of integers written in base b, i.e., any finite union

* With the support of a grant CGRI-FRS/CNRS Wallonie-Bruxelles International.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 477–488, 2011.

of arithmetic progressions. In the framework of numeration systems, syntactic complexity has an advantage in comparison to left or right quotients, we have the opportunity to work simultaneously on prefixes and suffixes of base b expansions, that is on most and least significant digits.

m	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
$M_{2,m}$	3	6	5	20	13	21	7	54	41	110	20	156	43	60	9	136	109	342	62	126	221

A motivation for this work comes from the following decision problem. Let S be an abstract numeration system built on a regular language. See [4, Chap. 3] for background. It is well-known that any ultimately periodic set is S-recognizable, i.e., it has a regular language of representations within the system S. An instance of the decision problem is given by an abstract numeration system S and a DFA accepting some S-recognizable set $X \subseteq \mathbb{N}$. The question is therefore to decide whether X is ultimately periodic or not. This problem was settled positively for integer base systems by Honkala in [10]. See also [2] and in particular [5] for a first order logic approach. Recently this decision problem was settled positively in [3] for a large class of numeration systems based on linear recurrence sequences. Considering this decision problem for any abstract numeration system turns out to be equivalent to the so-called ω-HD0L ultimate periodicity decision problem, see again [4], or [11]. In its full generality, this problem is still open.

Since syntactic complexity provides an alternative measure for the complexity of a regular language, one could try to develop new decision procedures based on the syntactic complexity instead of the state complexity of the corresponding languages. A step in that direction is to consider first integer base numeration systems. As a consequence of our results, we present such a procedure restricted to a prime base in Section 4.

In the next section, we recall basic definitions and fix notation. Section 3 contains our main results: Theorems 2, 3, 4 are about the particular sets $m\mathbb{N}$ and Propositions 2, 3 as well as Theorem 2 are about any periodic set. We end the paper with a procedure for the decision problem described above and we present some directions for future work.

2 Definitions

For $i \leq j$, we denote by $[\![i, j]\!]$ the interval of integers $\{i, i+1, \ldots, j-1, j\}$. A *deterministic finite automaton* (or DFA) over the alphabet A is a 5-tuple $\mathcal{A} = (Q, q_0, F, A, \delta)$ where Q is the set of states, q_0 is the initial state, F is the set of final states and $\delta : Q \times A^* \to Q$ is the (extended) transition function. We denote by $|u|$ the length of the word $u \in A^*$ and by $\#P$ the cardinality of P.

Integer Base Numeration Systems. Let $b \geq 2$ be an integer. We denote by A_b the canonical alphabet of digits $[\![0, b-1]\!]$. For any word $u = u_\ell \cdots u_0 \in A_b^*$, we define the *numerical value* of u as

$$\mathrm{val}_b(u) = \sum_{i=0}^{\ell} u_i \, b^i.$$

Note that $\mathrm{val}_b(uv) = \mathrm{val}_b(u)\, b^{|v|} + \mathrm{val}_b(v)$ for all $u, v \in A_b^*$. For any integer $n > 0$, we denote the usual base b expansion of n by $\mathrm{rep}_b(n)$. We assume that such a greedy expansion does not start with 0. By convention, $\mathrm{rep}_b(0)$ is the empty word ε. A set X of integers is said to be *b-recognizable* if the language $\mathrm{rep}_b(X) \subseteq A_b^*$ is a regular language accepted by some DFA.

A set $X \subseteq \mathbb{N}$ is *periodic* of *period* p if for all $n \in \mathbb{N}$, $n \in X \Leftrightarrow n + p \in X$. The period is always understood to be the minimal period of X. In particular, if $X \subseteq \mathbb{N}$ is periodic of period p, then for all $i, j \in \mathbb{N}$,

$$i \not\equiv j \bmod p \Rightarrow \exists r \in [\![0, p-1]\!] : (i + r \in X, j + r \notin X) \text{ or } (i + r \notin X, j + r \in X). \tag{1}$$

A set $X \subseteq \mathbb{N}$ is *ultimately periodic* of *period* p and *index* $I > 0$ if for all $n \geq I$, $n \in X \Leftrightarrow n + p \in X$ and exactly one of the two elements $I - 1, I + p - 1$ is in X. Again, index and period are always understood to be minimal. It is easy to see that any ultimately periodic set is b-recognizable for all bases $b \geq 2$.

Syntactic Complexity. Let L be a language over the finite alphabet A. The *context* of a word $u \in A^*$ with respect to L is given by the set of pairs

$$\mathtt{C}_L(u) = \{(x, y) \in A^* \times A^* \mid xuy \in L\}.$$

If L is clearly understood, we will simply write $\mathtt{C}(u)$. Define the *Myhill congruence* [12] of L by $u \leftrightarrow_L v$ if and only if, for all $x, y \in A^*$, $xuy \in L \Leftrightarrow xvy \in L$. In other words, $u \leftrightarrow_L v$ if and only if $\mathtt{C}_L(u) = \mathtt{C}_L(v)$. This congruence is also known as the *syntactic congruence* of L. The monoid $A^*/\!\leftrightarrow_L$ made of the equivalence classes of the relation $\leftrightarrow_L$, is the *syntactic monoid* of L. It is well-known that L is a regular language if and only if $A^*/\!\leftrightarrow_L$ is finite. The *syntactic complexity* of L is the cardinality of its syntactic monoid. If $X \subseteq \mathbb{N}$ is a b-recognizable set of integers, by extension we define the *syntactic complexity* of X (w.r.t. b) as the syntactic complexity of the language $0^* \,\mathrm{rep}_b(X)$.

Proposition 1. *Let L be a language over A. Two words $u, v \in A^*$ are such that $u \leftrightarrow_L v$ if and only if they perform the same transformation on the set of states of the minimal automaton $\mathcal{M} = (Q_L, q_{0,L}, F_L, A, \delta_L)$ of L, i.e., for all $r \in Q_L$, $\delta_L(r, u) = \delta_L(r, v)$. In particular, if u, v are such that $\delta_L(q_{0,L}, u) \neq \delta_L(q_{0,L}, v)$, then $u \not\leftrightarrow_L v$.*

Definition 1. *A language $L \subseteq A^*$ is* weakly n-definite, *if for any $x, y \in A^*$ satisfying $|x| \geq n$, $|y| \geq n$ and having the same suffix of length n, $x \in L$ if and only if $y \in L$ [13,7]. In other words, L can be written as $G \cup A^*F$ where F (resp. G) is finite and contains only words of length n (resp. less than n). Let $n \geq 1$. A language is* n-definite *if it is weakly n-definite and not weakly $(n-1)$-definite. One also finds the terminology* suffix testable *in the literature, see [14].*

3 Main Results

Let $m, x \geq 2$ be integers such that $(m, x) = 1$. We denote by $\mathrm{ord}_m(x)$ the order of x in the multiplicative group $U(\mathbb{Z}/m\mathbb{Z})$ made of the invertible elements in $\mathbb{Z}/m\mathbb{Z}$.

That is $\mathrm{ord}_m(x)$ is the smallest positive integer j such that $x^j \equiv 1 \bmod m$. In particular, $\mathrm{ord}_m(x)$ is the period of the sequence $(x^n \bmod m)_{n\geq 0}$.

We first consider the case where the base and the period are coprime. Interestingly, the syntactic complexity depends only on the period and not on the structure of the periodic set.

Theorem 2. *Let $m, b \geq 2$ be integers such that $(m, b) = 1$. If $X \subseteq \mathbb{N}$ is periodic of (minimal) period m, then its syntactic complexity is given by $m.\,\mathrm{ord}_m(b)$. In particular, this result holds for $X = m\mathbb{N}$.*

Proof. Let $X \subseteq \mathbb{N}$ be a periodic set of period m. For a word $w \in A_b^*$, the *context* (w.r.t. X and b) of w is $\mathtt{C}(w) = \{(x, y) \in A_b^* \times A_b^* \mid xwy \in 0^*\,\mathrm{rep}_b(X)\}$. Let $u, v \in A_b^*$. Let us first show that we have

$$u \leftrightarrow_{0^*\,\mathrm{rep}_b(X)} v \Leftrightarrow \mathtt{C}(u) = \mathtt{C}(v) \Leftrightarrow \begin{cases} \mathrm{val}_b(u) \equiv \mathrm{val}_b(v) & \bmod m, \\ |u| \equiv |v| & \bmod\ \mathrm{ord}_m(b). \end{cases} \tag{2}$$

Let α be a multiple of $\mathrm{ord}_m(b)$ such that $b^\alpha > m$. Since $(b^i \bmod m)_{i\geq 0}$ is a purely periodic sequence of period $\mathrm{ord}_m(b)$, it follows that $\mathrm{val}_b(u0^\alpha) \equiv \mathrm{val}_b(u) \bmod m$. Assume that $\mathrm{val}_b(u) \not\equiv \mathrm{val}_b(v) \bmod m$. Using (1) there exists $r \in [\![0, m-1]\!]$ such that $\mathrm{val}_b(u) + r \in X$ and $\mathrm{val}_b(v) + r \notin X$ (the other case is treated similarly). So $(\varepsilon, 0^{\alpha - |\,\mathrm{rep}_b(r)|}\,\mathrm{rep}_b(r))$ belongs to $\mathtt{C}(u)$ and not to $\mathtt{C}(v)$. Now assume that $\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod m$ and $|u| \not\equiv |v| \bmod \mathrm{ord}_m(b)$. In that case, we obtain that $\mathrm{val}_b(1u) \not\equiv \mathrm{val}_b(1v) \bmod m$ and we can proceed as in the first situation, there exists some $r \in [\![0, m-1]\!]$ such that $(1, 0^{\alpha - |\,\mathrm{rep}_b(r)|}\,\mathrm{rep}_b(r))$ belongs to $\mathtt{C}(u)$ and not to $\mathtt{C}(v)$.

Now proceed to the converse and assume that u, v are such that $\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod m$ and $|u| \equiv |v| \bmod \mathrm{ord}_m(b)$. For all $x, y \in A_b^*$, we have

$$\begin{aligned} \mathrm{val}_b(xuy) &= \mathrm{val}_b(x)\, b^{|u|+|y|} + \mathrm{val}_b(u)\, b^{|y|} + \mathrm{val}_b(y) \\ &\equiv \mathrm{val}_b(x)\, b^{|v|+|y|} + \mathrm{val}_b(v)\, b^{|y|} + \mathrm{val}_b(y) \equiv \mathrm{val}_b(xvy) \bmod m \end{aligned}$$

and we have again used the fact that the sequence $(b^i \bmod m)_{i\geq 0}$ is purely periodic of period $\mathrm{ord}_m(b)$.

To conclude the proof, by considering words of the kind $0^{\alpha - |\,\mathrm{rep}_b(r)| + j}\,\mathrm{rep}_b(r)$, for $r \in [\![0, m-1]\!]$ and $j \in [\![1, \mathrm{ord}_m(b)]\!]$, one can show that we have $m.\,\mathrm{ord}_m(b)$ non-empty classes of $\leftrightarrow_{0^*\,\mathrm{rep}_b(X)}$.

Remark 1. Let $I > 0$. This remark will be useful for the discussion in Section 4. With the same notation and assumptions as for the previous proof, for all $T \geq 0$ and for all $u \in A_b^*$, using (2) we have

$$u \leftrightarrow_{0^*\,\mathrm{rep}_b(X)} (1\,0^{\mathrm{ord}_m(b)-1})^{Tm}\, u.$$

Therefore, for each class of $\leftrightarrow_{0^*\,\mathrm{rep}_b(X)}$, there exists an arbitrarily large integer $k > I$ such that $\mathrm{rep}_b(k)$ belongs to this class.

Now consider the case where the period is a power of the base.

Theorem 3. *Let $b \geq 2$ and $m = b^n$ with $n \geq 1$. Then the syntactic complexity of $0^* \operatorname{rep}_b(m\mathbb{N})$ is given by $M_{b,m} = 2n + 1$.*

Proof. The words $\varepsilon, 0, \ldots, 0^n, 1, 10, \ldots, 10^{n-1}$ have pairwise different contexts w.r.t. the language $0^* \operatorname{rep}_b(m\mathbb{N})$. For $i = 0, \ldots, n$, $(10^{n-i}, \varepsilon)$ belongs to $\mathtt{C}(0^{i+\ell})$, for all $\ell \geq 0$, but does not belong to $\mathtt{C}(0^j)$ for $j < i$ nor $\mathtt{C}(10^k)$, for $0 \leq k \leq n-1$. In the same way, for $i = 1, \ldots, n$, $(\varepsilon, 0^i)$ belongs to $\mathtt{C}(10^{n-j})$, for $0 \leq j \leq i$, but not to $\mathtt{C}(10^j)$ for $j < n - i$. So the syntactic monoid of $0^* \operatorname{rep}_b(m\mathbb{N})$ has at least $2n + 1$ elements. Now consider some word $u \in A_b^+$. Write $u = v0^i$ where v is either empty or ends with a non-zero digit. If $i \geq n$, then $u \leftrightarrow_{0^* \operatorname{rep}_b(m\mathbb{N})} 0^n$. If $v \neq \varepsilon$ and $i < n$, then $u \leftrightarrow_{0^* \operatorname{rep}_b(m\mathbb{N})} 10^i$. If $v = \varepsilon$ and $i < n$, the case $u = 0^i$ was already considered.

Proposition 2. *Let $b \geq 2$. If $X \subseteq \mathbb{N}$ is a periodic set of (minimal) period $m = b^n$ with $n \geq 1$, then the syntactic complexity of $L = 0^* \operatorname{rep}_b(X)$ is greater than or equal to $n+1$. Moreover there exist arbitrarily large integers $t_1, \ldots, t_{n+1}$ such that the $n+1$ words $\operatorname{rep}_b(t_1), \ldots, \operatorname{rep}_b(t_{n+1})$ belong to different equivalence classes of $\leftrightarrow_L$.*

Proof. Let $X \subseteq \mathbb{N}$ be a periodic set of period b^n, $n \geq 1$. Note that there exist k words $s_1, \ldots, s_k$ of length n such that a word of length at least n belongs to L if and only if it belongs to $A_b^*\{s_1, \ldots, s_k\}$. By minimality of the period there exist $V \in A_b^*$, $\sigma, \tau \in A_b$ such that $\sigma \neq \tau$, $|V| = n - 1$, and for all $u \in A_b^*$, we have $\operatorname{val}_b(u\sigma V) \in X$ and $\operatorname{val}_b(u\tau V) \notin X$. (If that was not the case, the fact that a word w belongs to L would only depend on its suffix of length $n - 1$, so in particular, we would have $\operatorname{val}_b(w) \in X$ if and only if $\operatorname{val}_b(w) + b^{n-1} \in X$ for all words w. This contradicts the fact that b^n is the period of X.) In other words, L is a n-definite language (see Definition 1). Define an accessible DFA $\mathcal{A} = (Q, q_\varepsilon, F, A_b, \delta)$ where

$$Q_n = \{q_w \mid |w| = n\} \text{ and } Q = Q_n \cup \{q_w \mid |w| < n\}$$

and for all $u \in A_b^*$ such that $|u| < n$ and $a \in A_b$, we have $\delta(q_u, a) = q_{ua}$. Now if $|u| = n$, then $u = cx$ for some $c \in A_b$, $|x| = n - 1$ and we have $\delta(q_u, a) = q_{xa}$. Notice that $\mathcal{A}$ restricted to the states in Q_n is a strongly connected component isomorphic to the de Bruijn graph of order n over A_b. The set of final states of $\mathcal{A}$ is easily defined in such a way that the language accepted by $\mathcal{A}$ is L. In particular, a state in Q_n is final if and only if it is of the form q_{s_i} for $i \in [\![1, k]\!]$.

Consider the minimal automaton of L denoted by $\mathcal{M} = (Q_L, q_{0,L}, F_L, A_b, \delta_L)$ and the canonical morphism [9] of automata $\Phi : Q \to Q_L$ from $\mathcal{A}$ to $\mathcal{M}$ such that $\Phi(\delta(r, w)) = \delta_L(\Phi(r), w)$ for all $r \in Q$ and $w \in A_b^*$. Let $R := \Phi(Q_n)$. In other words, R is the set of states of $\mathcal{M}$ reached by words of length at least n. Using the same arguments as in [13], let us show that $\#R \geq n + 1$. For all $r, r' \in R$ and $i \geq 0$, define

$$E_i(r, r') \Leftrightarrow (\forall x \in A_b^*)\big[|x| \geq i \Rightarrow (\delta_L(r, x) \in F_L \Leftrightarrow \delta_L(r', x) \in F_L)\big].$$

This equivalence relation E_i over R induces a partition P_i of R into $\#P_i$ equivalence classes. It is clear that $E_i(r, r')$ implies $E_{i+1}(r, r')$ and thus $\#P_i \geq \#P_{i+1}$.

Consider the words σV and τV introduced in the first part of this proof with $V = v_1 \cdots v_{n-1}$. Let $T \geq n$ and $i \in [\![0, n-1]\!]$. Take the two states $r = \Phi(\delta(q_\varepsilon, 10^T \sigma v_1 \cdots v_{n-i-1}))$ and $r' = \Phi(\delta(q_\varepsilon, 10^T \tau v_1 \cdots v_{n-i-1}))$ in R. By considering the word $v_{n-i} \cdots v_{n-1}$ of length i, the states r and r' do not satisfy $E_i(r, r')$ but for all words u of length at least $i+1$, we have

$$\delta(q_\varepsilon, 10^T \sigma v_1 \cdots v_{n-i-1} u) = q_S = \delta(q_\varepsilon, 10^T \tau v_1 \cdots v_{n-i-1} u)$$

where S is the suffix of length n of $v_1 \cdots v_{n-i-1} u$ and thus $E_{i+1}(r, r')$. We have just shown that E_i is a refinement of E_{i+1} and $\#P_0 > \#P_1 > \cdots > \#P_{n-1} > \#P_n \geq 1$. Consequently, $\#R \geq \#P_0 \geq n+1$.

The minimal automaton $\mathcal{M}$ of L contains at least $n+1$ distinct states of the kind $\Phi(q_{u_1}), \ldots, \Phi(q_{u_{n+1}}) \in R$ for some words $u_1, \ldots, u_{n+1} \in A_b^*$ of length n. Let $I > 0$. Take a large enough T such that, for all $i \in [\![1, n+1]\!]$, $\mathrm{val}_b(10^T u_i) > I$ and observe that

$$\Phi(\delta(q_\varepsilon, 10^T u_i)) = \Phi(q_{u_i}) = \delta_L(q_{0,L}, 10^T u_i) \in R.$$

The words $10^T u_i$, $i = 1, \ldots, n+1$, perform pairwise distinct transformations on the set of states of the minimal automaton $\mathcal{M}$ and the syntactic monoid of L contains at least $n+1$ classes (see Proposition 1).

Remark 2. The bound in Proposition 2 is tight. One can for instance consider the set $5 + 8\mathbb{N}$. The corresponding syntactic monoid has exactly four infinite equivalence classes.

We now turn to the case where $m = b^n q$ with $(q, b) = 1$ and $n \geq 1$. For convenience, we set $s = \mathrm{ord}_q(b)$ in all what follows.

Remark 3. The sequence $(b^i \bmod m)_{i \geq 0}$ is ultimately periodic of period $\mathrm{ord}_q(b)$ and preperiod n. For instance, the sequence $(2^i \bmod 24)_{i \geq 0} = (1, 2, 4, (8, 16)^\omega)$ has preperiod $3 = \log_2 8$ and period $\mathrm{ord}_3(2)$. Indeed, $(b^i \bmod q)_{i \geq 0}$ is purely periodic of period $\mathrm{ord}_q(b)$ and $(b^i \bmod b^n)_{i \geq 0} = (1, b, b^2, \ldots, b^{n-1}, 0, 0, \ldots)$ is ultimately periodic with preperiod n and period 1.

Lemma 1. *Let $b \geq 2$ and $m = b^n q$ where $n \geq 1$ and $(q, b) = 1$ and $q \geq 2$. Let $X \subseteq \mathbb{N}$ be a periodic set of (minimal) period m. For any words $u, v \in A_b^*$ of length at least n, the following implications hold*

$$((|u| \equiv |v| \bmod s) \wedge (\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod m)) \Rightarrow u \leftrightarrow_{0^* \,\mathrm{rep}_b(X)} v, \tag{3}$$

$$((|u| \not\equiv |v| \bmod s) \vee (\mathrm{val}_b(u) \not\equiv \mathrm{val}_b(v) \bmod q)) \Rightarrow u \not\leftrightarrow_{0^* \,\mathrm{rep}_b(X)} v. \tag{4}$$

Proof. The arguments are similar to the ones developed in the proof of Theorem 2. Let $u, v \in A_b^*$ be two words of length at least n. Using the fact that $b^k \equiv b^{k+s} \bmod m$, for all $k \geq n$, notice that if u, v are such that $|u| \equiv |v| \bmod s$ and $\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod m$, then, for all $x, y \in A_b^*$, $\mathrm{val}_b(xuy) \equiv \mathrm{val}_b(xvy) \bmod m$ which means that $u \leftrightarrow_{0^* \,\mathrm{rep}_b(X)} v$. We have just shown that (3) holds.

Now we want to show that (4) holds. The reader may notice that a main difference between (3) and (4) is that congruences of numerical values are considered modulo m and q respectively. Let $\alpha > 0$ be such that $b^\alpha \geq m$. As a first case, suppose that $\mathrm{val}_b(u) \not\equiv \mathrm{val}_b(v) \bmod q$. Since $(b, q) = 1$, we get that $\mathrm{val}_b(u)b^\alpha \not\equiv \mathrm{val}_b(v)b^\alpha \bmod q$. Hence $\mathrm{val}_b(u)\, b^\alpha \not\equiv \mathrm{val}_b(v)\, b^\alpha \bmod m$ and, using (1), there exists $r \in [\![0, m-1]\!]$ such that $\mathrm{val}_b(u)b^\alpha + r \in X$ and $\mathrm{val}_b(v)b^\alpha + r \notin X$ (the other case is treated similarly). We can conclude that $(\varepsilon, 0^{\alpha - |\,\mathrm{rep}_b(r)|}\,\mathrm{rep}_b(r))$ belongs to $\mathtt{C}(u)$ and not to $\mathtt{C}(v)$. As a second case, suppose that $\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod q$ but that $|u| \not\equiv |v| \bmod s$. This implies that $b^{|u|} \not\equiv b^{|v|} \bmod q$. Therefore $\mathrm{val}_b(1u) \not\equiv \mathrm{val}_b(1v) \bmod q$ and we proceed as in the first case. There exists $r \in [\![0, m-1]\!]$ such that $(1, 0^{\alpha - |\,\mathrm{rep}_b(r)|}\,\mathrm{rep}_b(r))$ belongs to $\mathtt{C}(u)$ and not to $\mathtt{C}(v)$.

Corollary 1. *Let $b \geq 2$ and $m = b^n q$ where $n \geq 1$ and $(q, b) = 1$ and $q \geq 2$. If $X \subseteq \mathbb{N}$ be a periodic set of (minimal) period m and $u, v \in A_b^*$ are two words of length at least n, then $u \leftrightarrow_{0^*\,\mathrm{rep}_b(X)} v$ implies that the following conditions hold*

i) $|u| \equiv |v| \bmod s$

ii) either, $\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod m$ *or, there exists a unique* $i \in [\![0, n-1]\!]$ *such that* $(\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod b^i q) \wedge (\mathrm{val}_b(u) \not\equiv \mathrm{val}_b(v) \bmod b^{i+1} q)$.

Remark 4. The converse does not hold. Consider $X = 12\mathbb{N}$ and $b = 2$, one can check with $u = 100$ and $v = 10110$ that i) and ii) are fulfilled but $(1, \varepsilon)$ belongs to $\mathtt{C}(u)$ but not to $\mathtt{C}(v)$.

Proof. Using (4), we conclude that $|u| \equiv |v| \bmod s$. Proceed by contradiction and assume that ii) does not hold. In other words, $\mathrm{val}_b(u) \not\equiv \mathrm{val}_b(v) \bmod m$ and for all $i \in [\![0, n-1]\!]$,

$$(\mathrm{val}_b(u) \not\equiv \mathrm{val}_b(v) \bmod b^i q) \vee (\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod b^{i+1} q)$$

but with (4), we get $\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod q$ and since the above condition holds for $i = 0$, we must have $\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod qb$. Now using the same above condition for $i = 1$, we must have $\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod qb^2$ and iterating the argument, we must have $\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod qb^n$ contradicting the fact that $\mathrm{val}_b(u) \not\equiv \mathrm{val}_b(v) \bmod m$.

A word $u \in A_b^*$ of length at least n has three features: its length modulo s, $|u| \bmod s$, its value modulo q, $\mathrm{val}_b(u) \bmod q$, and its suffix of length n (i.e., its value modulo b^n). Observe that if we know $\mathrm{val}_b(u) \bmod q$ and $\mathrm{val}_b(u) \bmod b^n$, then we obtain $\mathrm{val}_b(u) \bmod b^n q$.

Lemma 2. *Let $q, b \geq 2$ be two coprime integers, $w \in A_b^*$ and $i \in [\![0, q-1]\!]$. There exists some word $x \in (0^{s-1}1)^*0^*$ such that $\mathrm{val}_b(xw) \equiv i \bmod q$.*

Proof. Let $L \geq \max\{n, |w|\}$ be smallest multiple of $\varphi(q)$, where φ is the Euler totient. We set $z = \mathrm{val}_b(w) \bmod q$. Using the periodicity of $(b^k \bmod q)_{k \geq 0}$, we have

$$\mathrm{val}_b[(0^{s-1}1)^{i+q-z}\, 0^{L-|w|} w] \equiv \sum_{k=0}^{i+q-z-1} b^{L+ks} + z \equiv i \bmod q.$$

Corollary 2. *For all word $w \in A_b^*$ of length at least n, $i \in [\![0, q-1]\!]$, $\ell \in [\![0, s-1]\!]$ and $I > 0$, there exists a word u having w as suffix and such that $\mathrm{val}_b(u) \equiv i \bmod q$, $|u| \equiv \ell \bmod s$ and $\mathrm{val}_b(u) > I$.*

Proof. With the previous lemma, there exists x such that $\mathrm{val}_b(xw) \equiv i \bmod q$. To conclude the proof, one has to add a prefix of the kind $0^r(0^{s-1}1)^{tq}$.

Theorem 4. *Let $b \geq 2$ and $m = b^n q$ where $n \geq 1$ and $(q, b) = 1$ and $q \geq 2$. Then the syntactic complexity of $0^* \mathrm{rep}_b(m\mathbb{N})$ is given by*

$$M_{b,m} = (n+1).M_{b,q} + n = (n+1).q.\mathrm{ord}_q(b) + n.$$

Proof. Let $X = m\mathbb{N}$. Notice that $d \in \mathbb{N}$ is a multiple of m if and only if $\mathrm{rep}_b(d) = x0^n$ and $\mathrm{val}_b(x)$ is a multiple of q. We get the first term in the formula. Let u, v be two words of length at least n. From (4), if $|u| \not\equiv |v| \bmod s$ or $\mathrm{val}_b(u) \not\equiv \mathrm{val}_b(v) \bmod q$, then u and v belong to two different classes for $\leftrightarrow_{0^* \mathrm{rep}_b(m\mathbb{N})}$. Now assume that $|u| \equiv |v| \bmod s$ and $\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod q$. Observe that if u ends with exactly $i \in [\![0, n-1]\!]$ zeroes and v ends with j zeroes with $j > i$, then $u \not\leftrightarrow_{0^* \mathrm{rep}_b(m\mathbb{N})} v$. With the same construction as in the proof of Lemma 2, there exists some word $x \in (0^{s-1}1)^*0^*$ such that $\mathrm{val}_b(xu) \equiv \mathrm{val}_b(xv) \equiv 0 \bmod q$ and we can conclude that $(x, 0^{\max\{n-j,0\}})$ belongs to $\mathtt{C}(v)$ and not $\mathtt{C}(u)$. This proves with Corollary 2 that words of length at least n provide the syntactic monoid of $0^* \mathrm{rep}_b(m\mathbb{N})$ with at least $(n+1).q.s$ classes.

Now observe that for any two words u, v of length at least n such that $|u| \equiv |v| \bmod s$, $\mathrm{val}_b(u) \equiv \mathrm{val}_b(v) \bmod q$ and u, v, either end with exactly the same number i of zeroes with $i < n$, or both end with at least n zeroes, then $u \leftrightarrow_{0^* \mathrm{rep}_b(m\mathbb{N})} v$. This proves that words of length at least n provide the syntactic monoid of $0^* \mathrm{rep}_b(m\mathbb{N})$ with no more than $(n+1).q.s$ classes.

Now we take into account words of length less than n and show that they provide the syntactic monoid of $0^* \mathrm{rep}_b(m\mathbb{N})$ with n new classes giving the second term in the expression of $M_{b,m}$. The n words $\varepsilon, 0, \ldots, 0^{n-1}$ have pairwise different contexts: for all $\ell \in [\![0, n-1]\!]$,

$$\mathtt{C}(0^\ell) \cap \{(\mathrm{rep}_b(q), 0^k) \mid k \geq 0\} = \{(\mathrm{rep}_b(q), 0^k) \mid k \geq n - \ell\}.$$

Let $\ell \in [\![0, n-1]\!]$. We show by contradiction that they are indeed n new classes. Assume that there exists $u \in A_b^*$ such that $|u| \geq n$ and $u \leftrightarrow_{0^* \mathrm{rep}_b(m\mathbb{N})} 0^\ell$. Since $(\varepsilon, \varepsilon)$ belongs to $\mathtt{C}(0^\ell)$, we deduce that $val_b(u) \equiv 0 \bmod m$. We have

$$\mathrm{val}_b(1u) \equiv b^{|u|} \bmod m \text{ and } \mathrm{val}_b(10^\ell) \equiv b^\ell \bmod m$$

but since the sequence $(b^k \bmod m)_{k \geq 0}$ is ultimately periodic with period s and preperiod n, from $\ell < n \leq |u|$ we conclude that

$$\mathrm{val}_b(1u) \not\equiv \mathrm{val}_b(10^\ell) \bmod m.$$

Let $t \geq n$ be a multiple of s. We have

$$\mathrm{val}_b((0^{t-1}1)^m u) \equiv \mathrm{val}_b(u) \equiv 0 \bmod m$$

and

$$\mathrm{val}_b((0^{t-1}1)^m\,0^\ell) = b^\ell + \sum_{k=1}^{m-1} b^{\ell+kt} \equiv b^\ell + (m-1)b^{\ell+t} \bmod m.$$

Since $\ell < n \leq \ell + t$, we have $b^\ell \not\equiv b^{\ell+t} \bmod m$. We conclude that

$$\mathrm{val}_b((0^{t-1}1)^m\,0^\ell) \not\equiv 0 \bmod m$$

proving that $((0^{t-1}1)^m, \varepsilon)$ belongs to $\mathtt{C}(u)$ and not to $\mathtt{C}(0^\ell)$.

To conclude the proof, we have to consider some word $u \notin 0^*$ of length less than n and prove that u does not provide any new equivalence class. This comes from the fact that $u \leftrightarrow_{0^*\,\mathrm{rep}_b(m\mathbb{N})} 0^{ks}u$ where k is chosen large enough such that $ks + |u| \geq n$. It is enough to show that $u \leftrightarrow_{0^*\,\mathrm{rep}_b(m\mathbb{N})} 0^s u$. Let $x, y \in A_b^*$. If $|uy| \geq n$ then $\mathrm{val}_b(xuy) \equiv \mathrm{val}_b(x0^s uy) \bmod m$ and $|xuy| \equiv |x0^s uy| \bmod s$ and we can use (3). Otherwise $|uy| < n$ and since $u \notin 0^*$, this means that we simultaneously have $\mathrm{val}_b(xuy) \not\equiv 0 \bmod m$ and $\mathrm{val}_b(x0^s uy) \not\equiv 0 \bmod m$. This means that $xuy \in 0^*\,\mathrm{rep}_b(m\mathbb{N})$ if and only if $x0^s uy \in 0^*\,\mathrm{rep}_b(m\mathbb{N})$.

Proposition 3. *Let b be a prime number and $m = b^n q$ where $n \geq 1$ and $(q, b) = 1$ and $q \geq 2$. If $X \subseteq \mathbb{N}$ is periodic of (minimal) period m, then the syntactic complexity of $0^*\,\mathrm{rep}_b(X)$ is greater than or equal to $(n+1).q$. Moreover there exist arbitrarily large integers $t_1, \ldots, t_{(n+1)q}$ such that $\mathrm{rep}_b(t_1), \ldots, \mathrm{rep}_b(t_{(n+1)q})$ belong to different equivalence classes of $\leftrightarrow_{0^*\,\mathrm{rep}_b(X)}$.*

Proof. Take a periodic set X of period $m = b^n q$. Consider the characteristic word $(x_t)_{t\geq 0} \in \{0,1\}^\omega$ of X where $x_t = 1$ if and only if $t \in X$. This infinite word is periodic of period m. The q infinite words $(x_{qt})_{t\geq 0}$, $(x_{qt+1})_{t\geq 0}$, $\ldots$, $(x_{qt+q-1})_{t\geq 0}$ are periodic and their own period divides b^n. By minimality of the period m of X, there exists $i \in [\![0, q-1]\!]$ such that $(x_{qt+i})_{t\geq 0}$ has period[1] exactly b^n. For any $T \geq 0$ such that $|\,\mathrm{rep}_b(qT+i)| \geq n$, consider the b^n words

$$\mathrm{rep}_b(qT+i),\ \mathrm{rep}_b(q(T+1)+i), \ldots, \mathrm{rep}_b(q(T+b^n-1)+i).$$

Notice that these words have pairwise different suffixes of length n. Indeed, proceed by contradiction and assume j, k are such that $0 \leq j < k < b^n$ and $\mathrm{rep}_b(q(T+j)+i)$ and $\mathrm{rep}_b(q(T+k)+i)$ have the same suffix of length n, then $q(k-j) \equiv 0 \bmod b^n$. Since $(q, b) = 1$, we get $k \equiv j \bmod b^n$ which is a contradiction.

Since $(x_{qt+i})_{t\geq 0}$ has (minimal) period b^n, there exists $j \in [\![0, (b-1)b^{n-1} - 1]\!]$ such that $q(T+j)+i \in X$ and $q(T+j+b^{n-1})+i \notin X$ (or equivalently $q(T+j)+i \notin X$ and $q(T+j+b^{n-1})+i \in X$). We can assume that there exist a word V of length $n-1$, some prefixes p, p' and two distinct symbols $\sigma, \tau \in A_b$ such that

$$\mathrm{rep}_b(q(T+j)+i) = p\sigma V \in 0^*\,\mathrm{rep}_b(X) \text{ and}$$

[1] If b is a composite number of the kind $b = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$, then there exists i such that $(x_{qt+i})_{t\geq 0}$ has period $p_1^{\beta_1} \cdots p_k^{\beta_k}$ where $\max \alpha_r = \max \beta_r$ which would lead to extra technicalities in the discussion.

$$\mathrm{rep}_b(q(T+j+b^{n-1})+i) = p'\tau V \notin 0^*\,\mathrm{rep}_b(X).$$

In particular, if $\mathrm{val}_b(u) \equiv i \bmod q$ and u has σV (resp. τV) as a suffix, then u belongs to $0^*\,\mathrm{rep}_b(X)$; $0^*\,\mathrm{rep}_b(X \cap (q\mathbb{N}+i))$ is n-definite. We use the same kind of construction as in the second part of the proof of Proposition 2. Let

$$S_i = \{w \in A_b^* \mid (\mathrm{val}_b(w) \equiv i \bmod q) \wedge (|w| \geq n) \wedge (|w| \equiv 0 \bmod s)\}.$$

For all $j \geq 0$, define the equivalence relation $\sim_j$ over S_i by

$$u \sim_j v \Leftrightarrow (\forall x, y \in A_b^*)[|y| \geq j \Rightarrow (xuy \in 0^*\,\mathrm{rep}_b(X) \Leftrightarrow xvy \in 0^*\,\mathrm{rep}_b(X))].$$

Note that $\sim_0$ is exactly $\leftrightarrow_{0^*\,\mathrm{rep}_b(X)}$ and $u \sim_j v$ implies $u \sim_{j+1} v$. Denote by $\#P_j$ the number of equivalence classes of $\sim_j$ over S_i. Write V as $v_1 \cdots v_{n-1}$. For $k \in [\![0, n-1]\!]$, consider the words $p\sigma v_1 \dots v_{n-k-1}$ and $p'\tau v_1 \dots v_{n-k-1}$. Using Corollary 2, there exist two words x_k, y_k such that $r_k := x_k p\sigma v_1 \dots v_{n-k-1}$ and $t_k := y_k p'\tau v_1 \dots v_{n-k-1}$ belong to S_i. We have $r_k \not\sim_k t_k$ but $r_k \sim_{k+1} t_k$ proving that $\#P_k > \#P_{k+1}$. Indeed, with the same construction as in the proof of Lemma 2 there exists some word x such that $|x| \equiv 0 \bmod s$ and

$$\mathrm{val}_b(xr_k v_{n-k} \cdots v_{n-1}) \equiv \mathrm{val}_b(xt_k v_{n-k} \cdots v_{n-1}) \equiv i \bmod q$$

and since $xr_k v_{n-k} \cdots v_{n-1}$ (resp. $xt_k v_{n-k} \cdots v_{n-1}$) has suffix σV (resp. τV) it belongs to (resp. does not belong to) $0^*\,\mathrm{rep}_b(X)$. For any word y of length at least $k+1$, the two words $r_k y$ and $t_k y$ have the same suffix of length n and have numerical values congruent modulo q. Therefore, their values are also congruent modulo m and $r_k y$ and $t_k y$ have the same length modulo s. Using (3), we get $r_k y \leftrightarrow_{0^*\,\mathrm{rep}_b(X)} t_k y$ and $r_k \sim_{k+1} t_k$. We conclude that we have a partition of S_i into at least $n+1$ congruence classes for $\sim_0$. Let $w_1, \dots, w_{n+1}$ be words in S_i belonging to pairwise distinct classes. The $(n+1).q$ words[2]

$$(0^{s-1}1)^k w_j,\ j = 1, \dots, n+1,\ k = 0, \dots, q-1$$

belong to pairwise distinct equivalence classes for $\leftrightarrow_{0^*\,\mathrm{rep}_b(X)}$. Indeed, consider the two words $x = (0^{s-1}1)^k w_j$ and $y = (0^{s-1}1)^{k'} w_{j'}$. Observe that $|w_j| \equiv |w_{j'}| \equiv 0 \bmod s$. If $k \neq k'$, then $\mathrm{val}_b(x) \not\equiv \mathrm{val}_b(y) \bmod q$ and we can use (4). To conclude with the proof, assume that $k = k'$ and $j \neq j'$. Assume that $x \leftrightarrow_{0^*\,\mathrm{rep}_b(X)} y$, then we get

$$w_j \leftrightarrow_{0^*\,\mathrm{rep}_b(X)} (0^{s-1}1)^{q-k} x \leftrightarrow_{0^*\,\mathrm{rep}_b(X)} (0^{s-1}1)^{q-k} y \leftrightarrow_{0^*\,\mathrm{rep}_b(X)} w_{j'}$$

which is a contradiction. To get the first equivalence, observe that the two words have the same suffix of length n (so $\mathrm{val}_b(w_j) \equiv \mathrm{val}_b((0^{s-1}1)^{q-k}x) \bmod b^n$) and also $\mathrm{val}_b(w_j) \equiv \mathrm{val}_b((0^{s-1}1)^{q-k}x) \bmod q$, thus we can apply (3).

Since for all words w of length at least n, $w \leftrightarrow_{0^*\,\mathrm{rep}_b(X)} (0^{s-1}1)^q w$, the elements of the different equivalence classes can be chosen arbitrarily large.

[2] Note that all these words have length congruent to 0 modulo s. By considering words of the kind $0^r(0^{s-1}1)^k w_j$ for $r \in \{0, \dots, s-1\}$, using (4) we can even improve the result to obtain $(n+1).q.s$ distinct equivalence classes for $\leftrightarrow_{0^*\,\mathrm{rep}_b(X)}$.

4 Application to a Decision Procedure

Let $X \subseteq \mathbb{N}$ be a b-recognizable set of integers such that $0^* \operatorname{rep}_b(X)$ is accepted by some DFA $\mathcal{A}$. A usual technique for deciding whether or not X is ultimately periodic is to prove that whenever X is ultimately periodic, then its period and its preperiod must be bounded by some quantities depending only on the size of the DFA $\mathcal{A}$. Therefore, one has a finite number of admissible periods and preperiods to test leading to a decision procedure. For details, see [3]. In particular, the following result [3, Prop. 44] stated in full generality for any abstract numeration system (i.e., the language of numeration is a regular language) shows that we have only to obtain an upper bound on the admissible periods.

Proposition 4. *Let $S = (L, \Sigma, <)$ be an abstract numeration system. If $X \subseteq \mathbb{N}$ is an ultimately periodic set of period p_X such that $\operatorname{rep}_S(X)$ is accepted by a DFA with d states, then the preperiod of X is bounded by an effectively computable constant C depending only on d and p_X.*

The following result is a consequence of Theorem 2, Propositions 2 and 3.

Theorem 5. *Let b be a prime number. If $X \subseteq \mathbb{N}$ is an ultimately periodic set of period $p_X = b^n q$ with $(q, b) = 1$ and $n \geq 0$, then the syntactic complexity of $0^* \operatorname{rep}_b(X)$ is greater than or equal to $(n+1)q$.*

Proof. Let I be the preperiod of X. Even if Remark 1, Propositions 2 and 3 are about (purely) periodic sets of integers, if we consider instead an ultimately periodic set, since we can choose words belonging to different equivalence classes in such a way that their numerical value is greater than I, then the lower bound on the number of classes is still valid for the ultimately periodic case.

Assume that $b \geq 2$ is a prime number[3]. Therefore, giving a DFA $\mathcal{A}$ accepting $0^* \operatorname{rep}_b(X)$ and so the corresponding syntactic monoid, if X is ultimately periodic, then we get an upper bound on its period.

5 Further Work

We will try to extend the present work to a wider class of numeration systems. For instance, for the Fibonacci numeration system (defined by the sequence $F_{n+2} = F_{n+1} + F_n$, $F_0 = 1$, $F_1 = 2$) where integers are represented using the greedy algorithm, the syntactic complexity of $0^* \operatorname{rep}_F(m\mathbb{N})$ is given by $M_{F,m} = 4.m^2.P_F(m) + 2$ where $P_F(m)$ is the period of $(F_i \bmod m)_{i \geq 0}$. The proof essentially follows the same lines as in the proof of Theorem 2 and [8]. Recall that a word is a valid representation if it does not contain a factor 11. This later fact explains the factor 4 in the expression $M_{F,m}$. Let $u = u_k \cdots u_0, v = v_\ell \cdots v_0 \in \{0,1\}^*$. We have $u \leftrightarrow_{0^* \operatorname{rep}_F(m\mathbb{N})} v$ if and only if

[3] If b is not a prime number, there are integers of the kind $m = b^n q$ where n is maximal and $(b, q) > 1$, as an example take $b = 4$ and $m = 72 = 4.18$. Such a situation is not taken into account by Theorem 2, Propositions 2 and 3.

$$(\mathrm{val}_F(u) \equiv \mathrm{val}_F(v) \bmod m) \wedge (\mathrm{val}_F(u0) \equiv \mathrm{val}_F(v0) \bmod m) \wedge \\ (|u| \equiv |v| \bmod P_F(m)) \wedge (u_k = v_\ell) \wedge (u_0 = v_0).$$

For the Tribonacci numeration system, the syntactic complexity of $0^* \operatorname{rep}_T(m\mathbb{N})$ is given by $M_{T,m} = 9.m^3.P_T(m) + 3$.

Acknowledgments

We would like to thank Émilie Charlier for her useful comments to improve this paper.

References

1. Alexeev, B.: Minimal DFA for testing divisibility. J. Comput. System Sci. 69(2), 235–243 (2004)
2. Allouche, J.P., Rampersad, N., Shallit, J.: Periodicity, repetitions, and orbits of an automatic sequence. Theoret. Comput. Sci. 410(30-32), 2795–2803 (2009)
3. Bell, J.P., Charlier, E., Fraenkel, A.S., Rigo, M.: A decision problem for ultimately periodic sets in non-standard numeration systems. Int. J. Algebra and Computation 19, 809–839 (2009)
4. Berthé, V., Rigo, M. (eds.): Combinatorics, Automata and Number Theory. Encyclopedia of Mathematics and its Applications, vol. 135. Cambridge University Press, Cambridge (2010)
5. Bruyère, V., Hansel, G., Michaux, C., Villemaire, R.: Logic and p-recognizable sets of integers. Bull. Belg. Math. Soc. 1, 191–238 (1994)
6. Brzozowski, J., Ye, Y.: Syntactic complexity of ideal and closed languages. preprint arXiv:1010.3263v1 (2010)
7. Brzozowski, J.A.: Canonical regular expressions and minimal state graphs for definite events. In: Fox, J. (ed.) Proceedings of the Symposium on Mathematical Theory of Automata New York. MRI Symposia Series, vol. 12, pp. 529–561. Polytechnic Press of the Polytechnic Institute of Brooklyn (1963)
8. Charlier, E., Rampersad, N., Rigo, M., Waxweiler, L.: The minimal automaton recognizing m$\mathbb{N}$ in a linear numeration system. Integers (to appear)
9. Eilenberg, S.: Automata, Languages, and Machines, vol. A. Academic Press, London (1974)
10. Honkala, J.: A decision method for the recognizability of sets defined by number systems. Theor. Inform. Appl. 20, 395–403 (1986)
11. Honkala, J., Rigo, M.: A note on decidability questions related to abstract numeration systems. Discrete Math. 285, 329–333 (2004)
12. Myhill, J.: Finite automata and representation of events. Technical Report 57-624, Wright Air Development Center Technical Report (1957)
13. Perles, M., Rabin, M.O., Shamir, E.: The theory of definite automata. IEEE Trans. Electron. Comp., 233–243 (1963)
14. Pin, J.E.: Syntactic semigroups. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Language Theory, vol. I, pp. 679–746. Springer, Heidelberg (1997)

Undecidability of the State Complexity of Composed Regular Operations*

Arto Salomaa[1], Kai Salomaa[2], and Sheng Yu[3]

[1] Turku Centre for Computer Science,
Joukahaisenkatu 3-5 B, 20520 Turku, Finland
asalomaa@utu.fi
[2] School of Computing,
Queen's University,
Kingston, Ontario, K7L 3N6, Canada
ksalomaa@cs.queensu.ca
[3] Department of Computer Science,
The University of Western Ontario,
London, Ontario, N6A 5B7, Canada
syu@csd.uwo.ca

Abstract. We consider the regularity-preserving operations of intersection and marked catenation and construct an infinite sequence $C_i, i = 1, 2, \ldots$, of compositions formed from the two operations. We construct also an infinite sequence of polynomials $S_i, i = 1, 2, \ldots$, with positive integer coefficients. As a main result we prove that it is undecidable whether or not S_i is a state complexity function of C_i. All languages needed are over a fixed alphabet with at most 50 letters. We also consider some implications and generalizations, as well as present some open problems.

Keywords: finite deterministic automaton, state complexity, undecidability, regularity-preserving operations, composition of operations.

1 Introduction

The study of *state complexity* has been one of the most vigorously investigated research areas in automata theory. It is well known that, for every regular language L, there is a unique finite deterministic automaton accepting L and minimal with the respect to the number of states. The effect of a regularity-preserving operation on the number of states is customarily referred to as the *state complexity* of that operation. For instance, if $L_i, 1 \leq i \leq 3$, are regular languages accepted by automata with x_i states, respectively, how many states does the union $L_1 \cup L_2 \cup L_3$ require in terms of the numbers x_i?

The effect of basic regularity-preserving operations was settled in [9]. The recent study of state complexity has been motivated by many new applications of

* This work is supported by Natural Science and Engineering Council of Canada Discovery Grants 147224 and 41630.

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 489–498, 2011.

automata, e.g., in natural language and speech processing, software engineering, and parallel processing, which utilize finite automata of very large sizes. The state complexity gives a good estimate of the size of the application and a lower bound of its time and space complexities.

Apart from the basic operations alone, also combined operations have been very much investigated [5,2,7,1]. The worst-case state complexity of the composition of two operations can be smaller than the one obtained directly from the (known) complexities of the two operations. For instance, the state complexity of the star operation on the result of the union of two regular languages, with the state complexities m and n, respectively, is $2^{m+n-1} - 2^{m-1} - 2^{n-1} + 1$. However, the mathematical composition of the two state complexities is $2^{mn-1} + 2^{mn-2}$, which is much higher than the actual state complexity [5].

Could there be a general method of determining the state complexity of arbitrary compositions of operations? If such a method existed, then studies concerning compositions of individual operations would become unnecessary. However, the undecidability results presented in this paper render the search of such a general method futile. Many known state complexity results require exponentiation [8]. However, it will be shown below that undecidability can be obtained already in the simple case where the proposed state complexity function is a polynomial with positive integer coefficients.

In our undecidability proof we will use reduction to *Hilbert's Tenth Problem*. Since we are dealing with polynomials, the arithmetical operations under consideration are *product* and *sum*. Two natural regularity-preserving operations associated with these operations are *marked catenation* and *intersection*, respectively. Therefore, the composition sequences considered below will mainly consist of these two operations. If reduction to some other undecidable problem is applied, then in general some other regularity-preserving operations must be used in compositions.

A brief outline of the contents of the paper follows. In Section 2 the undecidability of Hilbert's Tenth Problem is modified to a form suitable for our purposes. Also estimates concerning the degree of the polynomials, the number of variables, as well as the size of the overall alphabet are given. The following section contains a general discussion about state complexity and auxiliary results concerning the two relevant operations. The actual proof of undecidability is given in Section 4. In the final section we outline some possible variations of the undecidability result, as well as some generalizations of our results concerning the interconnection between arithmetical and regularity-preserving operations.

2 Diophantine Preliminaries

We begin with the universal unsolvability of *Hilbert's Tenth Problem*. For a proof of the following result, see Theorem 3.10 in [3].

Theorem 1. *There is a polynomial $Q(x_0, x_1, \ldots, x_m)$ with integer coefficients such that no algorithm exists for deciding whether or not an arbitrary equation of the form*

$$Q(x_0, x_1, \ldots, x_m) = 0,$$

where x_0 is a given positive integer, has a solution in nonnegative integers $x_1, \ldots, x_m$.

We will now modify this result to a form more suitable for our purposes.

Denote by $Q_i(x_1, \ldots, x_m)$, $i = 1, 2, \ldots$, the polynomial resulting from Q when the variable x_0 is substituted by the positive integer i. Consider the inequalities

$$0 \leq 2(Q_i(x_1, \ldots, x_m)^2) - 1, \ i = 1, 2, \ldots$$

Clearly, for any given i, this inequality is valid for all m-tuples $(x_1, \ldots, x_m)$ of nonnegative integers exactly in case the equation

$$Q_i(x_1, \ldots, x_m) = 0$$

possesses no solution in nonnegative integers. Therefore, by Theorem 1, there is no algorithm of deciding, given i, whether or not the inequality

$$0 \leq 2(Q_i(x_1, \ldots, x_m)^2) - 1$$

holds for all m-tuples $(x_1, \ldots, x_m)$ of nonnegative integers. We now move all negative terms from the right side to the left side. This gives rise to an inequality, equivalent to the original one,

$$R_i(x_1, \ldots, x_m) \leq S_i(x_1, \ldots, x_m),$$

where R_i and S_i are polynomials with positive integer coefficients.

Given i, our original inequality

$$0 \leq 2(Q_i(x_1, \ldots, x_m)^2) - 1$$

is valid for exactly those m-tuples $(x_1, \ldots, x_m)$ of nonnegative integers for which the inequality

$$1 \leq 2(Q_i(x_1, \ldots, x_m)^2) - 1$$

is valid. This means that, when comparing the polynomials R_i and S_i, it makes no difference if we take $R_i + 1$ instead of R_i. Consequently, we have obtained the following result.

Theorem 2. *There is no algorithm of deciding, given a positive integer i, whether or not the inequality*

$$R_i(x_1, \ldots, x_m) + 1 \leq S_i(x_1, \ldots, x_m)$$

holds for all m-tuples $(x_1, \ldots, x_m)$ of nonnegative integers. Here R_i and S_i are effectively constructible polynomials with positive integer coefficients, over a fixed set of variables $\{x_1, \ldots, x_m\}$.

In the sequel we will associate the polynomials R_i with specific compositions of regular operations, whereas the polynomials S_i will constitute the proposed state complexities. At this stage some further observations concerning the polynomials R_i are in order.

The polynomials R_i and S_i result from the original polynomial Q by giving an integer value to one of the variables. The polynomial Q itself is a (finite) sum of terms of the form

$$x_0^{j_0} x_1^{j_1} \cdots x_m^{j_m},\ j_\nu \geq 0,\ 0 \leq \nu \leq m,$$

provided with integer coefficients. This means that each R_i is a (finite) sum of terms of the form

$$x_1^{j_1} \cdots x_m^{j_m},\ j_\nu \geq 0,\ 1 \leq \nu \leq m,$$

provided with positive integer coefficients depending on i. Thus, we have a fixed set, independent of i, of terms of this form. The choice of i affects only the multiplicity of each term, i.e., it tells how many times each term appears in the polynomial R_i.

These observations will be important in the sequel. Upper bounds can be given both for the number of variables m, as well as for the degree of the polynomials R_i. These bounds are relevant if one wants to estimate the alphabet size in our main results. The polynomial Q can be assumed to be of degree 4, [3], so each R_i will be of degree at most 8. However, there is also a trade-off between the estimates of the degree and of m. A good combined estimate, [3], is $(5, 5)$: in Q we can assume that both m and the degree are 5. This gives the upper bound 10 for the degree of each R_i, and the upper bound 5 for the number m of variables. These values will be used in some estimations below.

3 State Complexity. Auxiliary Results from Automata Theory

We assume that the reader is familiar with the basics of finite automata theory. (For instance, [4], Vol. 1, Chapter 2 can be consulted.)

We consider in this paper only *finite deterministic automata*. We assume that the automata are *complete*, that is, for every input letter and state, the transition function determines a unique next state. An automaton has a specific initial state and a specific set of final states. We make the *convention* that, whenever we use the term *automaton*, we mean a finite deterministic automaton of the kind described above.

A state of an automaton is called a *sink* if no sequence of transitions leads from it to a final state. (Sinks are often also referred to as *garbage states*.)

In this paper we are mainly concerned with *compositions* of regular languages. By

$$C^n(L_1, \ldots, L_n),\ n \geq 1,$$

we mean a composition of regular languages $L_i,\ 1 \leq i \leq n$, using regularity-preserving operations. For instance,

$$((L_1 \cup L_2)^r \cap L_3) \cup ((\sim L_3) \cap L_1 L_2)$$

would be such a composition. There is no need for a formal definition of a general composition because we need only two regularity-preserving operations for our

undecidability result: intersection and *marked catenation* $L_1 \sharp L_2 \sharp \cdots \sharp L_n$, where $\sharp$ is a letter not in the alphabets of the languages $L_1, \ldots, L_n$, $n \geq 2$. We now give a formal definition in terms of these two operations.

Definition 1. *A* $(\cap, \sharp)$-composition *over the set* $\{L_1, \ldots, L_n\}$, $n \geq 2$, *of language variables is an expression*

$$\beta_1 \sharp \beta_2 \sharp \cdots \sharp \beta_r, \ r \geq 2,$$

where each β_i *is of the form*

$$\beta_i = K_1 \cap \cdots \cap K_{j(i)}, \ j(i) \geq 1,$$

such that the K*'s are different ones among the language variables* L_ν, $1 \leq \nu \leq n$.

For instance,

$$(L_1 \cap L_3) \sharp (L_1 \cap L_3) \sharp (L_2) \sharp (L_1 \cap L_2 \cap L_3)$$

is a $(\cap, \sharp)$-composition over the set $\{L_1, L_2, L_3\}$. Observe that the same term (here $(L_1 \cap L_3)$) may occur several times. Observe also that Definition 1 gives only a very simple way of nesting the two operations. However, it is the one needed in the sequel.

It is well known that, for a regular language L, there is a unique minimal automaton accepting L. The number of states in this automaton is referred to as the *state complexity* of L.

The situation is more involved for compositions of variable regular languages. We will now give a general definition of *state complexity functions*. The definition is given for arbitrary compositions, although we actually need it only for $(\cap, \sharp)$-compositions. The functions we are considering will be *polynomials with integer coefficients*. This is sufficient for our undecidability result because we will then consider only $(\cap, \sharp)$-compositions. (For general compositions also exponentiation, for instance, is needed.) In the usual state complexity considerations, each variable of the function corresponds to a unique language. We allow also the more general case, where several languages are associated with the same variable.

Definition 2. *Consider a polynomial* $P(x_1, \ldots, x_m)$, $m \geq 1$, *with integer coefficients, a composition* $C^n(L_1, \ldots, L_n)$, $n \geq m$, *as well as a surjective mapping* φ *of the index set* $\{1, \ldots, n\}$ *onto the index set* $\{1, \ldots, m\}$ *. Then the polynomial* $P(x_1, \ldots, x_m)$ *is a* state complexity function *of the composition* $C^n(L_1, \ldots, L_n)$ *if the following condition is satisfied. Let* $(x_1, \ldots, x_m)$ *be an arbitrary* m*-tuple of nonnegative integers. Whenever* $1 \leq i \leq m$ *and each* L_j, $j \in \varphi^{-1}(i)$, *is a regular language with state complexity* x_i*, then the composition* $C^n(L_1, \ldots, L_n)$ *is accepted by an automaton with at most* $P(x_1, \ldots, x_m)$ *states.*

Note that when we say that a polynomial $P(x_1, \ldots, x_m)$ is a state complexity function of the composition $C^n(L_1, \ldots, L_n)$, this means that the value of $P(x_1, \ldots, x_m)$ gives an *upper bound* for the state complexity of the language $C^n(L_1, \ldots, L_n)$ when each variable x_i is assigned the state complexity of the

languages L_j such that $\varphi(j) = i$. Here it is restricted that all languages L_j, $j \in \varphi^{-1}(i)$ have the same state complexity.

In the undecidability proof presented in the following section, we will associate a specific $(\cap, \sharp)$-composition with each of the polynomials R_i constructed in the preceding section. The end of this section will consist of material preparatory for this purpose. We begin with a result concerning intersections. It is similar to a construction in [9].

Theorem 3. *Assume that L_i, $1 \leq i \leq r$, are regular languages with the state complexity σ_i. Then the state complexity of the regular language $L_1 \cap \ldots \cap L_r$ is at most the product $\sigma = \sigma_1 \cdots \sigma_r$. Moreover, for any r-tuple $(\sigma_1, \ldots, \sigma_r)$ of nonnegative integers, it is possible to construct regular languages $K_i, 1 \leq i \leq r$, with the state complexity σ_i such that the intersection of the languages K_i has exactly the state complexity $\sigma = \sigma_1 \cdots \sigma_r$, and none of the minimal automata for K_i, $1 \leq i \leq r$, has a sink.*

Proof. The first claim is proved by considering an automaton whose states are r-tuples whose components simulate the state transitions of the automata for the languages L_i. The languages K_i are over the alphabet $\{a_1, \ldots, a_r\}$. For each i, $1 \leq i \leq r$. The language K_i consists of all words w such that the number of occurrences of the letter a_i in w is divisible by σ_i. It is then easy to see that the state complexity of K_i is σ_i and that the minimal automaton has no sink, as well as that the product $\sigma = \sigma_1 \cdots \sigma_r$ is the state complexity of the intersection of all languages K_i. □

The following result will take care of the sums appearing in the polynomials R_i. Since all coefficients are nonnegative, we can write R_i as a sum of products of variables, each product in each R_i coming from a specific finite set of products, determined by the original Q.

Theorem 4. *Assume that L_i are regular languages (maybe over different alphabets) with state complexities σ_i, $1 \leq i \leq r$, $r \geq 2$. Assume, further, that for each i, $1 \leq i \leq r$, the minimal automaton A_i for L_i has no sinks. Then the marked catenation*

$$L_1 \sharp L_2 \sharp \cdots \sharp L_r$$

is accepted by an automaton A with

$$\sum_{i=1}^{r} \sigma_i + 1 = \sigma$$

states but by no automaton with fewer than σ states. The alphabet of A consists of the union of the alphabets of L_i and of $\sharp$. The initial state of A_1 is the initial state of A, and the final states of A_r constitute the set of final states of A.

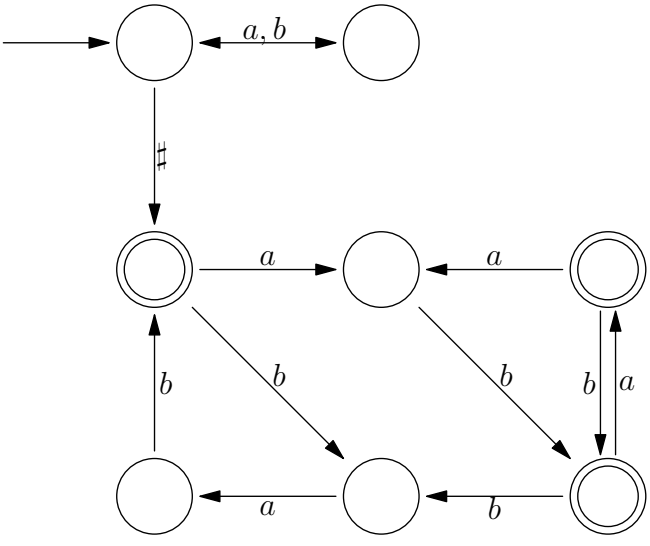
Proof. An automaton A accepting the marked catenation is obtaining by joining the automata A_i, $1 \leq i \leq r$, in the following way. From each final state of A_i, $1 \leq i \leq r-1$, introduce a transition labeled by $\sharp$ to the initial state of

A_{i+1}. From all other states of A_i, $1 \leq i \leq r-1$, as well as from all states of A_r, introduce a transition labeled by $\sharp$ to an additional sink state. It is clear that A accepts the marked catenation and has σ states. On the other hand, no automaton with fewer states can accept the marked catenation. Each word has to have exactly $r-1$ occurrences of $\sharp$. States in two different automata A_i cannot be combined because this would result into too many occurrences of the letter $\sharp$. □

If some of the automata A_i possess a sink, then the various sinks can be combined, and the total number σ can be reduced accordingly. For instance, consider the languages

$$L_1 = ((a+b)^2)^*, \ L_2 = (ab + bab + aba)^*$$

of state complexities 2 and 7, respectively. The automaton accepting L_2 has a sink. Consequently, the marked catenation $L_1 \sharp L_2$ is accepted by an automaton with only 9 states. Final states are marked by double circles in the picture. The sink and the transitions leading to it are omitted from the picture.



4 Undecidability

We will now construct a sequence of polynomials P_i with integer coefficients, as well as a sequence of $(\cap, \sharp)$-compositions C_i, for $i = 1, 2, \ldots$, such that there is no algorithm of deciding whether or not P_i is a state complexity function of C_i. The polynomials will be the polynomials S_i already constructed above in Section 2. (Observe that they have *nonnegative* integer coefficients.)

The $(\cap, \sharp)$-compositions C_i are obtained from the polynomials R_i by the following association procedure.

The set of variables of the polynomials R_i is $\{x_1, \ldots, x_m\}$. For each j, $1 \leq j \leq m$, there is a number M_j such that every exponent of x_j in every R_i is at

most M_j. Thus, M_j is independent of the index i of the polynomial R_i. This important fact follows because, as explained at the end of Section 2, a change of the index i in R_i affects only the multiplicities of the summands in R_i, not the summands themselves.

Consider now language variables

$$L_j^\nu,\ 1 \leq j \leq m,\ 1 \leq \nu \leq M_j.$$

The variables $L_j^\nu,\ 1 \leq \nu \leq M_j$ correspond to x_j in the sense of the mapping φ in Definition 2. (We have used upper indices to avoid complications in the notation.)

With each summand

$$x_1^{\nu_1} \cdots x_m^{\nu_m}$$

in R_i, we associate the intersection

$$L_1^1 \cap \ldots \cap L_1^{\nu_1} \cap \ldots \cap L_m^1 \cap \ldots \cap L_m^{\nu_m}.$$

The $(\cap, \sharp)$-composition C_i associated with R_i is now defined to be the marked catenation of all these intersections.

As a simple example, assume that

$$R_i(x_1, x_2, x_3) = x_1 x_2^2 + 3x_2^3 x_3 + x_1 x_2 x_3.$$

Then

$$C_i = (L_1^1 \cap L_2^1 \cap L_2^2)\sharp(L_2^1 \cap L_2^2 \cap L_2^3 \cap L_3^1)$$
$$\sharp(L_2^1 \cap L_2^2 \cap L_2^3 \cap L_3^1)\sharp(L_2^1 \cap L_2^2 \cap L_2^3 \cap L_3^1)\sharp(L_1^1 \cap L_2^1 \cap L_3^1).$$

The following result is now an immediate consequence of Theorems 3 and 4.

Theorem 5. *For each $i = 1, 2, \ldots,$ the polynomial $R_i(x_1, \ldots, x_m)+1$ is a state complexity function of the $(\cap, \sharp)$-composition C_i.*

Introduce now the alphabet Σ consisting of the letters

$$a_j^\nu,\ 1 \leq j \leq m,\ 1 \leq \nu \leq M_j.$$

Consider an arbitrary m-tuple $(x_1, \ldots, x_m)$ of nonnegative integers. Let

$$L_j^\nu,\ 1 \leq j \leq m,\ 1 \leq \nu \leq M_j$$

be the language over Σ consisting of all words w such that the number of occurrences of the letter a_j^ν in w is divisible by x_j. Finally, let D_i be the regular language, resulting from C_i by substituting these particular languages for the language variables. Then, again by Theorems 3 and 4, we get the following result.

Theorem 6. *For every m-tuple $(x_1, \ldots, x_m)$ of nonnegative integers, the state complexity of the language D_i equals $R_i(x_1, \ldots, x_m) + 1$.*

Using known upper bounds for the number m of variables and the degree δ of the polynomial Q in Theorem 1, we get upper bounds for the cardinality of the alphabet Σ. Observe that $M_j \leq 2\delta$, $1 \leq j \leq m$. For instance, using the known, [3], upper bound 5 for both m and δ, we get the upper bound 50 for the cardinality of Σ.

We are now ready for our main result.

Theorem 7. *For the sequence of polynomials S_i and $(\cap, \sharp)$-compositions C_i, $i = 1, 2, \ldots$, as constructed above, it is undecidable whether or not S_i is a state complexity function of C_i.*

Proof. We know by Theorem 5 that R_i+1 is a state complexity function of C_i. On the other hand, by Theorem 6, we can construct, for every m-tuple $(x_1, \ldots, x_m)$ of nonnegative integers, a language resulting from C_i whose state complexity equals exactly $R_i(x_1, \ldots, x_m)+1$. Consequently, S_i is a state complexity function of C_i exactly in case the inequality

$$R_i(x_1, \ldots, x_m) + 1 \leq S_i(x_1, \ldots, x_m)$$

holds for all m-tuples $(x_1, \ldots, x_m)$ of nonnegative integers. But this problem is undecidable, by Theorem 2. □

5 Further Considerations and Open Problems

In conclusion we briefly mention three related topics worth of further study. The topics generalize some of the constructions above.

5.1 The Effect of the Undecidability Reduction on the Associated Operations

When we used the undecidability of *Hilbert's Tenth Problem*, we had to interpret sum and product in terms of regularity-preserving operations. Marked catenation and intersection turned out to be natural ones in this respect. Reduction to other undecidable problems will in general lead to other operations. The undecidability of the state complexity will then concern composition sequences in terms of these operations.

5.2 Borderline between Decidability and Undecidability

As an example we consider marked catenation and form arbitrarily long compositions

$$L_1 \# L_2 \# \cdots \# L_n,$$

where some of the languages L_i may be equal. We want to test polynomials $S_i(x_1, \ldots, x_m)$ as possible state complexity functions of such compositions. In our undecidability result above the state complexities of the languages L_j depended on the values for the variables x_j in a rather complicated way. On the other hand,

if we assume that the state complexity of each of the component languages equals directly the value of one of the variables, then the state complexity of the marked catenation is a linear function of the variables $x_j, 1 \leq j \leq m$, and our problem is clearly decidable.

What about cases in between? If we assume that the state complexity of each component language is of the form x_j^t, where t is a positive integer, then the problem seems decidable but we have no formal proof. If the state complexities are products of powers of two variables, we might already have an undecidable problem.

Similar considerations apply to long composition sequences of operations other than marked catenation.

5.3 Regularity-Preserving Operations Associated with Classes of Functions

Above we considered certain polynomials R_i, and expressed their values in terms of state complexities, thus obtaining *composition sequences associated with polynomials*. Instead of polynomials, we might try to associate other function classes with composition sequences similarly as above. For instance, using the "maximal blow-up" result from [6], such an association is possible for polynomials with positive integer coefficients, in terms of variables and powers of the form 2^x, where x is a variable. We omit the details of this construction.

References

1. Ésik, Z., Gao, Y., Liu, G., Yu, S.: Estimation of state complexity of combined operations. Theor. Comput. Sci. 410(35), 3272–3280 (2009)
2. Gao, Y., Salomaa, K., Yu, S.: State complexity of catenation and reversal combined with star. In: Descriptional Complexity of Formal Systems (DCFS 2006), pp. 153–164 (2006)
3. Rozenberg, G., Salomaa, A.: Cornerstones of undecidability. Prentice Hall, New York (1994)
4. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 1-3. Springer, Heidelberg (1997)
5. Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations. TCS: Theoretical Computer Science 383, 140–152 (2007)
6. Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages. TCS: Theoretical Computer Science 320, 293–313 (2004)
7. Salomaa, K., Yu, S.: On the state complexity of combined operations and their estimation. IJFCS: International Journal of Foundations of Computer Science 18(4), 683–698 (2007)
8. Yu, S.: On the state complexity of combined operations. In: Ibarra, O.H., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 11–22. Springer, Heidelberg (2006)
9. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. TCS: Theoretical Computer Science 125, 315–328 (1994)

Restarting Automata with Auxiliary Symbols and Small Lookahead

Natalie Schluter

IT University of Copenhagen
Rued Langgaards Vej 7, 2300 Copenhagen S., Denmark
nael@itu.dk

Abstract. We present a study on lookahead hierarchies for restarting automata with auxiliary symbols and small lookahead. In particular, we show that there are just two different classes of languages recognised by RRWW automata, through the restriction of lookahead size. We also show that the respective (left-) monotone restarting automaton models characterise the context-free languages and that the respective right-left-monotone restarting automata characterise the linear languages both with just lookahead length 2.

1 Introduction

Restarting automata work in phases of scanning their input from the left end marker towards the right end marker, rewriting the lookahead contents with a shorter substring once per phase, and then restarting at some point before or at the right end marker. They were introduced to model the analysis by reduction grammar verification technique in the analysis of sentences in free-word order natural language. It has been shown that through various restrictions on the model, an important number of traditional and new formal language classes may be defined. Study of restarting automata has therefore also become important for both its original intent of computational linguistic application development, as well as for being an alternative machine model for investigating properties of traditional and newly distinguished formal language classes.

In his study of lookahead hierarchies, Mraz [1] showed that the expressive power of restarting automata without auxiliary symbols increases with the size of the lookahead. Schluter [3] later showed that for deterministic monotone and monotone restarting automata with auxiliary symbols, separation of rewrite and restart step is not a significant restriction on expressive power for any fixed lookahead size $k \geq 3$, and that for the deterministic model, the difference in power of the models can be overcome by approximately doubling the lookahead size, when $k \geq 3$. In both studies, it was remarked that lookahead hierarchies collapse for (left-)mon-RWW and (left-)mon-RRWW automata to $k = 3$. This paper presents a study on lookahead hierarchies for $k < 3$ of restarting automata with auxiliary symbols. In doing so, we also establish lookahead hierarchies for the most general model of restarting automata, for any k. In particular, we show

A.-H. Dediu, S. Inenaga, and C. Martín-Vide (Eds.): LATA 2011, LNCS 6638, pp. 499–510, 2011.

that there are only two different classes of languages recognised by RRWW automata, through restrictions on lookahead size.

We also partially improve a result from [3] and [1], by showing that the respective monotone and left-monotone restarting automaton models characterise the context-free languages with only lookahead size 2. And, we establish a corresponding result for the characterisation of the linear languages by the respective right-left-monotone restarting automata with lookahead size 2.

Following the definition of restarting automata and presentation of some useful properties in Section 2, we present our main results in Section 3.

Some notation. We refer to the ith symbol of a string x as $x[i]$, and its substring from the ith to jth symbols as $x[i, j]$. When we want to make the length of a string v such that $|v| = k$ explicit, we may refer to v as $v[1, k]$.

For $i, j \in \mathbb{N}$, with $i < j$, $[i, j]$ alone denotes the set $\{i, \ldots, j\}$. If $i = 1$, we say $[j] := [1, j]$.

If S is a set of symbols, then by S^i we denote the set of strings of length $i \in \mathbb{N}$ with symbols from S. Also $\lambda := S^0$ is the empty string.

Finally, REG, LIN and CFL denote the classes of regular, linear, and context-free languages respectively.

2 Preliminaries

A restarting automaton or RRWW-automaton, $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, is a nondeterministic machine model with a finite control unit and a lookahead (or read/write) window of size k (including the symbol under its scanning head, which is the first symbol of the lookahead contents) that works on a list of symbols delimited by end markers (or sentinels) $(\{¢, \$\})$, where ¢ is the left sentinel and \$ is the right sentinel. Σ is the input alphabet and $\Gamma \supseteq \Sigma$ the work tape alphabet. The symbols $\Gamma - \Sigma$ are called *auxiliary symbols*. Q is the finite set of states and $q_0 \in Q$ is the initial state. M's transition relation, δ, describes four types of transition steps (or instructions), where u is the contents of the lookahead.

(1) A *move-right step* is of the form $q' \in \delta(q, u)$, where $q, q' \in Q$. This means that M advances one tape square to the right and enters state q' upon reading u.
(2) A *rewrite step* is of the form $(q', \texttt{REWRITE}(v)) \in \delta(q, u)$, where $q, q' \in Q$, and v is such that $|v| < |u|$ $(u, v \in \Gamma^*)$. This means that M replaces its window contents u with v, advances to the tape square directly to the right of v, and enters state q'. In this rewrite instruction, we will refer to u as the *redex* and v as the *reduct*.
(3) A *restart step* is of the form $\texttt{RESTART} \in \delta(q, u)$, where $q \in Q$, in which M moves its read/write window to the beginning of the input and enters the initial state.
(4) An *accept step* is of the form $\texttt{ACCEPT} \in \delta(q, u)$, in which M halts and accepts. (This may also be viewed as the accept state.)

If $\delta(q,u) = \emptyset$, in which case we say that δ is *undefined*, M halts and rejects; we could exclude this possibility through the use of a model with both accept and reject states, in which case all possibilities for δ are defined. If $|\delta(q,u)| \leq 1$ for all q, u, then the restarting automaton is *deterministic*.

A *configuration* of M is uqv, where $u \in \{\lambda\} \cup \{\mathrm{¢}\} \cdot \Gamma^*$ is the contents of the worktape from the left sentinel to the position of the head, $q \in Q$ is the current state and $v \in \{\mathrm{¢}, \lambda\} \cdot \Gamma^* \cdot \{\$, \lambda\}$ is the contents of the worktape from the current first symbol under the scanning head to the right sentinel, and uv is the current contents of the worktape. The head scans the first k symbols of v (or all of v when $|v| \leq k$). A *restarting configuration*, for a word $w \in \Gamma^*$, is of the form $q_0 \mathrm{¢} w \$$. If $w \in \Sigma^*$, $q_0 \mathrm{¢} w \$$ is an *initial configuration*. An *accepting configuration* is a configuration with an accepting state.

A *computation* of M for an input word $w \in \Sigma^*$ is a sequence of configurations starting with an initial configuration, where two consecutive configurations are in the relation $\vdash_M$ induced by a finite set of instructions of one of the above mentioned types. The transitive closure of $\vdash_M$ is denoted $\vdash_M^*$. A *phase* of a computation begins with a restarting configuration and (exclusively) either (1) ends with the next encountered restarting configuration, in which case it includes exactly one rewrite step and is called a *cycle*, or (2) halts, in which case it includes at most one rewrite step and is called a *tail phase*. We refer to segments of a computation within a single phase before (resp. after) a rewrite as *left* (resp. *right*) *computation*.

An input word w is *accepted* or *recognised by* M if there is a computation which starts on the initial configuration and finishes in an accepting configuration. Also, we define $\mathcal{L}(M)$ as the language recognised by M.

Consider a cycle C and say the configuration from which M carries out a rewrite step is uqv in C; we define to the *right distance* of C as $D_r(C) := |v|$ and the left distance as $D_l(C) := |u|$. Let $\mathcal{C} = C_1, C_2, \ldots, C_n$ be a sequence of cycles of a restarting automaton M that, together with possibly a (final) tail phase, are M's computation on some input. If $D_r(C_i) \geq D_r(C_{i+1})$ for all $i \in [n-1]$, we say that $\mathcal{C}$ is *right-monotone* or simply *monotone*. Similarly, if $D_l(C_i) \geq D_l(C_{i+1})$ for all $i \in [n-1]$, we say that $\mathcal{C}$ is *left-monotone*. If $\mathcal{C}$ is both right- and left-monotone, then we say that $\mathcal{C}$ is *right-left-monotone*. If all the sequences of cycles corresponding to computations of a restarting automaton M are monotone (respectively left-monotone, right-left-monotone) then we say that M is *monotone* (respectively *left-monotone*, *right-left-monotone*). We denote the class of monotone RRWW-automata (respectively *left-* or *right-left-*RRWW automata), *mon*-RRWW ($left$-*mon*-RRWW or *right-left-mon*-RRWW).

Through restrictions on the restarting automaton model, we obtain many types of restarting automata. For instance, RRW-automata are RRWW-automata with no auxiliary symbols ($\Gamma = \Sigma$). An RR-automaton is an RRW-automaton with rewrite instructions that can only delete symbols. An RWW-automaton is an RRWW-automaton, which restarts immediately after any rewrite instruction, and an RW-automaton is an RRW-automaton that restarts immediately after any rewrite instruction. Finally, an R-automaton is an RR-automaton that

restarts after any rewrite instruction. All notions of monotonicity and determinism and corresponding notation extend to these more restrictive versions in the obvious way.

An X automaton, $X \in \{R, RR, RW, RWW, RRW, RRWW\}$, with lookahead size k, will be denoted by $X(k)$. For example, an RRWW(k) automaton is an RRWW automaton with lookahead size k.

2.1 Four Useful Properties

This section presents four basic lemmata used in the proofs of the main results in Section 3, without proof.[1]

The correctness preserving property is a fundamental property of restarting automata.

Proposition 1 (Correctness Preserving Property [2]). *Let M be a restarting automaton, and u, v be arbitrary input words from Σ^*. If $u \in \mathcal{L}(M)$ and $u \vdash_M^* v$ is an initial segment of an accepting computation of M, then $v \in \mathcal{L}(M)$.*

It will be useful to simplify the computations of the restarting automata that we discuss (without reducing their power). The next three lemmata serve this purpose.

A nondeterministic restarting automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is in *RR-semidet-form* if (1) halting (and restarting for automata with separate re- write and restart steps) occurs only when the right sentinel is under the lookahead, and (2) move-right steps are deterministic. The following lemma shows that non-deterministic restarting automata with lookahead length k can be assumed w.l.o.g. to be (1) in RR-semidet-form and (2) making move-right steps based only on the first symbol under the lookahead.[2]

Lemma 2. *For any X-Y automaton, $M_1 = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where $X \in \{$(right-left-, left-)mon-,$\lambda\}$ and $Y \in \{R, RR, RW, RRW, RWW, RRWW\}$, there is X-Y automaton, $M_2 = (Q', \Sigma, \Gamma, ¢, \$, q_0', k, \delta')$, such that*

1. *M_2 is in RR-semidet form,*
2. *M_2 makes move-right steps based on the couple $(u[1], q)$, where $u[1]$ is the first symbol under the lookahead and q is M_2's current state,*

and $\mathcal{L}(M_1) = \mathcal{L}(M_2)$.

If a restarting automaton M only rewrites when the contents of its lookahead is full, we say that M has *fixed rewrite size*.

Lemma 3. *For any X-Y automaton, M_1, where $X \in \{$(right-left-, left-)mon-,$\lambda\}$ and $Y \in \{R, RR, RW, RRW, RWW, RRWW\}$, there exists an X-Y automaton, M_2, that has fixed rewrite size, such that $\mathcal{L}(M_1) = \mathcal{L}(M_2)$.*

[1] Proofs of Lemmata 2, 3, and 4 may be found in the full version of this paper [4].

[2] Here, the decision whether or not to move-right remains non-deterministic; however, the decision of which move-right step to carry out becomes deterministic.

Lemma 4. *For any X-Y automaton, M_1, where $X \in \{$(right-left-, left-)mon-,$\lambda\}$ and $Y \in \{RWW, RRWW\}$, with lookahead size k, there exists an X-Y automaton, M_2, with lookahead size k, that reduces its input by only one symbol per cycle, and is such that $\mathcal{L}(M_1) = \mathcal{L}(M_2)$.*

For the remainder of this paper, we will assume w.l.o.g. that all discussed nondeterministic restarting automata with auxiliary symbols (1) are in RR-semi-det form, (2) carry out move-right steps based on the current state and the first symbol under the lookahead, (3) have fixed rewrite size, and (4) reduce their input by only one symbol per cycle.

3 Main Results

For restarting automata with auxiliary symbols and lookahead of size 1, Mráz (2001) showed that the separation of rewrite and restart step results in an increase in power for these automata. In fact, it is simple to slightly improve this result for monotonicity.[3]

Proposition 5 ([1]). *For $X \in \{$(right-left-, left-)mon-,$\lambda\}$,*

$$REG = \mathcal{L}(X\text{-}RWW(1)) \subsetneq \mathcal{L}(\textit{right-left-mon-}RRWW(1)).$$

We can also distinguish the classes of languages recognised by (monotone-) RWW (RRWW) automata with lookahead lengths 1 and 2.

Proposition 6. *For all $X \in \{$(right-left, left-)mon, $\lambda\}$, $Y \in \{RWW, RRWW\}$,*

$$\mathcal{L}(X\text{-}Y(2)) - \mathcal{L}(X\text{-}Y(1)) \neq \emptyset.$$

It turns out that further separation of language classes for RRWW is not possible. This is the main result of this paper: Theorem 7 and Corollary 12.

Theorem 7. *For $k \geq 2$ and $X \in \{$(right-left-, left-)mon, $\lambda\}$, we have*

$$\mathcal{L}(X\text{-}RRWW(k)) = \mathcal{L}(X\text{-}RRWW(k+1)).$$

Proof. Assume $M_1 = (Q_1, \Sigma, \Gamma_1, ¢, \$, q_0, k+1, \delta_1)$ is an RRWW(k+1) automaton. We construct $M_2 = (Q_2, \Sigma, \Gamma_2, ¢, \$, q_0, k, \delta_2)$ an RRWW(k) automaton to simulate M_1, such that $\mathcal{L}(M_1) = \mathcal{L}(M_2)$.

For this construction, the nondeterminacy of M_2 is essential. M_2's lookahead is one symbol shorter than M_1's. So, M_2 will simulate M_1's rewrites by guessing the contents of the tape square, τ_R, following the last symbol of its lookahead, contained in tape square τ_L. It will verify this guess within up to one step (of the same cycle), using a compound state holding this information, leaving behind in the compound symbol τ_L, how M_2 should read the guessed contents of τ_R in subsequent cycles; we'll call this instruction I. If there is a rewrite starting in

[3] The proofs of Propositions 5 and 6 may be found in the full version of this paper [4].

τ_R in a subsequent cycle, C_i, then M_2 will record in τ_R that it should ignore I in all cycles after C_i, using a Matching Lemma (Lemma 11) concerning the "interaction" of information in τ_L and τ_R. Note that this simulation could not work for $k = 1$, because then M_1 can only delete.

We now give the formal proof of the Theorem.

M_2's Work Tape. Let $\Theta_{t,\mathcal{C}} = \pi_{i_{-1}}\pi_{i_0}\pi_{i_1}\pi_{i_2}\cdots\pi_{i_{n-m}}\pi_{i_{n-m+1}}\pi_{i_{n-m+3}}$ denote M_2's work tape at time t in cycle C_m ($m \geq 1$) of computation $\mathcal{C}$ on an initial input of length n, where each π_{i_j} is a tape square boundary, for $j \in \{-1, 0\} \cup [n-m+3]$. Further, with respect to $\Theta_{t,\mathcal{C}}$, we let $\tau_R(\pi_{i_j}, t)$ denote the contents of tape square to the right of π_{i_j} at time t (if it exists) and $\tau_L(\pi_{i_j}, t)$ the contents of the tape square to the left of π_{i_j} at time t (if it exists). So, we always have, for example, $\tau_R(\pi_{-1}, t) = ¢ = \tau_L(\pi_0, t)$. We call a tape square boundary *internal* if it is between two tape squares. With each cycle, one tape square and boundary are destroyed and for this proof, we say that the second tape square involved in the redex and its boundary to the left are destroyed in the rewrite of the cycle.

Verification Information and Rewrite Set Notation. By *verification information*, `VerInf`, we will just mean some member of the set of M_1's rewrites, or the special blank symbol, $\mathbf{B} \notin \Gamma_2$, and we will denote the set of verification information as

$$\Pi := \{(q, u[1, k+1], v[1, k], q') \mid (q', \texttt{REWRITE}(v[1, k])) \in \delta_1(q, u[1, k+1])\} \cup \{\mathbf{B}\}.$$

We'll also refer to $\Pi_1 := \Pi - \{\mathbf{B}\}$ as the set of M_1's rewrites. For $\rho = (q, u[1, k+1], v[1, k], q') \in \Pi_1$, we denote to the components of ρ as follows:

$$\texttt{redex}(\rho) := u, \quad \texttt{reduct}(\rho) := v, \quad \texttt{from\_state}(\rho) : q, \text{ and } \texttt{to\_state}(\rho) := q'.$$

So, for example, $\texttt{reduct}(\rho)[k+1] = u[k+1]$ and $\texttt{redex}(\rho)[k] = v[k]$. Finally, we denote by Π_2, the set of M_2's rewrites (which will be defined shortly):

$$\Pi_2 := \{(q, x[1, k], y[1, k-1], q') \mid (q', \texttt{REWRITE}(y[1, k-1])) \in \delta_2(q, x[1, k])\}.$$

M_2's Tape Alphabet. M_2 has tape alphabet $\Gamma_2 := \Gamma_1 \cup \Delta$, where

$$\Delta := \{(x, \texttt{VerInf}, c_1, c_2) \mid x \in \Gamma_1, \texttt{VerInf} \in \Pi, c_1, c_2 \in \{0, 1, \texttt{neutral}\}\}.$$

The second through fourth components of the information from these compound symbols in Δ are used for verifying rewrite guesses, updating tape contents, and determining whether updating is necessary.

If $\texttt{VerInf} = \mathbf{B}$, we say that `VerInf` *is blank*; we refer to the set of compound symbols with blank verification information as Δ_B. Also, we refer to the set of compound symbols with the last component, c_2, not equal to `neutral` as Δ_{01}.

M_2 uses compound symbols as either the last and possibly also the first symbol of a reduct. The information `VerInf` is used for verifying rewrite guesses and updating tape contents; this component will be non-blank in the last symbol of a reduct. `VerInf` represents the latest simulated rewrite introducing a compound symbol in the tape square as the last symbol of the reduct.

The last two components of the 4-tuples in Δ take values that help determine when verification information is out of date; the third component gives instructions about information in the following tape square and the fourth component gives instructions about information in the preceding tape square. Their usage will be made precise in Remark 8 and in the description of M_2's rewrite and move-right instructions.

To refer to the different components of compound symbols $z = (z', \texttt{VerInf}, c_1, c_2) \in \Delta$, we introduce the notation $\texttt{comp}_i(z), i \in \{2,3,4\}$, which refers to the ith component of z. On the other hand, $\texttt{comp}_1$ is defined as a homomorphism $\texttt{comp}_1 : \Gamma_2 \cup \{¢, \$\} \to \Gamma_1 \cup \{¢, \$\}$ as follows, for $z \in \Gamma_2 \cup \{¢, \$\}$

$$\texttt{comp}_1(z) := \begin{cases} z & \text{if } z \in \Gamma_1 \cup \{¢, \$\} \\ x & \text{if } z = (x, \texttt{VerInf}, c_1, c_2) \in \Delta. \end{cases}$$

Then we extend $\texttt{comp}_1$ in the natural way to $\texttt{comp}_1 : (\Gamma_2 \cup \{¢, \$\})^* \to (\Gamma_1 \cup \{¢, \$\})^*$.

Further, we inductively define a mapping $h : (\Gamma_2 \cup \{\lambda, ¢\}) \times (\Gamma_2 \cup \{¢, \$\})^* \to (\Gamma_1 \cup \{¢, \$\})^*$ by

$$h(z', z) = \begin{cases} \texttt{comp}_1(z) & \text{if } z' \in \Gamma_1 \cup \Delta_B \cup \{¢\}, \text{ or if } z' = \lambda, \text{ or} \\ & \quad \text{if } z' \in \Delta - \Delta_B, z \in \Delta_{01}, \text{ and} \\ & \quad\quad \texttt{comp}_4(z) = \texttt{comp}_3(z') \\ \texttt{reduct}(\texttt{comp}_2(z'))[k] & \text{otherwise.} \end{cases}$$

Then we let $h(z', z\alpha) := h(z', z)h(z, \alpha)$, where z is a single symbol.

Since compound symbols may have various components in common, we will sometimes speak of components being introduced into tape squares. If at time t a tape square τ holds compound symbol z with some component $\texttt{comp}_i(z)$, but at time $t-1$, τ's contents held some symbol $z' \in \Gamma_2$ without the same component|that is, either $z' \in \Gamma_1$ or $\texttt{comp}_i(z') \neq \texttt{comp}_i(z)$|then we say that $\texttt{comp}_i(z)$ was *introduced* (into tape square τ) at time t.

M_2's State Set. For the definition of Q_2, we first define the two-by-two mutually exclusive sets Q_{21} and Q_{22} (which are also each mutually exclusive with Q_1).

$$\begin{aligned} Q_{21} := \{(q, \texttt{VerInf}, c, d) \mid\ & q \in Q_1 - \{\texttt{ACCEPT}, \texttt{REJECT}\}, \quad \texttt{VerInf} \in \Pi, \\ & c \in \{0, 1, \texttt{neutral}\}, d \in \{\texttt{verify}, \texttt{ignore}, \texttt{neutral}\}\} \\ Q_{22} := \{q_{u[1,k]} \mid\ & q \in Q_1, u[1,k] \in (\Gamma_1 \cup \{¢\})^k \text{ and} \\ & \delta_1(q, u[1,k]\$) \in \{\texttt{ACCEPT, REJECT}\}\} \end{aligned}$$

M_2 has the state set $Q_2 := Q_1 \cup Q_{21} \cup Q_{22}$, where Q_{22} is the set of all possible contexts leading to an accept state for M_1, used on exactly the accept step in M_2's computations. The compound states (from Q_{21}) are only used to "pick up" information from compound symbols.

To refer the different components of compound symbols $q = (q', \texttt{VerInf}, c, d) \in Q_{21}$, we introduce the notation $\texttt{COMP}_i(q), i \in \{2,3,4\}$, which refers to the ith

component of q. We further define the homomorphism $\texttt{COMP}_1(q) : Q_2 \rightarrow Q_1$ as follows, for $q \in Q_2$.

$$\texttt{COMP}_1(q) := \begin{cases} q & \text{if } q \in Q_1 \\ p & \text{if } q = p_{u[1,k]} \in Q_{22}, \text{ or if } q = (p, \texttt{VerInf}, c, d) \in Q_{21}. \end{cases}$$

The presentation of the proof is somewhat eased by first presenting some guiding properties for M_2 that the definition of rewrite and move-right steps will have to obey; this is the purpose of Remark 8 (some comments on Remark 8 follow). After this, we will prove some facts about M_2 based on these properties and use these results in the remainder of our definition of M_2 that follows.

Remark 8. M_2 will be defined according to the six following invariants:

(I1) M_2's rewrites will be of the form $(p, \texttt{REWRITE}(y[1, k-1])) \in \delta_2(q, x[1, k])$ where:
  (a) The last symbol of the reduct, $y[k-1]$, is from $\Delta - \Delta_B$ and is such that $\texttt{comp}_2(y[k-1]) \in \Pi_1$ is the rewrite of M_1 simulated.
  (b) The first symbol of the reduct, $y[1]$, is from $\Delta_{01} \cup \Gamma_1$.
  (c) All remaining symbols of the reduct, $y[i], i \in \{2, \ldots, k-2\}$ are from Γ_1.

(I2) M_2 will only write a symbol from Δ_{01} if in a compound state. In particular, if M_2 is in compound state q and writes symbol $y \in \Delta_{01}$, then $\texttt{comp}_2(y) = \texttt{COMP}_2(q)$ and $\texttt{comp}_4(y) = \texttt{COMP}_3(q)$.

(I3) M_2 will always enter a compound state after carrying out a rewrite step. In fact, if M_2 is in compound state q after writing compound symbol $y[k-1] \in \Delta - \Delta_B$, then $\texttt{COMP}_2(q) = comp_2(y[k-1])$, $\texttt{COMP}_3(q) = \texttt{comp}_3(y[k-1])$, and $\texttt{COMP}_4(q) \in \{\texttt{verify}, \texttt{ignore}\}$.

(I4) M_2 enters a compound state after reading a compound symbol from $\Delta - \Delta_B$ as the first symbol under the lookahead. Otherwise, after a move-right step M_2 must be in a state from Q_1. In fact, if M_2 reads symbol $z \in \Delta$, then it enters a compound state q such that $\texttt{COMP}_2(q) = \texttt{comp}_2(z)$, $\texttt{COMP}_3(q) = \texttt{comp}_3(z)$, and $\texttt{COMP}_4(q) = \texttt{neutral}$.

(I5) M_2 in compound state q with $\texttt{COMP}_4(q) \in \{\texttt{verify}, \texttt{ignore}\}$ rejects if it reads a compound symbol $z \in \Delta$ such that $\texttt{COMP}_3(q) = \texttt{COMP}_4(z)$.
Moreover, if M_2 does not reject and $\texttt{COMP}_4(q) = \texttt{verify}$, then M_2 checks that $\texttt{reduct}(\texttt{COMP}_2(q))[k+1] = \texttt{comp}_1(z)$ (M_2 verifies the symbol currently scanned).
Then M_2 (in both cases of $\texttt{COMP}_4(q)$) enters some state p such that $\texttt{COMP}_1(q) = \texttt{COMP}_1(p)$ and if $p \notin Q_1$, then $\texttt{COMP}_4(p) = \texttt{neutral}$ and $\texttt{COMP}_i(p) = \texttt{comp}_i(z)$ for $i \in \{2, 3\}$.

(I6) Let $p \in Q_2 - (\{\texttt{ACCEPT, REJECT}\} \cup Q_{22})$.
  (a) There is some left computation on prefix $¢\alpha \in \Gamma_2^*$ in which M_2 reaches state p if and only if there is some left computation on prefix $h(\lambda, ¢\alpha)$ that puts M_1 in state $q = \texttt{COMP}_1(p)$.
  (b) There is some right computation on prefix[4] $z\alpha$ after which M_2 enters state p where $z \in \Gamma_2, \alpha \in \Gamma_2^*$ starting in state p' if and only if there

[4] By *prefix in a right computation* we mean the prefix of the segment of work tape contents following the rewrite.

is some right computation on prefix $h(z, \alpha)$ after which M_1 enters state $\texttt{COMP}_1(p)$ starting in state $\texttt{COMP}_1(p')$.

(I1-I3) concern rewrite steps, (I4-I5) concern move-right steps, and (I6) is the main statement that ensures this proof works (valid simulations).

(I4) ensures that M_2 can update tape contents after reading a compound symbol from Δ, but that it should not verify that the rewrite guess indicated in this information is correct ($\texttt{COMP}_4(q) = \texttt{neutral}$). In fact, this verification should have taken place directly following the rewrite (in the same cycle) as is indicated in (I3) ($\texttt{COMP}_4(q) \in \{\texttt{verify, ignore}\}$). Points (I3-I5) together indicate that M_2 can only be in a state with fourth component equal a member of $\{\texttt{verify, ignore}\}$ at most once in a cycle: verification of the rewrite guess happens during a single move-right step in the same cycle.

(I2) ensures that M_2 can detect when an update of the tape contents has been written onto the tape. (I5) permits M_2 to keep track of cycle orders, to the extent that is necessary here. (See Lemma 11.)

From Remark 8, we easily obtain the following three facts:

Lemma 9. *At no time t in M_2's computation $\mathcal{C}$ is there an interior square boundary π on M_2's work tape $\Theta_{t,\mathcal{C}}$ such that $\tau_L(\pi, t) \in \Gamma_1 \cup \Delta_B \cup \{\text{¢}\}$ and $\tau_R(\pi, t) \in \Delta_{01}$. (No symbol from $\Gamma_1 \cup \Delta_B \cup \{\text{¢}\}$ directly precedes a symbol from Δ_{01} on M_2's work tape at any time t in the computation.)*

Proof. This follows from (I1-I4).

Corollary 10. *M_2 cannot read a symbol from Δ_{01} in a state from Q_1.*

The following Matching Lemma shows that M_2 can detect the order of rewrites over consecutive tape squares.

Lemma 11 (Matching Lemma). *At time t in M_2's computation $\mathcal{C}$ let π be an interior tape square boundary on M_2's work tape $\Theta_{t,\mathcal{C}}$. Suppose $\tau_L(\pi, t) \in \Delta - \Delta_B$ and $\tau_R(\pi, t) \in \Delta_{01}$. Then there are two cycles $C_{j_1}, C_{j_2} \in \mathcal{C}$, such that*

1. *M_2 uses rewrite $\rho_i = (q_i, x_i[1, k], y_i[1, k-1], q_i')$ at time t_i in C_{j_i} ($i \in [2]$) such that C_{j_1} introduced $\texttt{comp}_1(\tau_L(\pi, t)) = \texttt{comp}_1(y_1[k-1])$, and C_{j_2} introduced $\texttt{comp}_4(\tau_R(\pi, t)) = \texttt{comp}_4(y_2[1]) \in \{0, 1\}$.*
2. *(a) $\texttt{comp}_3(\tau_L(\pi, t_1)) = \texttt{comp}_4(\tau_R(\pi, t_2))$, implies $t_1 < t_2$.*
   (b) $\texttt{comp}_3(\tau_L(\pi, t_1)) \neq \texttt{comp}_4(\tau_R(\pi, t_2))$, implies $t_1 > t_2$.

Proof. (1) follows from (I1). (2a) follows from (I2) and (I4). (2b) follows from (I3) and (I5).

In case (2a) of the Matching Lemma, M_2 should update the tape square (in memory) $\tau_R(\pi, t)$ as it reads it, and in case (2b), M_2 should ignore the instruction in $\tau_L(\pi, t)$ to update the information in $\tau_R(\pi, t)$, since it is now "out of date". We also remark that the Matching Lemma helped provide the definition of the mapping h.

We now describe the rewrite and move-right instruction for M_2 with $k > 2$. The case for $k = 2$ is easily obtained from this by merging the requirements for the first and last symbols in reducts of the case $k > 2$.

Rewrite steps of M_2. Let $\rho = (q, u[1,k+1], v[1,k], q') \in \Pi_1$. We define a set of M_2's rewrites required for simulating ρ of the form

$$\rho' = (p, x[1,k], y[1,k-1], p') \subseteq \Pi_2$$

with the following component requirements.

1. $p = q$ if $p \in Q_1$, and $p = (q, \rho'', \texttt{comp}_3(\tau_L(\pi,t)), \texttt{neutral})$, otherwise, where ρ'' has further constraints with respect to $x[1]$. (See Item (7).)
2. $p' = (q', \rho, \texttt{comp}_3(y[k-1]), d)$ where $d \in \{\texttt{verify}, \texttt{ignore}\}$ (by (I3)). In particular,
$$d = \begin{cases} \texttt{ignore} & \text{if } x[k] \in \Delta - \Delta_B, \text{ and} \\ \texttt{verify} & \text{otherwise } (x[k] \in \Gamma_1 \cup \Delta_B). \end{cases}$$
3. Any $x[2,k-1] \in \Gamma_2^{k-2}$ such that $h(x[1], x[2,k-1]) = u[2,k-1]$.
4. $y[2,k-2] = v[2,k-2]$.
5. $y[k-1] = (v[k-1], \rho, c_1, \texttt{neutral}), \quad$ with $c_1 \in \{0,1\}$, by (I1).
6. (a) $x[k] \in \Gamma_1$ such that $x[k] = u[k]$, or
   (b) any $x[k] \in \Delta_B$ such that $h(x[k-1], x[k]) = u[k]$, $\texttt{comp}_3(x[k]) = \texttt{neutral}$, and $\texttt{comp}_4(x[k]) \in \{0,1\}$ or
   (c) any $x[k] \in \Delta - \Delta_B$ such that $\texttt{reduct}(\texttt{comp}_2(x[k]))[k] = u[k+1]$, and $\texttt{comp}_3(x[k]) \in \{0,1\}$.
7. Finally for $x[1]$, $y[1]$,
   - If $p \in Q_1$, then $y[1] = v[1]$ and any $x[1] \in \Gamma_2 \cup \{\texttt{¢}\}$ such that $\texttt{comp}_1(x[1]) = u[1]$ will suffice.
   - If $p \in Q_{21}$, then $y[1] = (v[1], \mathbf{B}, \texttt{neutral}, \texttt{COMP}_3(p))$ and
     - any $x[1] \in (\Gamma_2 \cup \{\texttt{¢}\}) - \Delta_{01}$ such that $\texttt{comp}_1(x[1]) = \texttt{redex}(\texttt{COMP}_2(p))[k+1]$ and $\texttt{reduct}(\texttt{COMP}_2(p))[k] = u[1]$, or
     - any $x[1] \in \Delta_{01}$ such that
       * $\texttt{COMP}_3(p) \neq \texttt{comp}_4(x[1])$, $\texttt{comp}_1(x[1]) = \texttt{redex}(\texttt{COMP}_2(p))[k+1]$ and $\texttt{reduct}(\texttt{COMP}_2(p))[k] = u[1]$, or
       * $\texttt{COMP}_3(p) = \texttt{comp}_4(x[1])$ and $\texttt{comp}_1(x[1]) = u[1]$.

     by the Matching Lemma.

There are no other rewrites in δ_2.

Note that M_2 cannot rewrite over the right sentinel, since it always simulates M_1's rewrites using only the first k symbols and M_1 has fixed rewrite size.

Move-right steps of M_2. There are two types of move-right steps for M_2 that are not derived from M_1's move right steps, for verifying rewrite guesses. These two cases, for $\delta_2(p, x[1,k])$ are when $p \in Q_{21}$ with $\texttt{COMP}_4(p) \in \{\texttt{verify}, \texttt{ignore}\}$. According to (I5), if $x[1] \in \Delta_{01}$, then we must have $\texttt{comp}_4(x[1]) \neq \texttt{COMP}_3(p)$, so M_2 will know that the rewrite guess just made was made after $x[1]'s$ information was written onto the tape (Matching Lemma). If this is not the case, M_2 rejects. Otherwise, $x[1] \in \Gamma_2 - \Delta_{01}$ and

1. if $\texttt{COMP}_4(p) = \texttt{verify}$, then M_2, having just made a rewrite guess must now verify it; we must have $\texttt{redex}(\texttt{comp}_2(p))[k+1] = \texttt{comp}_1(x[1])$ otherwise M_2 rejects. There are no other constraints on $x[1]$. Moreover,

2. if $\texttt{COMP}_4(p) = \texttt{ignore}$, then M_2 rewrote over a previous rewrite guess and should not check anything else in this tape square.

If $x[1] \in \Gamma_1$, M_2 then moves right and into state $\texttt{COMP}_1(p)$. Otherwise M_2 moves into state

$$(\texttt{COMP}_1(p), \texttt{comp}_2(x[1]), \texttt{comp}_3(x[1]), \texttt{neutral}),$$

indicating that M_2 remains in the "same" state (with respect to M_1's state), picks up $x[1]$'s verification information (in case it must update tape contents), and its matching information (to keep track of the order of rewrites). The fourth component is always `neutral` in the compound state following any step that does not verify a rewrite step.

Otherwise, M_2's move-right steps nondeterministically simulate those of M_1 simultaneously updating tape contents because of rewrite guesses. Recall that since M_1 is in the RR-semidet-form, we only need to consider the first symbol under the lookahead for M_1's move-right steps (so, in particular, we can talk about move-right steps in δ_1 on a lookahead contents of size k instead of $k+1$).

Let

$$q' \in \delta_1(q, u[1, k+1]) \tag{1}$$

be a move-right step for M_1. Then, $q' \in \delta_2(q, u[1,k])$. In addition, M_2 has the following instructions.

If $q' = \texttt{ACCEPT}$ (so $u[k+1] = \$$), then we have, for $q_{u[1,k]} \in Q_{22}$, $q_{u[1,k]} \in \delta_2(p, x[1,k])$, and

$$\delta_2(q_{u[1,k]}, x[2,k]z) \ni \begin{cases} \texttt{ACCEPT} & \text{if } z = \$, \text{ and} \\ \texttt{REJECT} & \text{otherwise.} \end{cases}$$

for all p such that $\texttt{COMP}_1(p) = q$ and $\texttt{COMP}_4 = \texttt{neutral}$, and for all $x[1,k] \in (\Gamma_2 \cup \{\$\}) \cdot \Gamma_2^{k-1}$, $z \in \Gamma_2$. Here, M_2 first guesses that M_1 would accept and then verifies its guess. We must have $\texttt{COMP}_4 = \texttt{neutral}$, because in the step after rewriting, M_2 should only be verifying or ignoring the symbol and not halting (for a valid simulation of M_1).

If $q' = \texttt{REJECT}$, then we have simply $\texttt{REJECT} \in \delta_2(p, x[1,k])$ for all p such that $\texttt{COMP}_1(p) = q$ and for all $x[1,k] \in (\Gamma_2 \cup \{\$\}) \cdot \Gamma_2^{k-1}$, so long as $\texttt{COMP}_4(p) = \texttt{neutral}$. M_2 can guess that the M_1 would reject; if this is not the case, there is still some computation that does not reject.

By Corollary 10, the remaining cases for the simulation of (1) are where M_2 reads a compound symbol (as the first symbol under the lookahead) and/or is in a compound state.

Suppose $p \in Q_1$, then $p = q$. By Corollary 10, we must have $x[1] \in \Delta - \Delta_{01}$ and therefore $\texttt{comp}_1(x[1]) = u[1]$. Now M_2 simply picks up the information in $x[1]$ and moves right as M_1 would:

$$(q', \texttt{comp}_2(x[1]), \texttt{comp}_3(x[1]), \texttt{neutral}) \in \delta_2(p, x[1,k]). \tag{2}$$

Finally, suppose $p \in Q_{21}$; then $\texttt{COMP}_1(p) = q$. The only case left to treat is where $\texttt{COMP}_4(p) = \texttt{neutral}$.

1. If $x[1] \in (\Gamma_2 \cup \{¢\} - \Delta_{01}$, then $\mathtt{comp}_1(x[1]) = \mathtt{redex}(\mathtt{COMP}_2(p))[k+1]$ and $\mathtt{reduct}(\mathtt{COMP}_2(p))[k] = u[1]$.
2. If $x[1] \in \Delta_{01}$. Then by the Matching Lemma,
   (a) $\mathtt{COMP}_3(p) \neq \mathtt{comp}_4(x[1]), \mathtt{comp}_1(x[1]) = \mathtt{redex}(\mathtt{COMP}_2(p))[k+1]$ and $\mathtt{reduct}(\mathtt{COMP}_2(p))[k] = u[1]$, or
   (b) $\mathtt{COMP}_3(p) = \mathtt{comp}_4(x[1])$ and $\mathtt{comp}_1(x[1]) = u[1]$.

M_2 rejects for all other contexts (except where it can rewrite).

M_2's rewrite and move-right steps being determined by M_1's, it follows that $\mathcal{L}(M_1) = \mathcal{L}(M_2)$. □

As a corollary of Theorem 7, we have the following lookahead hierarchy collapsal.

Corollary 12. *For $k \geq 2$ and $X \in \{$(left-, right-left-)mon, $\lambda\}$, we have*

$$\mathcal{L}(X\text{-}RRWW) = \bigcup_{k=2}^{\infty} \mathcal{L}(X\text{-}RRWW(k)) = \mathcal{L}(X\text{-}RRWW(2))$$

Corollary 12 reduces the most important question concerning restarting automata|whether the separation of rewrite and restart steps results in an increase in power|to the same question about restarting automata with lookahead length 2: $\mathcal{L}(RWW) = \mathcal{L}(RRWW) \iff \mathcal{L}(RWW) = \mathcal{L}(RRWW(2))$. It also leads to the following improvements on results of [3].

Corollary 13. *For all $k \geq 2$ and $X \in \{$left-mon, mon$\}$, we have $\mathcal{L}(X\text{-}RRWW(k)) = CFL$.*

Corollary 14. *For all $k \geq 2$, we have $\mathcal{L}$(right-left-$RRWW(k)) = LIN$.*

4 Concluding Remarks

We showed that the restriction on lookahead length is not as important a restriction for restarting automata with auxiliary symbols as opposed to those without auxiliary symbols, so long as restart and rewrite steps are separated, distinguishing only two different language classes for RRWW automata. The respective question for RWW automata remains open.

Acknowledgements. We thank the anonymous reviewers for their helpful comments.

References

1. Mráz, F.: Lookahead hierarchies of restarting automata. Journal of Automata, Languages and Combinatorics 6(4), 493–506 (2001)
2. Otto, F.: Restarting automata. Recent Advances in Formal Languages and Applications 25, 269–303 (2006)
3. Schluter, N.: On lookahead hierarchies for monotone and deterministic restarting automata with auxiliary symbols (Extended abstract). In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 440–441. Springer, Heidelberg (2010)
4. Schluter, N.: Restarting automata with auxiliary symbols and small lookahead (2011), arXiv:1101.1640

Author Index