Adrian Horia Dediu
Armand Mihai Ionescu
Carlos Martín-Vide (Eds.)

Language and Automata Theory and Applications

LNCS 5457

Third International Conference, LATA 2009
Tarragona, Spain, April 2009
Proceedings

Springer

Lecture Notes in Computer Science 5457

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

Adrian Horia Dediu Armand Mihai Ionescu
Carlos Martín-Vide (Eds.)

Language and Automata Theory and Applications

Third International Conference, LATA 2009
Tarragona, Spain, April 2-8, 2009
Proceedings

Springer

Volume Editors

Adrian Horia Dediu
Research Group on Mathematical Linguistics
Universitat Rovira i Virgili
Tarragona, Spain
E-mail: adrian.dediu@urv.cat

Armand Mihai Ionescu
Research Group on Mathematical Linguistics
Universitat Rovira i Virgili
Tarragona, Spain
E-mail: armandmihai.ionescu@urv.cat

Carlos Martín-Vide
European Research Council Executive Agency
Brussels, Belgium
E-mail: carlos.martin@urv.cat

Library of Congress Control Number: Applied for

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

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

ISSN 0302-9743
ISBN-10 3-642-00981-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-00981-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12626878 06/3180 5 4 3 2 1 0

Preface

These proceedings contain all the papers that were presented at the Third International Conference on Language and Automata Theory and Applications (LATA 2009), held in Tarragona, Spain, during April 2–8, 2009.

The scope of LATA is rather broad, including: algebraic language theory; algorithms on automata and words; automata and logic; automata for system analysis and program verification; automata, concurrency and Petri nets; biomolecular nanotechnology; cellular automata; circuits and networks; combinatorics on words; computability; computational, descriptional, communication and parameterized complexity; data and image compression; decidability questions on words and languages; digital libraries; DNA and other models of bio-inspired computing; document engineering; extended automata; 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 natural language processing, artificial intelligence and artificial life; mathematical evolutionary genomics; parsing; patterns and codes; power series; quantum, chemical and optical computing; regulated rewriting; string and combinatorial issues in computational biology and bioinformatics; symbolic dynamics; symbolic neural networks; term rewriting; text algorithms; text retrieval, pattern matching and pattern recognition; transducers; trees, tree languages and tree machines; and weighted machines.

LATA 2009 received 121 submissions, many among them of good quality. Each one was reviewed by at least three Program Committee members plus, in most cases, by additional external referees. After a thorough and vivid discussion phase, the committee decided to accept 58 papers (which means an acceptance rate of 47.93%). 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 reviewers for their cooperation and Springer for the collaboration and publication.

January 2009

Adrian Horia Dediu
Armand Mihai Ionescu
Carlos Martín-Vide

Organization

LATA 2009 was hosted by the Research Group on Mathematical Linguistics (GRLMC) at Rovira i Virgili University, Tarragona, Spain.

Program Committee

Parosh Abdulla	Uppsala, Sweden
Stefania Bandini	Milan, Italy
Stephen Bloom	Hoboken, USA
John Brzozowski	Waterloo, Canada
Maxime Crochemore	London, UK
Jürgen Dassow	Magdeburg, Germany
Michael Domaratzki	Winnipeg, Canada
Henning Fernau	Trier, Germany
Rusins Freivalds	Riga, Latvia
Vesa Halava	Turku, Finland
Juraj Hromkovič	Zurich, Switzerland
Lucian Ilie	London, Canada
Kazuo Iwama	Kyoto, Japan
Aravind Joshi	Philadelphia, USA
Juhani Karhumäki	Turku, Finland
Jarkko Kari	Turku, Finland
Claude Kirchner	Bordeaux, France
Maciej Koutny	Newcastle, UK
Hans-Jörg Kreowski	Bremen, Germany
Kamala Krithivasan	Chennai, India
Martin Kutrib	Giessen, Germany
Andrzej Lingas	Lund, Sweden
Aldo de Luca	Naples, Italy
Rupak Majumdar	Los Angeles, USA
Carlos Martín-Vide (Chair)	Brussels, Belgium
Joachim Niehren	Lille, France
Antonio Restivo	Palermo, Italy
Jörg Rothe	Düsseldorf, Germany
Wojciech Rytter	Warsaw, Poland
Philippe Schnoebelen	Cachan, France
Thomas Schwentick	Dortmund, Germany
Helmut Seidl	Munich, Germany
Alan Selman	Buffalo, USA
Jeffrey Shallit	Waterloo, Canada
Frank Stephan	Singapore

External Reviewers

Jean-Paul Allouche
Carl Alphonce
Marcella Anselmo
Franz Baader
M. Sakthi Balan
Dorothea Baumeister
Paul Bell
Suna Bensch
Clara Bertolissi
Eike Best
Henrik Björklund
Hans-Joachim Böckenhauer
Mikołaj Bojańczyk
Henning Bordihn
Ahmed Bouajjani
Pascal Bouvry
Paul Brauner
Anne Brüggemann-Klein
Michelangelo Bucci
Arturo Carpi
Giuseppa Castiglione
Stephan K. Chalup
Jan Chomicki
Christine Choppy
Hubert Comon-Lundh
Christophe Costa Florêncio
Stefano Crespi Reghizzi
Erzsébet Csuhaj-Varjú
Flavio D'Alessandro
Peter Damaschke
Alessandro De Luca
Hervé Debar
Giorgio Delzanno
Stéphane Demri
Alain Denise
Jacques Désarménien
Chrysanne DiMarco
Daniel Dougherty
Jean-Philippe Dubernard
Michael Emmi
Gábor Erdélyi
Pierre Ganty
Florent Garnier
Olivier Gauwin
Thomas Gawlitza
Wouter Gelade
Pierre Genevès
Rémi Gilleron
Francesc Godoy
Guillem Godoy
Massimiliano Goldwurm
Jean Goubault-Larrecq
Hermann Gruber
Stefan Gulan
Frank Gurski
Peter Habermehl
Tero Harju
Michael A. Harrison
Pierre-Cyrille Héam
Mika Hirvensalo
Markus Holzer
Christopher Homan
Florent Jacquemard
Petr Jancar
Ryszard Janicki
Jesper Jansson
Nataša Jonoska
Helmut Jürgensen
Michael Kaminski
Aleksandr Karbyshev
Tomi Kärki
Victor Khomenko
Dennis Komm
Adrian Kosowski
Lukasz Kowalik
Richard Královic
Bohuslav Krena
Dalia Krieger
Vladislav Kubon
Aurélien Lemay
Alberto Leporati
Martin Leucker
Christos Levcopoulos
Maria Madonia
Kalpana Mahalingam
Andreas Malcher
Sabrina Mantaci
Wim Martens

Tomáš Masopust
Giancarlo Mauri
Ian McQuillan
Massimo Merro
Tommi Meskanen
Tobias Mömke
Debdeep Mukhopadhyay
Aniello Murano
Anca Muscholl
Benedek Nagy
Stefan Näher
N. S. Narayanaswamy
Gonzalo Navarro
Frank Neven
Phong Q. Nguyen
Damian Niwinski
Thomas Noll
Dirk Nowotka
Ulrik Nyman
Mitsunori Ogihara
Alexander Okhotin
Friedrich Otto
Matteo Palmonari
Elisabeth Pelz
Mia Persson
Jean-Éric Pin
Sophie Pinchinat
Wojciech Plandowski
Denis Poitrenaud
Ely Porat
Igor Potapov
Sylvia Pott
Raghavan Rama
Narad Rampersad
Silvio Ranise
Kenneth W. Regan
Andreas Reuss
Christophe Reutenauer
Magnus Roos
Giovanna Rosone
David Sabel
Martin Schwarz
Marinella Sciortino
Sebastian Seibert
David James Sherman
Nataliya Skrypnyuk
Holger Spakowski
Andreas Sprock
Monika Steinová
Martin Sulzmann
Siamak Taati
Isabelle Tellier
Pascal Tesson
John Thistle
Ralf Treinen
Bianca Truthe
Otto Urpelainen
György Vaszil
Masilamani Vedhanayagam
Kumar Neeraj Verma
Stephan Waack
Osamu Watanabe
Bruce W. Watson
James Worrell
Menno van Zaanen
Hans Zantema
Louxin Zhang
Paweł Żyliński

Organizing Committee

Mădălina Barbaiani
Gemma Bel-Enguix
Adrian Horia Dediu
Szilárd-Zsolt Fazekas
Armand Mihai Ionescu
Maria Dolores Jiménez-López
Alexander Krassovitskiy
Guangwu Liu
Carlos Martín-Vide (Chair)
Zoltán-Pál Mecsei
Robert Mercaş
Cătălin-Ionuţ Tîrnăucă
Cristina Tîrnăucă
Bianca Truthe
Sherzod Turaev
Florentina-Lilica Voicu

Table of Contents

Invited Talks

Regular Papers

Recent Developments in Algorithmic Teaching

Frank J. Balbach[1] and Thomas Zeugmann[2]

[1] Neuhofen, Germany
frank-balbach@gmx.de
[2] Division of Computer Science
Hokkaido University, Sapporo 060-0814, Japan
thomas@ist.hokudai.ac.jp

Abstract. The present paper surveys recent developments in algorithmic teaching. First, the traditional teaching dimension model is recalled.

Starting from the observation that the teaching dimension model sometimes leads to counterintuitive results, recently developed approaches are presented. Here, main emphasis is put on the following aspects derived from human teaching/learning behavior: the order in which examples are presented should matter; teaching should become harder when the memory size of the learners decreases; teaching should become easier if the learners provide feedback; and it should be possible to teach infinite concepts and/or finite and infinite concept classes.

Recent developments in the algorithmic teaching achieving (some) of these aspects are presented and compared.

1 Introduction

When preparing a lecture, a good teacher is carefully selecting *informative examples*. Additionally, a good teacher is taking into account that students do not memorize everything previously taught. And usually we make a couple of assumptions about the learners. They should neither be ignorant, lazy, nor should they be tricky. Thus, it is only natural to ask whether or not such human behavior is at least partially reflected in some algorithmic learning and/or teaching models studied so far in the literature.

Learning concepts from examples has attracted considerable attention in learning theory and machine learning. Typically, a learner does not know much about the source of these examples. Usually the learner is required to learn from all such sources, regardless of their quality. This is even true for the query learning model introduced by Angluin [1,2], since the teacher or oracle, though answering truthfully, is assumed to behave adversarially whenever possible. Therefore, it was only natural to ask whether or not one can also model scenarios in which a helpful teacher is honestly interested in the learner's success.

Perhaps the first approach was proposed by Freivalds, Kinber, and Wiehagen [3,4]. They developed a learning model in the inductive inference paradigm of identifying recursive functions in which the learner is provided with *good* examples chosen by an implicitly given teacher. Jain, Lange, and Nessel [5] adopted

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 1–18, 2009.

this model to learn recursively enumerable languages from *good* examples in the inductive inference paradigm.

The next step was to consider teaching as the natural counterpart of learning. Teaching has been modeled and investigated in various ways within algorithmic learning theory. However, the more classical models studied so far all follow one of two basically different approaches.

In the first approach, the goal is to find a teacher *and* a learner such that a given learning task can be carried out by them. Jackson and Tomkins [6] as well as Goldman and Mathias [7,8] defined models of teacher/learner pairs where teachers and learners are constructed explicitly. In all these models, some kind of adversary disturbing the teaching process is necessary to avoid collusion between the teacher and the learner. That is, when modeling teaching, a major problem consists in avoiding coding tricks. Though there is no generally accepted definition of coding tricks, it will be clear from our exposition that *no* form of coding tricks is used and thus no collusion occurs.

Angluin and Kriķis' [9,10] model prevents collusion by giving incompatible hypothesis spaces to teacher and learner. This makes simple encoding of the target impossible.

In the second approach, a teacher has to be found that teaches *all* deterministic consistent learners. Here a learner is said to be consistent if its hypothesis is correctly reflecting all examples received. This prevents collusion, since teaching happens the same way for all learners and cannot be tailored to a specific one. Goldman, Rivest, and Shapire [11] and Goldman and Kearns [12] substitute the adversarial teacher in the online learning model by a helpful one selecting good examples. They investigate how many mistakes a consistent learner can make in the worst case. In Shinohara and Miyano's [13] model the teacher produces a set of examples for the target concept such that it is the only consistent one in the concept class. The size of this set is the same as the worst case number of mistakes in the online model. This number is termed the *teaching dimension* of the target. Because of this similarity we shall from now on refer to both models as the *teaching dimension model* (abbr. TD model).

One difficulty of teaching in the TD model results from the fact that the teacher is not knowing anything about the learners besides them being consistent. In reality a teacher can benefit a lot from knowing the learners' behavior or their current hypotheses. It is therefore natural to ask how teaching can be improved if the teacher may observe the learners' hypotheses after each example. We refer to this scenario as to teaching with *feedback*.

After translating this question into the TD model, one sees that there is no gain in sample size at all. The current hypothesis of a consistent learner reveals nothing about its following hypothesis. Even if the teacher knew the hypothesis and provided a special example in response, he can only be sure that the learner's next hypothesis will be consistent. But this was already known to the teacher. So, in the TD model, feedback is useless.

There are also several other deficiencies in the teaching models studied so far. These deficiencies include that the order in which the teacher presents examples

does not matter, and that teaching infinite concepts or infinite concept classes is severely limited. Another drawback is the rather counterintuitive dependence on the memory size of the learner. If the learner's memory size is large enough to store all examples provided by the teacher, then successful teaching is possible. Otherwise, it immediately becomes impossible. Another problem is that there are concept classes which are intuitively easy to teach that have a large teaching dimension.

Therefore, our goal has been to devise teaching models that remedy the above mentioned flaws. In particular, our aim has been to develop teaching models such that the following aspects do matter.

(1) The order in which the teacher presents the information should have an influence on the performance of the teacher.
(2) Teaching should get harder when the memory size of the learners decreases, but it should not become impossible for small memory.
(3) Teaching should get easier when the learners give feedback to the teacher.
(4) Concepts that are more complex should be harder to teach.
(5) The teaching model should work for both finite and infinite concepts and/or finite and infinite concept classes.

We studied and developed several models of algorithmic teaching to overcome these flaws to a different extent (cf. [14,15,16,17,18]). Within the present paper, we shortly summarize our and the related results obtained.

The paper is organized as follows. Section 2 shortly recalls the TD model and fundamental definitions needed subsequently. Then we discuss more recent approaches. In Section 3 we summarize results concerning teaching learners that have to obey restrictions on possible mind changes. Next, we turn our attention to a randomized model of teaching (see Section 4). Finally, we shortly touch teaching dimensions for complexity based learners and for cooperative learning.

2 The Teaching Dimension Model

We start by introducing the necessary notions and definitions. Let $\mathbb{N} = \{0, 1, \ldots\}$ denote the set of all natural numbers. For any set S we write $|S|$ to denote its cardinality. Let X be any (finite) set of *instances* also called *instance space*. A *concept* c is a subset of X and a *concept class* $\mathcal{C}$ is a set of concepts over X. It is convenient to identify every concept c with its characteristic function, i.e., for all $x \in X$ we have $c(x) = 1$ if $x \in c$ and $c(x) = 0$ otherwise. We consider mainly three instance spaces: $\{0,1\}^n$ for Boolean functions, Σ^* for languages over a finite and non-empty alphabet Σ, and $X_n = \{x_1, \ldots, x_n\}$ for having any fixed instance space of cardinality n.

By $\mathcal{X} = X \times \{0,1\}$ we denote the set of *examples* over X. An example (x, b) is either *positive*, if $b = 1$, or *negative*, if $b = 0$.

A concept c is *consistent* with a set $S = \{(x_1, b_1), \ldots, (x_n, b_n)\}$ of examples iff $c(x_i) = b_i$ for all $i = 1, \ldots, n$.

In the TD model, a *learning algorithm* takes as input a set S of examples for a concept $c \in \mathcal{C}$ and computes a hypothesis h. As mentioned in the Introduction, we

have to restrict the set of admissible learners. A *consistent* and *class preserving* learning algorithm is only allowed to choose the hypotheses from the set

$$\mathcal{H}(S) = \{h \in \mathcal{C} \mid h \text{ is consistent with } S\} .$$

A *teaching set*[1] for a concept c with respect to $\mathcal{C}$ is a set S of examples such that c is the only concept in $\mathcal{C}$ consistent with S, i.e., $\mathcal{H}(S) = \{c\}$ (cf. [12,11]). The *teaching dimension* $\mathrm{TD}(c)$ is the size of the smallest teaching set for c, the teaching dimension of $\mathcal{C}$ is

$$\mathrm{TD}(\mathcal{C}) = \max\{\mathrm{TD}(c) \mid c \in \mathcal{C}\} . \tag{1}$$

Consequently, the teaching dimension of a concept c determines the number of examples needed by an optimal teacher for teaching c to all consistent and class preserving learning algorithms. So in the TD model the information theoretic complexity of teaching is reduced to a combinatorial parameter. Note that the teaching dimension has been calculated for many natural concept classes such as (monotone) monomials, monotone k-term DNFs, k-term μ-DNFs, monotone decision lists and rectangles in $\{0, 1, \ldots, n-1\}^d$ (cf. [12]); for linearly separable Boolean functions (cf. [21,22]); for threshold functions (cf. [13]); and for k-juntas and sparse GF_2 polynomials (cf. [23]).

Since the teaching dimension does depend exclusively on the concept class, it has also been compared to other combinatorial parameters studied in learning theory. These parameters comprise the query complexity in Angluin's [2] query learning model, the VC-dimension and parameters studied in the online learning model (cf. Hegedűs [24,25], Ben-David and Eiron [26], and Rivest and Yin [27]).

Despite its succinctness and elegance, the teaching dimension has also drawbacks. For seeing this, let us consider the following example. Fix any natural number $n \geq 2$ and define the concept class $\mathcal{S}_n = \{c_0, c_1, \ldots, c_n\}$ over X_n as follows: $c_0 = \emptyset$ as well as $c_i = \{x_i\}$ for all $i = 1, \ldots, n$. Then we have $\mathrm{TD}(c_i) = 1$ for all $i = 1, \ldots, n$, since the single positive example $(x_i, 1)$ is sufficient for teaching c_i. Nevertheless, $\mathrm{TD}(c_0) = n$, since there are at least two consistent hypotheses until all n negative examples have been presented to the learners. Thus, $\mathrm{TD}(\mathcal{S}_n) = n$ despite the fact that the class $\mathcal{S}_n$ seems rather simple. However, the teaching dimension is the maximum possible.

Similar effects can be observed for the class of all monomials, all 2-term DNFs, all 1-decision lists, and all Boolean functions (over $\{0,1\}^n$), since all these classes have the same teaching dimension, i.e., 2^n.

2.1 The Average Teaching Dimension

As we have shortly outlined, the teaching dimension does not always capture our intuition about the difficulty to teach concepts. One reason for the implausibility of the results sometimes obtained is due to the fact that the teaching dimension of the class is determined by the worst case teaching dimension over all concepts.

[1] Note that a teaching set is also called *key* [13], *discriminant* [19] and *witness set* [20].

Thus, all easily learnable concepts are not taken into account. So a natural remedy is to consider the *average teaching dimension* instead of the worst case teaching dimension.

Definition 1. *Let $\mathcal{C}$ be a concept class. The* average teaching dimension *of $\mathcal{C}$ is defined as* $\overline{\mathrm{TD}}(\mathcal{C}) = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \mathrm{TD}(c)$.

Looking again at the class $\mathcal{S}_n$ defined above, we directly see that

$$\overline{\mathrm{TD}}(\mathcal{S}_n) = \frac{n + n \cdot 1}{n+1} < 2 \quad \text{for all } n \geq 2$$

and thus much smaller than the (worst case) teaching dimension $\mathrm{TD}(\mathcal{S}_n) = n$.

Anthony, Brightwell and Shawe-Taylor [22] showed that the average teaching dimension for the class of linearly separable Boolean functions is $O(n^2)$ and Kuhlmann [28] proved that all classes of VC-dimension 1 have an average teaching dimension of less than 2 and that balls of radius d in $\{0,1\}^n$ have an average teaching dimension of at most $2d$.

A more general result was shown by Kushilevitz, Linial, Rabinovich, and Saks [20]. They showed an upper bound of $O(\sqrt{|\mathcal{C}|})$ for the average teaching dimension of any concept class $\mathcal{C}$. Additionally, in [20] a family of classes is defined for which the average teaching dimension is $\Omega(\sqrt{|\mathcal{C}|})$.

Naturally, determining the average teaching dimension for classes that are more complex than $\mathcal{S}_n$ is often much harder than calculating their worst case teaching dimension. However, recently progress has been made. Balbach [14] succeeded in showing that 2-term DNFs and 1-decision lists have an average teaching dimension of $O(n)$ nicely contrasting their teaching dimension which is 2^n.

Based on Balbach's [14] results, Lee, Servedio, and Wan [23] have shown that the class of DNFs with at most $s \leq 2^{\Theta(n)}$ terms has an average teaching dimension of $O(ns)$. Furthermore, they proved that the class of k-juntas has an average teaching dimension of at most $2^k + o(1)$ and that the average teaching dimension of the class of GF_2 polynomials with $s \leq (1-\varepsilon)\log_2 n$ monomials is at most $ns + 2s$.

Nevertheless, there are still points of concern when comparing the TD model and the average teaching dimension model to a scenario where we have a machine teacher and human learners. Such scenarios are of interest for *intelligent tutoring systems* (abbr. ITS), see e.g., `www.aaai.org/AITopics/html/tutor.html`.

Human learners are not necessarily consistent, they do not remember all examples, they are sensitive to the order of examples, and they usually provide feedback about their learning progress.

Clearly, the order of examples does not matter in the TD model and as mentioned in the Introduction, in the TD model feedback is useless. Learners not being consistent with all examples are excluded by the definition of the TD model. There is, however, a dependence on the memory of the learners. As long as the learners can memorize at least $\mathrm{TD}(c)$ many examples, teaching the concept c is possible. If less than $\mathrm{TD}(c)$ many examples can be memorized then teaching becomes *impossible*.

Last but not least, the applicability to infinite concepts and classes is limited. Even a rather simple class, like the class of all finite languages over a fixed alphabet Σ yields an infinite teaching dimension. Therefore, we continue with different approaches to model algorithmic teaching.

3 Teaching Learners with Restricted Mind Changes

In this section we summarize some of the results from Balbach and Zeugmann [16]. We modify the TD model by introducing a neighborhood relation over all possible hypotheses. The learners are then restricted to choose a new hypothesis from the neighborhood of their current one. This may reflect human behavior, since humans tend to modify their hypotheses instead of creating completely new ones.

We then compare basically two variants: In the first, the teacher receives the learner's hypothesis after every example taught. In the second, the teacher has no feedback available. As a matter of fact, in this new model feedback can really make a difference. Some concept classes can be taught much faster with feedback than without and some cannot be taught unless feedback is available to the teacher.

Some additional notation is necessary. Let R be a set of strings. We say that R *represents* the class $\mathcal{C}$ iff there is a function $\gamma\colon R \times X \to \{0,1\}$ such that $\mathcal{C} = \{\mathcal{C}_r \mid r \in R\}$, where $\mathcal{C}_r = \{x \mid \gamma(r,x) = 1\}$. The length of r is denoted by $|r|$ and $size(c) := \min\{|r| \mid \mathcal{C}_r = c\}$ for every $c \in \mathcal{C}$. For any set S, we denote by S^* the set of all finite tuples over S. We use the symbols $\circ$ for concatenation of tuples and $\triangle$ for the symmetric difference of two sets. Let c be a concept and let $\boldsymbol{x} \in \mathcal{X}^*$ be a list of examples, then $err(\boldsymbol{x}, c)$ is the set of all examples in $\boldsymbol{x}$ that are inconsistent with c.

For studying feedback, the learners in our model have to evolve over time. We adopt the online learning model and divide the teaching process into rounds. In each round the teacher provides an example to the learner who then computes a hypothesis from R. At the end of the round the teacher observes this hypothesis.

Thus, we describe a teacher by a function $T\colon R \times R^* \to \mathcal{X}$ receiving a concept's representation and a sequence of previously observed hypotheses as input and outputting an example.

A learner can be described by a function $L\colon \mathcal{X}^* \to R$ receiving a sequence of examples as input and outputting a hypothesis. Let $\nu \subseteq R \times R$ be a relation over R. Then L is called *restricted* to ν iff $\forall \boldsymbol{x} \in \mathcal{X}^*\ \forall z \in \mathcal{X}\ [(L(\boldsymbol{x}), L(\boldsymbol{x} \circ z)) \in \nu]$, that is ν defines the admissible mind changes of L. Now, (R, ν) is a directed graph and we define the neighborhood of $r \in R$ as $Nb(r) := \{s \in R \mid (r,s) \in \nu\} \cup \{r\}$ and denote by $dist(r,s)$ the length of a shortest path from r to s.

As we have seen, in the TD model, the learner is required to always output a consistent hypothesis. Since in the restricted model all admissible hypotheses might be inconsistent, we have to modify this demand. We require that L chooses only among the admissible hypotheses with least error with respect to the known examples. Moreover, we require a form of *conservativeness*: L may only change its hypothesis if the new one has a smaller error. This ensures that L will not

change its mind after reaching a correct hypothesis. On the other hand, we also *require* L to search for a better hypothesis if it receives an inconsistent example. Otherwise, L could stay at the initial hypothesis forever and teaching were impossible.

Definition 2. *Let R be a representation language for a concept class $\mathcal{C}$ and let $\nu \subseteq R \times R$ be a relation over R and $h_0 \in R$ a starting hypothesis. A* ν-learner *is a function $L\colon \mathcal{X}^* \to R$ with $L(\emptyset) = h_0$ and for all $\boldsymbol{x} \in \mathcal{X}^*$ and for all $z \in \mathcal{X}$:*

(1) $(L(\boldsymbol{x}), L(\boldsymbol{x} \circ z)) \in \nu$,
(2) *if $L(\boldsymbol{x}) \neq L(\boldsymbol{x} \circ z)$ then z is inconsistent with $\mathcal{C}_{L(\boldsymbol{x})}$,*
(3) *if z is inconsistent with $\mathcal{C}_{L(\boldsymbol{x})}$ then*
$L(\boldsymbol{x} \circ z) \in \arg\min_{s \in Nb(L(\boldsymbol{x}))} |err(\boldsymbol{x} \circ z, \mathcal{C}_s)|$.

We briefly remark that one can think of many plausible variants of the above definition. For instance, the learner could be allowed to change its mind on a consistent example if its hypothesis is inconsistent with an example received earlier. In this section, however, all learners follow Definition 2.

The teaching process for a concept $c = \mathcal{C}_r$ is fully described by a teacher T and a learner L together with an initial hypothesis h_0. Such a process will result in a series $(h_i)_{i \in \mathbb{N}}$ of hypotheses and a series $(z_i)_{i \in \mathbb{N}}$ of examples: $h_{i+1} = L(z_0, \ldots, z_i)$ and $z_i = T(r, (h_0, \ldots, h_i))$.

Definition 3. *Let $\mathcal{C}$ be a concept class with representation R and let $\nu \subseteq R \times R$. We call* $\mathcal{C}$ teachable to ν-learners in the limit with feedback *iff there is a teacher T such that for all representations $r \in R$ and all ν-learners L the series $(h_i)_{i \in \mathbb{N}}$ of hypotheses converges to an h with $\mathcal{C}_h = \mathcal{C}_r$.*

The teaching time of T on r *is the maximum i such that there is a ν-learner L that reaches a representation of $\mathcal{C}_r$ at round i for the first time.*

Note that an infinite teaching time does not imply unteachability of a concept. For studying the influence of feedback, we also have to define teaching without feedback. In this situation the teacher is modeled as a function $T\colon R \times \mathbb{N} \to \mathcal{X}$, where the second argument specifies the round. The series of hypotheses is then given by $h_{i+1} = L(T(r, 0), \ldots, T(r, i))$. With this notation the definition of teaching *in the limit without feedback* is literally the same as Definition 3.

In the situation with feedback the teacher can stop teaching as soon as the learner has reached the goal. If there is no feedback, the teacher may or may not know when to stop. A teacher stopping after finitely many examples and still ensuring the learning success is said to teach *finitely without feedback*. More formally we consider $T\colon R \times \mathbb{N} \to \mathcal{X} \cup \{\perp\}$ where $\perp$ means "teaching has stopped."

With feedback we do not need to distinguish teaching finitely from teaching in the limit and we shall call this kind of teaching simply *teaching with feedback*.

Definition 4. *Let $\mathcal{C}$ be a concept class with representation R and let $\nu \subseteq R \times R$. We call* $\mathcal{C}$ finitely teachable to ν-learners without feedback *iff there is a teacher T such that for all representations $r \in R$ and all ν-learners L the hypothesis h_j with $j = \min\{i \mid T(r, i) = \perp\}$ satisfies $\mathcal{C}_{h_j} = \mathcal{C}_r$.*

Setting $\nu = R \times R$ in Definition 4 gives the teacher-directed learning model having no restriction on hypothesis changes (cf. [11]). Theorem 5 justifies the use of *arbitrary* ν's for studying the impact of feedback on the teaching process.

Theorem 5. *Let $\mathcal{C}$ be a concept class with representations R and let $\nu = R \times R$. Then the following statements are equivalent:*

(1) *$\mathcal{C}$ is finitely teachable to ν-learners without feedback,*
(2) *$\mathcal{C}$ is teachable in the limit to ν-learners without feedback,*
(3) *$\mathcal{C}$ is teachable to ν-learners with feedback.*

Furthermore in all three cases the same teacher can be used to obtain minimum teaching time which for all $c \in \mathcal{C}$ equals $\mathrm{TD}(c)$ with respect to $\mathcal{C}$.

Note that Theorem 5 relies on the fact that neither the teacher nor the learners nor the function γ are required to be recursive. Adding these requirements leads to new questions which we skip here due to space constraints.

Next, we apply the new framework to the class $\mathcal{C}_{fin}$ of all finite languages over an alphabet Σ. This class cannot be taught in the TD model. By using different ν-restrictions we demonstrate various effects.

We fix any total ordering on all strings over Σ and use as representation language R the set of all comma-separated ordered lists of strings over Σ, i.e., $r = w_1, \ldots, w_m \in R$ represents the language $\{w_1, \ldots, w_m\}$. We define the allowed transitions from r to s by $(r, s) \in \nu$ iff $|\mathcal{C}_r \triangle \mathcal{C}_s| \leq 1$. The initial hypothesis is the empty string ε representing the empty concept. Now we have:

Fact 6. *$\mathcal{C}_{fin}$ is finitely teachable to ν-learners without feedback.*

Feedback can be utilized when the restriction is modified. We define $(r, s) \in \nu'$ iff $\mathcal{C}_s = \mathcal{C}_r \cup \{w_1, w_2\}$ for some $w_1, w_2 \in \Sigma^*$ or $\mathcal{C}_s = \mathcal{C}_r \setminus \{w_1\}$. In both cases, we require that the size of the hypotheses may at most double each round: $|s| \leq 2|r|$. In the special case $r = \varepsilon$ we allow every singleton concept as neighbor: $(\varepsilon, s) \in \nu$ for all s with $|\mathcal{C}_s| = 1$. For ν'-learners there is a big difference in teaching time between teaching with and without feedback.

Fact 7. *$\mathcal{C}_{fin}$ is teachable to ν'-learners with feedback such that for all $c \in \mathcal{C}$ the number of examples is $O(|c|) \leq O(size(c))$.*

If we remove the size restriction from ν' we obtain ν''.

Fact 8. *$\mathcal{C}_{fin}$ is not finitely teachable to ν''-learners without feedback, but it is finitely teachable with feedback as well as in the limit without feedback.*

Finally we define ν'''. It differs from ν'' in that a string may only be removed from the hypothesis if neither its predecessor nor its successor (with respect to the fixed ordering on Σ^*) is contained in the hypothesis.

Fact 9. *$\mathcal{C}_{fin}$ is not teachable to ν'''-learners in the limit without feedback, but it is finitely teachable with feedback.*

If we denote by $TFIN, TFB, TLIM$ the set of all $(\mathcal{C}, R, \nu, h_0)$ such that $\mathcal{C}$ is finitely teachable without feedback, with feedback or in the limit, respectively, we have just proved the following theorem.

Theorem 10. *$TFIN \subset TLIM \subset TFB$.*

The teaching times in our model can hardly be compared to the teaching dimension, since the latter depends only on $\mathcal{C}$, whereas different choices of ν can lead to different teaching times for the same $\mathcal{C}$. The problem of finding an optimal teacher (with or without feedback) for ν-learners is $\mathcal{NP}$-hard, since it is a generalization of finding an optimal teaching set, namely if $\nu = R \times R$ (cf. [13,12,21]). More precisely, we have the following theorem.

Theorem 11. *For all notions of teaching, the following problem is $\mathcal{NP}$-hard:*

Instance*: $\mathcal{C}, R, \nu$, and a concept c^* as 0-1-vector of length $|X|$.*
Question*: Can c^* be taught to ν-learners?*

For infinite instance spaces or classes (and infinite ν) the next theorem applies.

Theorem 12. *The following function is not computable:*

Input*: Algorithms computing total functions deciding $\mathcal{C}$ and ν.*
Output*: 1, if $\mathcal{C}$ can be taught to ν-learners; 0 otherwise.*

We finish this section by looking at teaching *without* feedback. A teacher T without feedback knows all learners' initial hypotheses h_0, but can quickly lose track of them during teaching. On the other hand, T can rule out neighbors r of h_0 by giving examples consistent with h_0, but inconsistent with r. If in such a way T can eliminate all but one neighbor r', he effectively forces all learners to switch to r'. By continuing in this manner, T always knows all learners' hypotheses even without feedback. If the enforced hypotheses approach the target, T will be successful. Figure 1 describes this strategy more formally.

1 $r := h_0$;
2 **while** $\mathcal{C}_r \neq c^*$ **do**:
  2.1 Find $s \in Nb(r)$, $S \subseteq \mathcal{X}$, and $z \in \mathcal{X}$ such that (1) $\mathcal{C}_r$ is consistent with S, but not with z, (2) s is the only neighbor of r consistent with $S \cup \{z\}$, and (3) $dist(s, r^*) < dist(r, r^*)$;
  2.2 Teach S in arbitrary order and then z;
  2.3 $r := s$;

Fig. 1. A simple general strategy for teaching without feedback by forcing all learners to make the same mind changes. The initial hypothesis is h_0, r^* represents the target.

The feasibility of this strategy depends on Step 2.1. If teaching does not need to be finite, the condition in Step 2 does not need to be checked. Albeit simple, the strategy works surprisingly often for natural concept classes and ν-restrictions. In the following we give some examples.

First, we consider the class of all monomials over n variables. Let $R = \{0, 1, *\}^n$ and define $(r, s) \in \nu$ iff r and s differ only in one "bit." As initial hypothesis $h_0 = *^n$ is used.

Fact 13. *Monomials are finitely teachable without feedback. The teaching time for each concept equals its teaching dimension.*

Next, we look at decision trees. Each learner starts at the tree consisting of only one negative leaf. In each round one leaf may be substituted by an internal node that has two differently labeled leaves as children. This specifies a relation ν_{DT} over all decision trees. Then, we have:

Fact 14. *The class of Boolean functions represented as decision trees can be taught without feedback to ν_{DT}-learners. The teaching time is linear in the size of the tree representation.*

Note that the teaching dimension with respect to all Boolean functions is 2^n for all concepts. As we have seen, for ν-learners based on decision trees, teaching can often be successful with much fewer examples. Finally, we have been a bit surprised to obtain:

Fact 15. *The class of monotone 1-decision lists can be taught to ν_{DL}-learners with feedback using $m + 1$ examples for a list of length m. It cannot be taught without feedback.*

In our model of teaching learners with restricted mind changes several effects regarding feedback can be observed. Feedback can be useless, helpful, or even indispensable for teaching. In addition, natural infinite concept classes can be taught in this model. However, the main drawback is that one has to define suitably a neighborhood relation for every concept class. Our next model avoids this difficulty. It will also allows us to study teaching learners with limited memory.

4 The Randomized Teaching Model

For the sake of motivation, let us consider the concept class of all Boolean functions over $\{0, 1\}^n$. To teach a concept to all consistent learning algorithms, i.e., in the TD model, the teacher must present all 2^n examples. Teaching a concept to all consistent learners that can memorize less than 2^n examples is impossible; there is always a learner with a consistent, but wrong hypothesis. So teaching gets harder, but in a rather abrupt way.

It seems that the worst case analysis style makes it impossible to investigate the influence of memory limitations or learner's feedback. A common remedy for this is to perform an average case analysis instead (cf. Subsection 2.1). In this section we look at a rather radical approach, i.e., we replace the set of learners by a single one that is intended to represent an "average learner."

We achieve this goal by substituting the set of deterministic learners by a single randomized one. Basically, such a learner picks a hypothesis at random

from all hypotheses consistent with the known examples. Teaching is successful as soon as the learner hypothesizes the target concept. For ensuring that the learner maintains this correct hypothesis, we additionally require the learner to be *conservative*, i.e., it can change its hypotheses only on examples that are inconsistent with its current hypothesis. The complexity of teaching is measured by the *expected* teaching time.

Next, we explain why this model should work. Since at every round there is a chance to reach the target, the target will eventually be reached even if, for instance, the randomized learner can only memorize few examples. The ability of the teacher to observe the learner's current hypothesis should be advantageous, since it enables the teacher to teach an inconsistent example in every round. Recall that only these examples can cause a hypothesis change. Below we show these intuitions to be valid.

Note that for randomized learners the complexity of the teaching process does not only depend on the examples, but also on the *order* in which they are given to the learner.

The randomized teaching model can be regarded as a *Markov Decision Process* (abbr. MDP). Such processes have been studied for several decades and we shall make use of some results from this theory (cf. [29,30]). An MDP is a probabilistic system whose state transitions can be influenced during the process by actions which incur costs. Let $\mathbb{R}$ denote the set of all real numbers. Formally, an MDP consists of a finite set S of states, an initial state $s_0 \in S$, a finite set A of actions, a function $cost\colon S \times A \to \mathbb{R}$, and a function $p\colon S \times A \times S \to [0,1]$; $cost(s,a)$ is the cost incurred by action a in state s; $p(s,a,s')$ is the probability for the MDP to change from state s to s' under action a.

In the *total cost infinite horizon* setting, the goal is to choose actions such that the expected total cost, when the MDP runs forever, is minimal. This makes sense only if there is a costless absorbing state $s^* \in S$. In the *finite horizon* setting the MDP is only run for finitely many rounds.

The actions chosen at each point in time are described by a *policy*. This is a function depending on the observed history of the MDP and the current state. A basic result says that there is a minimum-cost policy that is *stationary*, i.e., that depends only on the current state. A stationary policy $\pi\colon S \to A$ defines a Markov chain over S and for all $s \in S$ an expected time $H(s)$ to reach s^* from s. Such a policy is optimal iff for all $s \in S$:

$$\pi(s) \in \operatorname*{argmin}_{a \in A} \left(cost(s,a) + \sum_{s' \in S} p(s,a,s') \cdot H(s') \right) .$$

Finding optimal policies can be phrased as a linear programming problem and can thus be done in polynomial time in the representation size of the MDP.

For the following, we need a bit more notation. For numbers a, b with $a < b$ we write $[a,b]$ to denote the set $\{a, a+1, \ldots, b\}$ or $\{a, a+1, \ldots\}$ if $b = \infty$. As above, for any set S, we denote by S^* the set of all finite lists of elements from S. Furthermore, by S^m and $S^{\leq m}$ we denote the set of all lists with length m and at most length m, respectively. The operator $\circ_\mu$ concatenates a list of length at most

μ with a single element resulting in a list of length at most μ: $\langle x_1, \ldots, x_\ell \rangle \circ_\mu \langle y \rangle$ equals $\langle x_1, \ldots, x_\ell, y \rangle$ if $\ell < \mu$ and $\langle x_2, \ldots, x_\ell, y \rangle$ if $\ell = \mu$. We regard $\circ_\infty$ as the usual list concatenation. For a list $\boldsymbol{x}$ of examples, we set

$$\mathcal{C}(\boldsymbol{x}) = \{c \in \mathcal{C} \mid \boldsymbol{x} \text{ is consistent with } c\} .$$

We denote by $\mathcal{M}_n$ the concept class of monomials over $\{0,1\}^n$. We exclude the empty concept from $\mathcal{M}_n$ and can thus identify each monomial with a string from $\{0,1,*\}^n$ and vice versa. We use $\mathcal{D}_n$ to denote the set of all 2^n concepts over $[1,n]$. Thus, there are 2^n many concepts in $\mathcal{D}_n$.

Next, we define the randomized teaching model. The teaching process is divided into rounds. In each round the teacher gives the learner an example of a target concept. The learner memorizes this example and computes a new hypothesis based on its last hypothesis and the known examples.

The Learner. In a sense, consistency is a minimum requirement for a learner. We thus require our learners to be consistent with all examples they know. However, the hypothesis is chosen at *random* from all consistent ones.

The memory of our learners may be limited to $\mu \geq 1$ examples. If the memory is full and a new example arrives, the oldest example is erased. In other words, the memory works like a queue. Setting $\mu = \infty$ models unlimited memory.

The goal of teaching is making the learner to hypothesize the target *and to maintain it*. Consistency alone cannot guarantee this behavior if the memory is too small. In this case, there is more than one consistent hypothesis at every round and the learner would oscillate between them rather than maintaining a single one. To avoid this, *conservativeness* is required, i.e., the learner can change its hypothesis only when taught an example inconsistent with its current one.

To study the influence of the learners' feedback to the teacher, we distinguish between private and public output of the learner. The private output is the result of the calculation during a round (i.e., new memory content and hypothesis), the public output is that part of the private one observable by the teacher. So, if the learner gives feedback, the teacher can observe in every round the complete hypothesis computed by the learner. If the learner does not give feedback, the teacher can observe nothing. The following algorithm describes the behavior of the *μ-memory learner with/without feedback* (short: L_μ^+ / L_μ^-) during one round of the teaching process.

Input: memory $\boldsymbol{x} \in \mathcal{X}^{\leq \mu}$, hypothesis $h \in \mathcal{C}$, example $z \in \mathcal{X}$.
Private Output: memory $\boldsymbol{x}'$, hypothesis h'.
Public Output: hypothesis h' / nothing.
1 $\boldsymbol{x}' := \boldsymbol{x} \circ_\mu \langle z \rangle$;
2 **if** $z \notin \mathcal{X}(h)$ **then** pick h' uniformly at random from $\mathcal{C}(\boldsymbol{x}')$;
3 **else** $h' := h$;

For making our results dependent on $\mathcal{C}$ alone, rather than on an arbitrary initial state of the learner, we stipulate a special initial hypothesis, called *init*. We assume every example inconsistent with *init*. Thus, *init* is left after the first example and cannot be reached again. Moreover, the initial memory is empty.

The Teacher. A teacher is an algorithm taking initially a given target concept c^* as input. Then, in each round, it receives the public output of the learner (if any) and outputs an example for c^*.

Definition 16. *Let $\mathcal{C}$ be a concept class and $c^* \in \mathcal{C}$. Let L_μ^σ be a learner, where $\sigma \in \{+,-\}$, let T be a teacher and let $(h_i)_{i\in\mathbb{N}}$ be the series of random variables for the hypothesis at round i. The event* "teaching success in round t," *denoted by G_t, is defined as*

$$h_{t-1} \neq c^* \quad \wedge \quad \forall t' \geq t \colon h_{t'} = c^* \ .$$

The success probability *of T is $\Pr\left[\bigcup_{t\geq 1} G_t\right]$. A teaching process is* successful *iff the success probability equals 1. A successful teaching process is called* finite *iff there is a t' such that $\Pr\left[\bigcup_{1\leq t\leq t'} G_t\right] = 1$, otherwise it is called* infinite. *For a successful teaching process we define the* expected teaching time *as $\mathbb{E}[T, L_\mu^\sigma, c^*, \mathcal{C}] := \sum_{t\geq 1} t \cdot \Pr[G_t]$.*

Definition 17. *Let $\mathcal{C}$ be a concept class, $c^* \in \mathcal{C}$ and L_μ^σ a learner. We call c^** teachable *to L_μ^σ iff there is a successful teacher T. The* optimal teaching time for c^* *is*

$$E_\mu^\sigma(c^*) := \inf_T \mathbb{E}[T, L_\mu^\sigma, c^*, \mathcal{C}]$$

and the optimal teaching time for $\mathcal{C}$ is denoted by $E_\mu^\sigma(\mathcal{C}) := \max_{c\in\mathcal{C}} E_\mu^\sigma(c)$.

For exemplifying our model, we compute the optimal teaching times for $\mathcal{D}_n$. To the learner L_μ^+ $(1 \leq \mu \leq n)$ the teacher gives an example inconsistent with the current hypothesis in each round. For all such examples, there are $2^{n-\mu}$ hypotheses consistent with the μ examples in the learner's memory and it chooses one of them. So the probability of choosing the target concept is $2^{-(n-\mu)}$. Since in the first $\mu - 1$ rounds the memory contains less then μ examples, $E_\mu^+(\mathcal{D}_n)$ is, for constant μ, asymptotically equal to $2^{n-\mu}$. Clearly, teaching becomes faster with growing μ. Moreover the teaching speed increases continuously with μ and not abruptly as in the classical deterministic model. In particular, teaching is possible even with the smallest memory size ($\mu = 1$), although it takes very long (2^{n-1} rounds).

Teaching is more difficult when feedback is unavailable. In this situation the teacher can merely guess examples hoping that they are inconsistent with the current hypothesis. Roughly speaking, when teaching $\mathcal{D}_n$, the teacher needs two guesses on average to find such an example. Hence, the expected teaching time E_μ^- is about two times E_μ^+. Thus feedback doubles the teaching speed for $\mathcal{D}_n$.

Fact 18. *For all $\mathcal{C}$ and $\mu \in [1,\infty]$ all $c^* \in \mathcal{C}$ and $\sigma \in \{+,-\}$:*
(1) $E_\mu^+(c^*) \leq E_\mu^-(c^*)$, (2) $E_\infty^\sigma(c^*) \leq E_{\mu+1}^\sigma(c^*) \leq E_\mu^\sigma(c^*)$.
Proper inequality holds for the concepts in $\mathcal{D}_n$.

Next, we relate the TD model, i.e., the teaching dimension, to the randomized model (in terms of the expected teaching time).

Lemma 19 ([18]). *Let $\mathcal{C}$ be a concept class and let $c^* \in \mathcal{C}$ be a target. For all $\mu \in [1, \mathrm{TD}(c^*)]$,*

$$E^-_\mu(c^*) \;\geq\; E^+_\mu(c^*) \;\geq\; \frac{\mu(\mu-1)}{2\mathrm{TD}(c^*)} + \mathrm{TD}(c^*) + 1 - \mu,$$

and for all $\mu > \mathrm{TD}(c^*)$, $E^-_\mu(c^*) \;\geq\; E^+_\mu(c^*) \;\geq\; \mathrm{TD}(c^*)/2$.

Now, we take a closer look at learners with feedback. For the sake of presentation we start with learners with 1-memory. A teaching process involving L_1^+ can be modeled as an MDP with $S = \mathcal{C} \cup \{init\}$, $A = \mathcal{X}(c^*)$, $cost(h, z) = 1$ for $h \neq c^*$ and $cost(c^*, z) = 0$. Furthermore, for $h \neq c^*$, $p(h, z, h') = 1/|\mathcal{C}(z)|$ if $z \in \mathcal{X}(h') \setminus \mathcal{X}(h)$ and $p(h, z, h') = 0$ otherwise; finally $p(c^*, z, c^*) = 1$ (see [30,29]). The initial state is *init* and the state c^* is costless and absorbing. The memory is not part of the state, since the next hypothesis only depends on the newly given example which is modeled as an action.

An example $z \in \mathcal{X}(h)$ does not change the learner's state h and is therefore useless. An optimal teacher refrains from teaching such examples and thus we can derive the following criterion.

Lemma 20. *Let $\mathcal{C}$ be a concept class over X and let c^* be a target. A teacher $T\colon \mathcal{C} \cup \{init\} \to \mathcal{X}(c^*)$ with expectations $H\colon \mathcal{C} \cup \{init\} \to \mathbb{R}$ is optimal iff for all $h \in \mathcal{C} \cup \{init\}$:*

$$T(h) \in \operatorname*{argmin}_{\substack{z \in \mathcal{X}(c^*) \\ z \notin \mathcal{X}(h)}} \left(1 + \frac{1}{|\mathcal{C}(z)|} \sum_{h' \in \mathcal{C}(z)} H(h') \right) .$$

This criterion can be used to prove optimality for teaching algorithms.

We compare E_1^+ with other dimensions. The comparison of E_1^+ with the number MQ of membership queries (see Angluin [1]) is interesting because MQ and E_1^+ are both lower bounded by the teaching dimension.

Fact 21

(1) *For all $\mathcal{C}$ and $c^* \in \mathcal{C}$: $E_1^+(c^*) \geq \mathrm{TD}(c^*)$.*
(2) *There is no function of* TD *upper bounding $E_1^+(c)$.*
(3) *There is no function of E_1^+ upper bounding MQ.*
(4) *There is a concept class $\mathcal{C}$ with $E_1^+(\mathcal{C}) > MQ(\mathcal{C})$.*
(5) *For all concept classes $\mathcal{C}$, $E_1^+(\mathcal{C}) \leq 2^{MQ(\mathcal{C})}$.*

Roughly speaking, teaching L_1^+ can take arbitrarily longer than teaching in the classical model, but is still incomparable with membership query learning.

We finish this subsection by looking at learners with ∞-memory. A straightforward MDP for teaching c^* to L_∞^+ has states $S = (\mathcal{C} \cup \{init\}) \times \mathcal{X}(c^*)^{\leq |X|}$. The number of states can be reduced because two states (h, m) and (h, m') with $\mathcal{C}(m) = \mathcal{C}(m')$ are equivalent from a teacher's perspective, but in general the size of the resulting MDP will not be polynomial in the size of the matrix representation of $\mathcal{C}$. Therefore, optimal teachers cannot be computed efficiently by the known general MDP algorithms.

A similar criterion as Lemma 20 can be stated for the L^+_∞ learner, too, and used to prove optimality of algorithms. Note that there is always a teacher that needs at most $\mathrm{TD}(c^*)$ rounds by giving a minimal teaching set, hence $E^+_\infty(c^*) \leq \mathrm{TD}(c^*)$. Second, it follows from Lemma 19 that $E^+_\infty(c^*) \geq \mathrm{TD}(c^*)/2$. This means that every algorithm computing $E^+_\infty(c^*)$ also computes a factor 2 approximation of the teaching dimension.

As it has often been noted [13,21,12], the problem of computing the teaching dimension is essentially equivalent to the `SET-COVER` (or `HITTING-SET`) problem which is a difficult approximation problem. Raz and Safra [31] have shown that there is no polynomial time constant-factor approximation (unless $\mathcal{P} = \mathcal{NP}$). Moreover, Feige [32] proved that `SET-COVER` cannot be approximated better than within a logarithmic factor (unless $\mathcal{NP} \subseteq DTime(n^{\log\log n})$).

Corollary 22. *Unless $\mathcal{NP} \subseteq DTime(n^{\log\log n})$, computing E^+_∞ is $\mathcal{NP}$-hard and cannot be approximated with a factor of $(1-\epsilon)\log(|\mathcal{C}|)$ for any $\epsilon > 0$.*

However, we have:

Fact 23. *Let $\mathcal{C}$ be a concept class and $c^* \in \mathcal{C}$ a target. Then there is a successful teacher for the learner L^+_∞ halting after at most $|X|$ rounds that is also optimal.*

Note that our model of teaching a randomized learner also allows for studying teaching from *positive data* only. The interested reader is referred to [18] for details. Here we only mention the following topological characterization.

Theorem 24. *Let $\mathcal{C}$ be a concept class and $c^* \in \mathcal{C}$ a target concept. Then for all learners L^σ_μ with $\mu \in [1,\infty]$, $\sigma \in \{+,-\}$: The concept c^* is teachable from positive data iff there is no $c \in \mathcal{C}$ with $c \supset c^*$.*

Studying the teachability of randomized learners without feedback is much harder, since the problem of finding the optimal cost in an MDP whose states are not observable is much more difficult. We refer the interested reader to Balbach and Zeugmann [17] for results in this regard.

5 Further Directions

A number of variations of the teaching dimension have been studied in which the learner is assumed to act smarter than just choosing a consistent hypothesis. One such model (see Balbach [14,15]) assumes the learner picks hypotheses that are not only consistent but of minimal complexity. This model is inspired by the *Occam's razor* principle. A teacher that exploits this learning behavior can make do with fewer examples than in the original teaching dimension model. For example, 2-term DNFs and 1-decision lists can be taught with $O(n)$ examples, as opposed to 2^n examples (n being the number of variables). Note that this model presupposes a measure of complexity for all concepts in the class, such as

the length of a decision list or the number of terms in a DNF. But there might not be a unique, natural complexity measure for a given class.

Another variant devised by Balbach [15] assumes that the learners know the teaching dimensions of all concepts in the class and choose their hypotheses only from the consistent ones with a teaching dimension at least as large as the sample given by the teacher. In other words, they assume the teacher does not give more examples than necessary for the concept to be taught. This *optimal teacher teaching dimension (OTTD)* model demands more of the learners, as they need to know all the teaching dimensions, but reduces the number of examples compared to the plain teaching dimension. For example, the OTTD of monomials is linear in the number of variables, and the OTTD of 1-decision lists is bounded by $\Omega(\sqrt{n} \cdot 2^{n/2})$ and $O(n\sqrt{\log n} \cdot 2^{n/2})$.

If the learners in the OTTD model base their reasoning on the OTTD rather than the teaching dimension, one obtains yet another dimensionality measure. Intuitively, the learners assume that the teacher knows how they think and adjusts the sample he gives. Now the teacher can exploit this new behavior of the students, again reducing the number of examples needed. This gives rise to a series of decreasing dimensionality notions which, for finite concept classes, will eventually converge.

In a similar vein, Zilles, Lange, Holte, and Zinkevich [33] defined a teaching model which is based on cooperative learners. Here the learners are assumed to know all minimal teaching sets for all concepts in the class, and they always choose from the consistent hypotheses a minimal teaching set which contains all examples given so far. In other words they assume that at all times the sample given by the teacher can be extended to a minimal teaching set for the concept to be taught. This demands even more of the learners than the OTTD, as they now have to know all minimal teaching sets of all concepts. In a similar way as above, this idea can be iterated, yielding a series of dimensionality notions that converge to one called the *subset teaching dimension (STD)*. The STD is a lower bound for the iterated OTTDs. Zilles *et al.* [33] show a number of surprising properties of the STD. For example, the STD of the class of monomials is two, independent of the number of variables. A surprising property of the STD is its nonmonotonicity, that is, the STD of some classes is less than that of one of their subclasses.

Finally, Zilles *et al.* [33] devise a dimensionality notion called the *recursive teaching dimension (RTD)*, which combines the easy teachability of the monomials in the STD model with the property of monotonicity in the TD and OTTD models. However, it is not based on refined assumptions about the learners' behavior and only measures the teachability of an entire concept class, not that of individual concepts. The basic idea is to order the concepts in a class and determine the teaching dimension of each concept with respect only to the class of all following concepts. The maximum teaching dimension for any concept minimized over all possible orderings determines the RTD of the class. The RTD is a lower bound for all iterated OTTDs, but its precise relationship to the STD is unknown. The STD is conjectured to be a lower bound for the RTD.

References

1. Angluin, D.: Queries and concept learning. Machine Learning 2, 319–342 (1988)
2. Angluin, D.: Queries revisited. Theoret. Comput. Sci. 313, 175–194 (2004)
3. Freivalds, R., Kinber, E.B., Wiehagen, R.: Inductive inference from good examples. In: Jantke, K.P. (ed.) AII 1989. LNCS (LNAI), vol. 397, pp. 1–17. Springer, Heidelberg (1989)
4. Freivalds, R., Kinber, E.B., Wiehagen, R.: On the power of inductive inference from good examples. Theoret. Comput. Sci. 110, 131–144 (1993)
5. Jain, S., Lange, S., Nessel, J.: On the learnability of recursively enumerable languages from good examples. Theoret. Comput. Sci. 261, 3–29 (2001)
6. Jackson, J., Tomkins, A.: A computational model of teaching. In: Proc. 5th Annual ACM Workshop on Computational Learning Theory, pp. 319–326. ACM Press, New York (1992)
7. Goldman, S.A., Mathias, H.D.: Teaching a smarter learner. J. of Comput. Syst. Sci. 52, 255–267 (1996)
8. Mathias, H.D.: A model of interactive teaching. J. of Comput. Syst. Sci. 54, 487–501 (1997)
9. Angluin, D., Kriķis, M.: Teachers, learners and black boxes. In: Proceedings of the Tenth Annual Conference on Computational Learning Theory, pp. 285–297. ACM Press, New York (1997)
10. Angluin, D., Kriķis, M.: Learning from different teachers. Machine Learning 51, 137–163 (2003)
11. Goldman, S.A., Rivest, R.L., Schapire, R.E.: Learning binary relations and total orders. SIAM J. Comput. 22, 1006–1034 (1993)
12. Goldman, S.A., Kearns, M.J.: On the complexity of teaching. J. of Comput. Syst. Sci. 50, 20–31 (1995)
13. Shinohara, A., Miyano, S.: Teachability in computational learning. New Generation Computing 8, 337–348 (1991)
14. Balbach, F.J.: Teaching classes with high teaching dimension using few examples. In: Auer, P., Meir, R. (eds.) COLT 2005. LNCS (LNAI), vol. 3559, pp. 668–683. Springer, Heidelberg (2005)
15. Balbach, F.J.: Measuring teachability using variants of the teaching dimension. Theoret. Comput. Sci. 397, 94–113 (2008)
16. Balbach, F.J., Zeugmann, T.: Teaching learners with restricted mind changes. In: Jain, S., Simon, H.U., Tomita, E. (eds.) ALT 2005. LNCS (LNAI), vol. 3734, pp. 474–489. Springer, Heidelberg (2005)
17. Balbach, F.J., Zeugmann, T.: Teaching memoryless randomized learners without feedback. In: Balcázar, J.L., Long, P.M., Stephan, F. (eds.) ALT 2006. LNCS (LNAI), vol. 4264, pp. 93–108. Springer, Heidelberg (2006)
18. Balbach, F.J., Zeugmann, T.: Teaching randomized learners. In: Lugosi, G., Simon, H.U. (eds.) COLT 2006. LNCS (LNAI), vol. 4005, pp. 229–243. Springer, Heidelberg (2006)
19. Natarajan, B.K.: Machine Learning: A Theoretical Approach. Morgan Kaufmann, San Mateo (1991)
20. Kushilevitz, E., Linial, N., Rabinovich, Y., Saks, M.: Witness sets for families of binary vectors. Journal of Combinatorial Theory Series A 73, 376–380 (1996)
21. Anthony, M., Brightwell, G., Cohen, D., Shawe-Taylor, J.: On exact specification by examples. In: Proc. 5th Annual ACM Workshop on Computational Learning Theory, pp. 311–318. ACM Press, New York (1992)

22. Anthony, M., Brightwell, G., Shawe-Taylor, J.: On specifying Boolean functions by labelled examples. Discrete Applied Mathematics 61, 1–25 (1995)
23. Lee, H., Servedio, R.A., Wan, A.: DNF are teachable in the average case. In: Lugosi, G., Simon, H.U. (eds.) COLT 2006. LNCS, vol. 4005, pp. 214–228. Springer, Heidelberg (2006)
24. Hegedűs, T.: Combinatorial results on the complexity of teaching and learning. In: Privara, I., Ružička, P., Rovan, B. (eds.) MFCS 1994. LNCS, vol. 841, pp. 393–402. Springer, Heidelberg (1994)
25. Hegedűs, T.: Generalized teaching dimensions and the query complexity of learning. In: Proc. 8th Annual Conference on Computational Learning Theory, pp. 108–117. ACM Press, New York (1995)
26. Ben-David, S., Eiron, N.: Self-directed learning and its relation to the VC-dimension and to teacher-directed learning. Machine Learning 33, 87–104 (1998)
27. Rivest, R.L., Yin, Y.L.: Being taught can be faster than asking questions. In: Proc. 8th Annual Conference on Computational Learning Theory, pp. 144–151. ACM Press, New York (1995)
28. Kuhlmann, C.: On teaching and learning intersection-closed concept classes. In: Fischer, P., Simon, H.U. (eds.) EuroCOLT 1999. LNCS, vol. 1572, pp. 168–182. Springer, Heidelberg (1999)
29. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Chichester (1994)
30. Bertsekas, D.P.: Dynamic Programming and Optimal Control. Athena Scientific (2001)
31. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In: Proc. of the 29th ACM Symposium on Theory of Computing, pp. 475–484. ACM Press, New York (1997)
32. Feige, U.: A threshold of ln n for approximating set cover. J. of the ACM 45, 634–652 (1998)
33. Zilles, S., Lange, S., Holte, R., Zinkevich, M.: Teaching dimensions based on cooperative learning. In: 21st Annual Conference on Learning Theory - COLT 2008, Helsinki, Finland, pp. 135–146. Omnipress (2008)

Monadic Second-Order Logic for Graphs: Algorithmic and Language Theoretical Applications*

Bruno Courcelle

Université Bordeaux-1, LaBRI, CNRS
Institut Universitaire de France
351, Cours de la Libération
33405, Talence cedex, France
courcell@labri.fr

Abstract. This tutorial will present an overview of the use of Monadic Second-Order Logic to describe sets of finite graphs and graph transformations, in relation with the notions of tree-width and clique-width. It will review applications to the construction of algorithms, to Graph Theory and to the extension to graphs of Formal Language Theory concepts.

We first explain the role of Logic. A graph, either finite or infinite, can be considered as a logical structure whose domain (the ground set of the logical structure) consists of the set of vertices; a binary relation on this set represents adjacency. Graph properties can be expressed by logical formulas of different languages and classified accordingly. First-order formulas are rather weak in this respect because they can only express *local properties*, like having degree or diameter at most k for fixed k. Most properties of interest in Graph Theory can be expressed in second-order logic (this language allows quantifications on relations of arbitrary but fixed arity), but unfortunately, little can be obtained from such expressions.

Second-order formulas that only use quantifications on unary relations, i.e., on sets are *Monadic second-order formulas*. They can express many basic and useful graph properties like connectivity, k-colorability, planarity and minor inclusion, just to take a few examples. These properties are said to be *monadic second-order expressible* and the corresponding sets of graphs are called *monadic second-order definable*. Many algorithmic properties follow from such logical descriptions. In particular, every monadic second-order definable set of *finite* graphs of bounded *tree-width* has a linear time recognition algorithm ([1], [2], [3], [4], [5]).

Monadic second-order formulas are also used in Formal Language Theory to describe *languages*, i.e., sets of words or terms. A fundamental result in this field is that monadic second-order formulas and finite automata have the same expressive power. It is fundamental for the theory and practice of model-checking,

* Supported by the GRAAL project of "Agence Nationale pour la Recherche".

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 19–22, 2009.

and, in Language Theory, it helps to analyze and classify the *regular languages*, i.e., those defined by finite automata. Monadic second-order formulas are even more important for describing sets of graphs than for describing languages because there is no convenient notion of graph automaton. They can also replace finite automata for defining graph transformations, that we call *transductions*, as in Language Theory.

However, monadic second-order logic alone yields no interesting results. As mentioned above every monadic second-order graph property has, for each k, a polynomial time checking algorithm for graphs of tree-width at most k. No such algorithm can exist for arbitrary graphs, otherwise P=NP because 3-colorability is expressible by a monadic second-order formula. The checking problem for every monadic second-order graph property is nevertheless *fixed parameter tractable* (FPT) for tree-width as parameter. (See [4], [5] on this notion). Hence in order to be useful, the expression of a graph property by a monadic second-order formula must be coupled with constraints on the considered graphs like having bounded tree-width, or with other constraints that are also based on appropriate types of hierarchical decompositions. *Clique-width* is another graph parameter, based on certain graph decompositions, that yields FPT algorithms for monadic second-order expressible properties ([2], [6], [7]). Tree-width and clique-width are important for language theoretical issues as well as for the construction of polynomial algorithms.

It is convenient to formalize the *hierarchical graph decompositions* that yield the notions of tree-width and clique-width and fit with monadic second-order logic, as algebraic terms written with appropriate *graph operations* that generalize the concatenation of words. The value of such a term t is a finite graph G and t is one of its hierarchical decompositions of the considered type. The subterms of t define (roughly speaking) combinations of larger and larger subgraphs of G. This fact justifies the use of the word "hierarchical" in the above description.

Sets of finite graphs can be described as subsets of certain graph algebras, by means of notions of Universal Algebra that are already known in Language Theory. Two of these notions are those of an *equational* and of a *recognizable* set of elements of an algebra. In the monoid of words, the corresponding classes of sets are respectively those of context-free and of regular languages. The notion of an equational set generalizes to arbitrary algebras the well-known *Least-Fixed Point Characterization* of context-free languages, and that of a recognizable set generalizes the characterization of regular languages in terms of finite congruences. Many properties of equational and recognizable sets of graphs are just particular instances of results that hold in arbitrary algebras, but others are particular to the considered algebras of graphs.

Finite graphs can thus be handled in two ways. The "logical way" characterizes graphs "from inside", that is, in terms of what they are made of and contain: vertices, edges, paths, minors, subgraphs. The "algebraic way" characterizes sets of graphs in a global way : a graph is treated as an element of an algebra and related with other elements of the same algebra, that are not necessarily among its subgraphs. Two important theorems relate these two approaches. The

Recognizability Theorem says that every set of finite graphs that is monadic second-order definable is recognizable. The *Equationality Theorem* says that a set of finite graphs is equational if and only if it is the set of graphs "definable inside finite trees" by a fixed finite tuple of monadic second-order formulas : we will say : "is the image of a set of finite trees under a *monadic second-order transduction*".

It follows from the Recognizability Theorem that, for a graph G defined by a term t (relative to an appropriate graph algebra), one can check in time $O(|t|)$ whether or not G satisfies a fixed monadic second-order property. It follows also that the graphs of an equational set that satisfy a fixed monadic second-order property (like planarity) form an equational set. We call this result the *Filtering Theorem*. It generalizes the classical fact that the intersection of a context-free language with a regular one is context-free. Since the emptiness of an equational set is decidable, we get as a corollary that the *monadic second-order satisfiability problem* is decidable for every equational set L . This means that one can decide whether or not a given monadic second-order formula is satisfied by some graph in L. The Equationality Theorem entails that the family of equational sets of graphs is preserved under monadic second-order transductions. This corollary is similar to the fact that the image of a context-free language under a rational transduction is context-free. (A rational transduction can be specified by a nondeterministic finite-state transducer, and if the image of every word is a finite set, then it is also a monadic second-order transduction). The corollary follows from the fact that the class of monadic second-order transductions is closed under composition.

The Recognizability and the Equationality Theorem contribute to establishing the foundations of a sound and robust extension of the theory of formal languages intended to cover descriptions of sets of finite graphs, and in which monadic second-order logic plays a major role. From the above informal statements, this extension may seem to be straightforward. However, general graphs are intrinsically more complex than words and terms, and some results do not extend. Let us give two examples. The set of all finite graphs is not equational, whereas the set of all words on a finite alphabet is (trivially) context-free. There are uncountably many recognizable sets of graphs, and this fact forbids any characterization of these sets in terms of graph automata, that would generalize a classical characterization of regular languages. These examples reflect the fact that the sets of graph operations upon which Recognizability and Equationality are based are infinite, and that this infiniteness is unavoidable.

According to the above presentation, monadic second-order formulas expressing graph properties do not use edge set quantifications. This is due to the chosen representation of a graph by a relational structure. If we replace a graph G by its incidence (bipartite) graph $Inc(G)$, where each edge is made into a vertex and where the adjacency relation $edg_{Inc(G)}(e, v)$ expresses that a vertex v of G is an end of edge e, then monadic second-order formulas to be interpreted in $Inc(G)$ can use edge set quantifications. A graph property is MS_2-*expressible* if it is expressible by a monadic second-order formula on incidence graphs. This variant of

monadic second-order logic has strictly more expressive power than the initially defined language. The existence of a perfect matching is MS_2-expressible but not monadic second-order expressible. However, the two variants of monadic second-order logic have the same expressive power on words, on trees and on certain classes of graphs like those of planar graphs or, for each k, of graphs of degree at most k.

The Recognizability Theorem and the Equationality Theorem have thus two versions, relative to the two possible representations of graphs by relational structures and to two different graph algebras. The graph algebra corresponding to MS_2-formulas, called the *HR algebra*, is the one with graph operations that express tree-decompositions and characterize tree-width. The one corresponding to monadic second-order formulas without edge set quantifications is called the *VR algebra*, it defines clique-width. The acronyms HR and VR refer respectively to *hyperedge replacement* and *vertex replacement* because the equational sets of the HR and of the VR algebras are the sets of graphs generated by certain context-free graph grammars called respectively *hyperdege replacement* and *vertex replacement graph grammars*. (See [1] and the first two chapters of the same volume.)

These results are also interesting for Structural Graph Theory, i.e., for the study of graph decompositions, of embeddings of graphs on surfaces, of forbidden configuration characterizations, of colorings expressed as homomorphisms between graphs. Quick proofs that certain graph classes have bounded tree-width or clique-width can be obtained from the Equationality Theorem.

All these definitions and results are in a book in preparation [2].

References

1. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformations. Foundations, vol. 1, pp. 313–400. World Scientific, Singapore (1997)
2. Courcelle, B.: Graph structure and monadic second-order logic. In: Book in preparation. Cambridge University Press, Cambridge (2009), http://www.labri.fr/perso/courcell/ActSci.html
3. Courcelle, B., Mosbah, M.: Monadic second-order evaluations on tree-decomposable graphs. Theor. Comput. Sci. 109, 49–82 (1993)
4. Downey, R., Fellows, M.: Parameterized Complexity. Springer, Heidelberg (1999)
5. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
6. Courcelle, B., Makowsky, J., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. Theory Comput. Syst. 33, 125–150 (2000)
7. Makowsky, J.: Algorithmic uses of the feferman-vaught theorem. Ann. Pure Appl. Logic 126, 159–213 (2004)

Descriptional and Computational Complexity of Finite Automata

Markus Holzer and Martin Kutrib

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

Abstract. Over the last half century, a vast literature documenting the importance of deterministic, nondeterministic, and alternating finite automata as an enormously valuable concept has been developed. In the present paper, we tour a fragment of this literature. Mostly, we discuss developments relevant to finite automata related problems like, for example, (i) simulation of and by several types of finite automata, (ii) standard automata problems such as, e.g., fixed and general membership, emptiness, universality, equivalence, and related problems, and (iii) minimization and approximation. We thus come across descriptional and computational complexity issues of finite automata. We do not prove these results but we merely draw attention to the big picture and some of the main ideas involved.

1 Introduction

Nondeterministic finite automata (NFAs) were introduced in [59], where their equivalence to deterministic finite automata (DFAs) was shown. Later the concept of alternation was developed in [10], where also alternating finite automata (AFAs) were investigated, which turned out to be equivalent to DFAs, too. Many work has been done in the study of descriptional complexity of simulation of and by several types of automata and on the computational complexity of decision problems related to finite automata. The goal of this research is to obtain tight bounds on simulation results and to classify the computational complexity of problems according to the complexity classes NC^1, L, NL, P, NP, and PSPACE, or others—for basics in computational complexity theory we refer to, e.g., [33]. Our tour on the subjects listed in the abstract of finite automata related problems cover some (recent) results in the field of descriptional and computational complexity. It obviously lacks completeness and it reflects our personal view of what constitute the most interesting links to descriptional and computational complexity theory. In truth there is much more to the regular languages, DFAs, NFAs, etc., than one can summarize here. For a recent survey on finite automata we refer to [68] and [30].

Our nomenclature of finite automata is as follows: The powerset of a set Q is denoted by 2^Q and the empty word by λ. A *nondeterministic finite automaton* (NFA) is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite set of *states*, Σ

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 23–42, 2009.

is the finite set of *input symbols*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states*, and $\delta : Q \times \Sigma \to 2^Q$ is the *transition function*. A finite automaton is *deterministic* (DFA) if and only if $|\delta(q,a)| = 1$, for all states $q \in Q$ and letters $a \in \Sigma$. In this case we simply write $\delta(q,a) = p$ instead of $\delta(q,a) = \{p\}$ assuming that the transition function is a mapping $\delta : Q \times \Sigma \to Q$. So, any DFA is complete, that is, the transition function is total, whereas it may be a partial function for NFAs in the sense that the transition function of nondeterministic machines may map to the empty set. The *language accepted* by the NFA or DFA A is defined as $L(A) = \{\, w \in \Sigma^* \mid \delta(q_0,w) \cap F \neq \emptyset \,\}$, where the transition function is recursively extended to $\delta : Q \times \Sigma^* \to 2^Q$. A finite automaton is said to be *minimal* if its number of states is minimal with respect to the accepted language. Note that a sink state is counted for DFAs, since they are always complete, whereas it is not counted for NFAs, since these devices are not necessarily complete. For further details we refer to [33].

We identify the logical values *false* and *true* with 0 and 1 and write $\{0,1\}^Q$ for the set of finite functions from Q into $\{0,1\}$, and $\{0,1\}^{\{0,1\}^Q}$ for the set of Boolean formulas (functions) mapping $\{0,1\}^Q$ into $\{0,1\}$. An *alternating finite automaton* (AFA) is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0, and F are as for NFAs, and $\delta : Q \times \Sigma \to \{0,1\}^{\{0,1\}^Q}$ is the *transition function*. The transition function maps pairs of states and input symbols to Boolean formulas. Before we define the language accepted by the AFA A we have to explain how a word is accepted. As the input is read (from left to right), the automaton "builds" a propositional formula, starting with the formula q_0, and on reading an input a, replaces every $q \in Q$ in the current formula by $\delta(q,a)$. The input is *accepted* if and only if the constructed formula on reading the whole input evaluates to 1 on substituting 1 for q, if $q \in F$, and 0 otherwise. This substitution defines a mapping from Q into $\{0,1\}$ which is called the *characteristic vector* f_A of A. Then the *language accepted* by A is defined as $L(A) = \{\, w \in \Sigma^* \mid w \text{ is accepted by } A \,\}$. Two automata are *equivalent* if and only if they accept the same language. For further details we refer to [10] and [33].

2 Descriptional Complexity of Finite Automata Simulations

Since regular languages have many representations in the world of finite automata, it is natural to investigate the succinctness of their representation by different types of automata in order to optimize the space requirements. Here we measure the costs of representations in terms of the states of a minimal automaton accepting a language. More precisely, the *simulation problem* is defined as follows:

– Given two classes of finite automata C_1 and C_2, how many states are sufficient and necessary in the worst case to simulate n-state automata from C_1 by automata from C_2?

In particular, we are interested in simulations between DFAs, NFAs, and AFAs.

It is well known that to any NFA one can always construct an equivalent DFA [59]. This so-called *powerset construction*, where each state of the DFA is associated with a subset of NFA states, turned out to be optimal, in general. That is, the bound on the number of states necessary for the construction is tight in the sense that for an arbitrary n there is always some n-state NFA which cannot be simulated by any DFA with strictly less than 2^n states [56,58]. So, NFAs can offer exponential saving in the number of states compared with DFAs. This gives rise to the following theorem.

Theorem 1 (NFA by DFA Simulation). *Let $n \geq 1$ and A be an n-state NFA. Then 2^n states are sufficient and necessary in the worst case for a DFA to accept $L(A)$.*

The situation becomes more involved when AFAs come into play. Alternating finite automata as we have defined them have been developed in [10]. At the same period in [9] the so-called *Boolean automata* were introduced. Note, that several authors use the notation "alternating finite automata" but rely on the definition of Boolean automata. Though it turned out that both types are almost identical, there are differences with respect to the initial configurations. While for AFAs the computation starts with the fixed propositional formula q_0, a Boolean automaton starts with an arbitrary propositional formula. Clearly, this does not increase their computational capacities. However, it might make a difference of one state from a descriptional complexity point of view when simulating a Boolean automaton by an AFA. It is an open problem whether or not the additional state is really necessary, that is, whether the bound of $n + 1$ is tight.

Next we turn to the simulation of AFAs by NFAs and DFAs. The tight bound of 2^{2^n} states for the deterministic simulation of n-state AFAs has already been shown in the famous fundamental papers [10] for AFAs and [9,50] for Boolean automata.

Theorem 2 (AFA by DFA Simulation). *Let $n \geq 1$ and A be an n-state AFA or Boolean automaton. Then 2^{2^n} states are sufficient and necessary in the worst case for a DFA to accept $L(A)$.*

The original proofs of the upper bound rely on the fact that an AFA or a Boolean automaton can enter only finitely many internal situations, which are given by Boolean functions depending on n Boolean variables associated with the n states. The number of 2^{2^n} such functions determines the upper bound.

In [17] the constructions of simulating NNFAs are presented which implies the same upper bound. Basically, an NNFA is an NFA with multiple entry states, where initially one is nondeterministically chosen. The basic idea is that the NNFA simulates the AFA or Boolean automaton A by guessing the sequence of functions of the form $\{0,1\}^Q$ that appear during the evaluation of the propositional formula computed by the A in reverse order. Since there are 2^n such functions we obtain the upper bound stated in Theorem 3. Moreover, since the powerset constructions works also fine for the NNFA by DFA simulation, the

presented construction also reveals the upper bound for the AFA simulation by DFAs already stated in Theorem 2.

It is known that any NNFA can be simulated by an NFA having one more state. The additional state is used as new sole initial state which is appropriately connected to the successors of the old initial states. On the other hand, in general this state is needed. Nevertheless, it is an open problem whether there are languages accepted by n-state AFAs or Boolean automata such that any equivalent NFA has at least $2^n + 1$ states. In [17] it is conjectured that this bound presented in the following theorem is tight.

Theorem 3 (AFA by NNFA and NFA Simulation). *Let $n \geq 1$ and A be an n-state AFA or Boolean automaton. Then 2^n states are sufficient and necessary in the worst case for an NNFA to accept $L(A)$. Moreover, $2^n + 1$ states are sufficient for an NFA to accept $L(A)$, and for every $n \geq 1$ there is an n-state AFA or Boolean automaton A such that any NFA accepting $L(A)$ has at least 2^n states.*

The matching lower bound of Theorem 2 is shown in [10] for AFAs by witness languages in a long proof. Before we come back to this point for Boolean automata, we turn to an interesting aspect of AFAs and Boolean automata. One can observe that the construction of the simulating NNFA is backward deterministic [10]. So, the reversal of a language accepted by an n-state AFA or Boolean automaton is accepted by a *not necessarily complete* 2^n-state DFA which in turn can be simulated by a (2^n+1)-state *complete* DFA. This result has significantly be strengthened in [50], where it is shown that the reversal of *every* n-state DFA language is accepted by a Boolean automaton with $\lceil \log_2(n) \rceil$ states. With other words, *with restriction to reversals* of regular languages a Boolean automaton can always save exponentially many states compared with a DFA. The next theorem summarizes these results.

Theorem 4 (Reversed AFA by DFA Simulation). *Let $n \geq 1$ and A be an n-state AFA or Boolean automaton. Then $2^n + 1$ states are sufficient and necessary in the worst case for a DFA to accept the reversal of $L(A)$. If the minimal DFA accepting the reversal of $L(A)$ does not have a rejecting sink state, then 2^n states are sufficient. Moreover, the reversal of every language accepted by an n-state DFA is accepted by a Boolean automaton with $\lceil \log_2(n) \rceil$ states.*

The theorem lefts open whether the reversal of *every* n-state DFA language is also accepted by some *AFA* with $\lceil \log_2(n) \rceil$ states. However, we know that $\lceil \log_2(n) \rceil + 1$ states are sufficient for this purpose.

Now we are prepared to argue for the matching lower bound of Theorem 2 for Boolean automata in a simple way. It is well known that for any $m \geq 1$ there is an m-state DFA A such that any DFA accepting the reversal of $L(A)$ has 2^m states [50]. Setting $m = 2^n$ we obtain a 2^n-state DFA language $L(A)$ whose reversal is accepted by a Boolean automaton with n states by Theorem 4. On the other hand, the reversal of $L(A)$ takes at least 2^{2^n} states to be accepted deterministically.

Next we argue that the upper bound of Theorem 3 cannot be improved in general. To this end, let A be an n-state AFA or Boolean automaton such that any equivalent DFA has 2^{2^n} states. Let m be the minimal number of states for an equivalent NNFA. Since the NNFA can be simulated by a DFA with at most 2^m states, we conclude $2^m \geq 2^{2^n}$, that is, the NNFA has at least $m \geq 2^n$ states.

We now direct our attention to the question whether alternation can always help to represent a regular language succinctly. It is well known that nondeterminism cannot help for all languages. So, how about the worst case of the language representation by alternating finite automata? The situation seems to be more sophisticated. Theorem 4 says that for reversals of n-state DFA languages we can *always* achieve an exponential saving of states. Interestingly, this potential gets lost when we consider the n-state DFA languages itself (instead of their reversals). The next theorem and its corollary are from [51].

Theorem 5. *For every $n \geq 1$ there exists a minimal DFA A with n states such that any AFA or Boolean automaton accepting $L(A)$ has at least n states.*

The DFAs $A_n = (\{q_0, q_1, \ldots, q_{n-1}\}, \{a, b\}, \delta, q_1, F)$ witness the theorem for $n \geq 2$, where $F = \{\, q_i \mid 0 \leq i \leq n-1 \text{ and } i \text{ even} \,\}$ and the transition function given by

$$\delta(q_i, a) = q_{(i+1) \bmod n} \quad \text{and} \quad \delta(q_i, b) = \begin{cases} q_i & \text{for } 0 \leq i \leq n-3 \\ q_{n-1} & \text{for } i \in \{n-2, n-1\}. \end{cases}$$

Each DFA A_n has the property that any DFA A'_n accepting the reversal of $L(A)$ has at least 2^n states. Moreover, A_n and A'_n both are minimal, complete and do not have a rejecting sink state [50]. Assume that $L(A)$ is accepted by some AFA or Boolean automaton with $m < n$ states. Then the reversal of $L(A)$ would be accepted by some DFA having at most 2^m states by Theorem 4. This is a contradiction since $2^m < 2^n$.

Up to now we dealt with simulations whose costs optimality is witnessed by regular languages which may be built over alphabets with two or more letters. For the particular case of *unary regular languages*, that is, languages over a single letter alphabet, the situation turned out to be significantly different. The problem of evaluating the costs of unary automata simulations was raised in [61], and has led to emphasize some relevant differences with the general case. So, we next turn to draw a part of that picture, which is complemented by the sophisticated studies in [55] which reveal tight bounds also for many other types of finite automata and, in addition, is a valuable source for further references. For state complexity issues of unary finite automata Landau's function $F(n) = \max\{\, \operatorname{lcm}(x_1, \ldots, x_k) \mid x_1, \ldots, x_k \geq 1 \text{ and } x_1 + \cdots + x_k = n \,\}$ plays a crucial role. Here, lcm denotes the least common multiple. Since F depends on the irregular distribution of the prime numbers, we cannot expect to express $F(n)$ explicitly by n. The following asymptotic tight bound on the unary NFA by DFA simulation was presented in [13,14].

Theorem 6 (Unary NFA by DFA Simulation). *Let $n \geq 1$ and A be an n-state NFA accepting a unary language. Then $e^{\Theta(\sqrt{n \cdot \ln n})}$ states are sufficient and necessary in the worst case for a DFA to accept $L(A)$.*

In general, the deterministic simulation of AFAs may cost a double exponential number of states. The unary case is cheaper. Since every unary language coincides trivially with its reversal, the upper bound of the following theorem is immediately derived from Theorem 4. Interestingly, to some extend for unary languages it does not matter in general whether we simulate an AFA deterministically or nondeterministically. The tight bounds differ at most by one state. The upper bound of this claim follows since any DFA is also an NFA and NFAs are not necessarily complete. The lower bounds can be seen by considering the single word language $L_n = \{a^{2^n-1}\}$. For each $n \geq 1$, the language L_n is accepted by some minimal $(2^n + 1)$-state DFA as well as by some minimal 2^n-state NFA.

Theorem 7 (Unary AFA by DFA and NFA Simulation). *Let $n \geq 1$ and A be an n-state AFA accepting a unary language. Then $2^n + 1$ states are sufficient and necessary in the worst case for a DFA to accept $L(A)$. If the minimal DFA does not have a rejecting sink state, then 2^n states are sufficient. Moreover, 2^n states are sufficient and necessary in the worst case for an NFA to accept $L(A)$.*

Theorem 5 revealed that alternation cannot help to reduce the number of states of DFAs or NFAs in all cases. The same is true for nondeterministic simulations of DFAs in general and in the unary case. However, for unary languages alternation does help. By Theorem 7 we know already that any AFA simulating an n-state DFA accepting a unary language has not less than $\lceil \log_2(n) \rceil - 1$ states. Once more the unary single word languages L_n are witnesses that this saving can be achieved. This gives rise to the next theorem.

Theorem 8 (Unary DFA by AFA Simulation). *Let $n \geq 1$ and A be an n-state DFA accepting a unary language. Then $\lceil \log_2(n) \rceil - 1$ states are necessary for an AFA to accept $L(A)$. Moreover, there exists a minimal DFA A with n states accepting a unary language such that any minimal AFA accepting $L(A)$ has exactly $\lceil \log_2(n) \rceil - 1$ states.*

Finally, we derive the *always possible* savings for unary NFA by AFA simulations as follows. Given some n-state NFA accepting a unary language, by Theorem 6 we obtain an equivalent DFA that has at most $e^{\Theta(\sqrt{n \cdot \ln n})} = 2^{\Theta(\sqrt{n \cdot \ln n})}$ states. Now Theorem 4 in combination with says essentially that there is an equivalent AFA with $\Theta(\sqrt{n \cdot \ln n})$ states. In order to see that these savings are optimal in general, consider a unary n-state NFA such that any equivalent DFA must have $e^{\Theta(\sqrt{n \cdot \ln n})}$ states. Since the bound of Theorem 6 is tight such automata exist. Clearly, any equivalent AFA has at least $\Theta(\sqrt{n \cdot \ln n})$ states. Otherwise there would be an equivalent DFA with less than $e^{\Theta(\sqrt{n \cdot \ln n})}$ states by Theorem 7.

Theorem 9 (Unary NFA by AFA Simulation). *Let $n \geq 1$ and A be a minimal n-state NFA accepting a unary language. Then $\Theta(\sqrt{n \cdot \ln n})$ states are*

sufficient for an AFA to accept $L(A)$. Moreover, there exists an n-state NFA accepting a unary language such that any equivalent AFA requires $\Theta(\sqrt{n \cdot \ln n})$ states. If A is a unary n-state NFA such that any equivalent DFA has at least $e^{\Theta(\sqrt{n \cdot \ln n})}$ states, then any AFA accepting $L(A)$ has at least $\Theta(\sqrt{n \cdot \ln n})$ states.

Concerning structural properties in [17] it is shown that negations in the Boolean functions defining an AFA can be avoided at the cost of increasing the number of states by factor of two. For the role played by the number of accepting states the following is known. While the family of languages accepted by DFAs with k accepting states is strictly contained in the family of languages accepted by DFAs with $k+1$ accepting states, for $k \geq 0$, it is known that for NFAs two states are always sufficient. The situation for AFAs is in contrast to the situation for DFAs but parallels the situation for NFAs. More precisely, in [17] it has been shown that for every n-state AFA A accepting a λ-free regular language one can construct an equivalent n-state AFA A' without accepting state. If $L(A)$ contains the empty word, then A' has one sole accepting state that coincides with the start state.

3 Computational Complexity of Some Decision Problems for Finite Automata

We recall what is known from the computational complexity point of view on some standard problems for regular languages. The problems considered in this section are all decidable, as most problems for finite automata, and they will be grouped as mentioned in the abstract.

3.1 The Fixed and General Membership Problem

Our tour on problems for regular languages is started with the definition of the fixed and general membership problem: The former problem is device independent by definition and is commonly referred to in the literature as the *fixed membership problem* for regular languages:

– *Fix* a finite automaton A. For a given word w, does the word w belong to the language $L(A)$, i.e., is $w \in L(A)$?

A natural generalization is the *general membership problem*, which is defined as follows:

– *Given* a finite automaton A and a word w, i.e., a suitable coding[1] $\langle A, w \rangle$, does the word w belong to the language $L(A)$, i.e., is $w \in L(A)$?

[1] A coding function $\langle \cdot \rangle$ maps a finite automaton A and a string w to a word $\langle A, w \rangle$ over a fixed alphabet Σ. We do not go into the details of $\langle \cdot \rangle$, but assume it fulfills certain standard properties; for instance, that the coding of the input alphabet symbols as well as the coding of the states is of logarithmic length.

Obviously, the fixed membership problem for regular languages reduces to the general membership problem for any suitable class of automata, that describes the family of regular languages. On the other hand, the complexity of the general membership problem may depend on the given language descriptor. For instance, it is easy to see that the general membership problem for DFAs is in L, and in fact, complete for L under weak reductions. The problem is NL-complete for NFAs [47], and becomes P-complete for AFAs as shown in [45]. These completeness results even hold for finite automata accepting languages over a singleton, i.e., unary languages. We summarize these results in the following theorem:

Theorem 10 (General Membership). *The general membership problem for DFAs is* L*-complete with respect to constant depth reducibilities. Moreover, the problem is* NL*-complete for NFAs and becomes* P*-complete for AFAs. The results remain valid for automata accepting unary languages.*

For the fixed membership problem there is much more to tell, since there is a deep and nice connection between this problem and circuit complexity theory. There exist several characterizations, in terms of formal logic, semigroup theory, and operations on languages, of the regular languages in the circuit complexity classes AC^0, ACC^0, and NC^1—see, e.g., [6,7,53]. Here AC^0 (NC^1, respectively) is the class of languages accepted by uniform circuit families of constant (logarithmic, respectively) depth, polynomial size, with AND- and OR-gates of unbounded (bounded, respectively) fan-in and ACC^0 is the class of languages accepted by AC^0 circuits with additional MODULO-gates. Hence $\mathsf{AC}^0 \subseteq \mathsf{ACC}^0 \subseteq \mathsf{NC}^1$ and note that AC^0 is distinct from NC^1 by [3] and [18]. It is conjectured that ACC^0 is a proper subset of NC^1, too.

First, observe that by a divide and conquer approach it is easy to see that regular languages in general belong to NC^1. On the other hand, the NC^1 lower bound (under weak reductions such as, e.g., constant depth reducibilities) was established in the landmark paper of Barrington [5], where it was shown that bounded width polynomial size programs over finite monoids recognize exactly the languages in NC^1. To this end, it was shown how to simulate the AND-, OR-, and NOT-gates of a NC^1-circuit with programs over the symmetric group S_5 on five elements. Programs over monoids are a straightforward generalization of the concept of of recognizability. Here language $L \subseteq \Sigma^*$ is *recognizable* if and only if there exists a finite monoid M, a morphism $\varphi : \Sigma^* \to M$, and a subset $N \subseteq M$ such that $L = \varphi^{-1}(N)$. In other words an input word $w = a_1 a_2 \dots a_n$ is translated *via* the morphism φ to the word $\varphi(a_1)\varphi(a_2)\dots\varphi(a_n)$ of monoid elements, that is evaluated in the monoid to yield a value whose N membership is tested. A *program over a monoid* M, for short M*-program*, takes an input word $a_1 a_2 \dots a_n$ of length n and transforms it into a word $\sigma = \sigma_1\sigma_2\dots\sigma_m$, for some m, over monoid elements by querying the input positions in arbitrary order and transforming the read letter into a monoid element, which is multiplied in the monoid to yield the value of the program. More formally, an M-program for input of length n, is a sequence of instructions $P_n = \langle i_1, \varphi_1\rangle\langle i_2, \varphi_2\rangle \dots \langle i_m, \varphi_m\rangle$, where for each j with $1 \leq j \leq m$ we have $1 \leq i_j \leq n$ and $\varphi_j : \Sigma \to M$, and an accepting subset $N \subseteq M$. On input $w = a_1 a_2 \dots a_n$, the program produces the

word $\varphi(w) = \varphi_1(a_{i_1})\varphi_2(a_{i_2})\dots\varphi_m(a_{i_m})$ of monoid elements, that is multiplied out in M and whose N membership determines whether the input is recognized or not, i.e., the word w is recognized if and only if $\varphi(w)$ is in N. The words recognized by the M-program P_n define a language $L \subseteq \Sigma^n$ in the obvious way. The result in [5] reads as follows:

Theorem 11 (Fixed Membership). *The fixed membership problem for regular languages is* NC^1*-complete with respect to constant depth reducibilities.*

The strong algebraic background on this result has triggered further studies on M-programs over monoids satisfying certain restrictions. For instance, one of the best investigated restriction is aperiodicity. Here a monoid is *aperiodic* if and only if all elements x from the monoid satisfy $x^t = x^{t+1}$, for some $t \geq 0$. It is well known that a language L has a aperiodic syntactic monoid if and only if language L is star-free, i.e., it can be obtained from the elementary languages $\{a\}$, for $a \in \Sigma$, by applying Boolean operations and finitely many concatenations, where complementation is with respect to Σ^*. These languages are exhaustively studied in, e.g., [54] and [60]. We mention in passing that first it was shown in [63] that the aperiodicity problem (given a finite automaton with input alphabet Σ, does it accept an aperiodic or star-free language?) for DFAs is coNP-hard and belongs to PSPACE. Later this result was improved to PSPACE-completeness [11]. In fact, using some algebraic background developed in [67] on the parameterization of aperiodic and solvable monoids one can show the following result [7].

Theorem 12 (Fixed Membership). *(1) The fixed membership problem for regular languages* recognized *by aperiodic monoids belongs to* AC^0*. (2) The fixed membership problem for regular languages* recognized *by solvable monoids belongs to* ACC^0*.*

A closer look reveals that one can obtain even more, namely a tight connection between the parameterization of AC^0 in terms of circuit depth k and a parameterization of aperiodic monoids, namely Brzozowski's dot-depth hierarchy [15,16]. Instead of using a divide-and-conquer approach as for regular languages in general, the inductive definition of dot-depth k monoids or languages allows a straightforward decomposition of language membership that gives a one-to-one correspondence between dot-depth and circuit depth. In [7] it was shown that the fixed membership problem for regular languages *recognized* by dot-depth k monoids is solvable in AC^0 by a family of depth k circuits.

3.2 Emptiness, Universality, Equivalence, and Related Problems

In this subsection we consider non-emptiness, universality, equivalence, and some related problems such as, e.g., intersection emptiness or bounded universality or equivalence, for finite automata in more detail. Obviously, these standard problems are related to each other and we will briefly discuss their relations and moreover some consequences to the complexity of some non-trivial properties for

problems on DFAs, NFAs, and AFAs. The *non-emptiness problem for NFAs* is defined as follows:

– Given a nondeterministic finite automaton A, is $L(A) \neq \emptyset$?

Moreover, the *universality problem for NFAs* is:

– Given a nondeterministic finite automaton A with input alphabet Σ, is the language $L(A)$ universal, i.e., $L(A) = \Sigma^*$?

The *equivalence problem for NFAs* is defined for two devices as:

– Given two nondeterministic finite automata A_1 and A_2, is $L(A_1) = L(A_2)$?

This notation naturally generalizes to other types of finite automata.

Intuitively, the universality problem can be much harder than the corresponding emptiness problem, which may also be true for the equivalence problem and the universality problem. For instance, it is easy to see that emptiness reduces to non-universality if the automata are logspace effectively closed under arbitrary homomorphism and concatenation with regular languages. Here a class of automata C is logspace effectively closed under arbitrary homomorphism, if for any automaton A from C with n states and any homomorphism h, one can construct within deterministic logspace an automaton B from C that accepts language $h(L(A))$. This implies that the number of states of B is bounded by some polynomial $p_R(n)$. Similarly logspace effective closure under concatenation with regular languages is defined. More general conditions for logspace many-one reductions of universality or emptiness to equivalence, where one of the languages is a fixed language, were studied in detail in a series of papers by Hunt III and co-authors [37,38,39].

Now let us come to the complexity of the emptiness problem for finite automata. In general, if automata are logspace effectively closed under intersection with regular sets, then the general membership logspace many reduces to the non-emptiness problem for the same type of automata, because $w \in L(A)$ if and only if $L(A) \cap \{w\} \neq \emptyset$. Conversely, non-emptiness logspace many-one reduces to general membership, if the automata are logspace effectively closed under homomorphism, since $L(A) \neq \emptyset$ if and only if $h(L(A)) \neq \emptyset$ if and only if $\lambda \in h(L(A))$, where $h(a) = \lambda$, for $a \in \Sigma$. In [47] the following result on the non-emptiness problem for NFAs was shown, which even holds for DFAs—since nondeterministic space complexity classes are closed under complementation [43,66] the result also holds for the emptiness problem. Moreover, non-emptiness for AFAs was considered in [45] and [28].

Theorem 13 (Non-Emptiness). *The non-emptiness problem for NFAs and DFAs is* NL*-complete, and it is* PSPACE*-complete for AFAs. The results remain valid for automata accepting unary languages.*

A natural variant of non-emptiness is *intersection non-emptiness*. This is the problem to decide whether $\bigcap_{1 \leq i \leq n} L(A_i) \neq \emptyset$, for given finite automata A_1, A_2, ..., A_n? If the number of automata in the input instance is bounded by

some function $g(n)$, then this problem is referred to as the $g(n)$-*bounded intersection non-emptiness problem*. For easier readability we abbreviate the former problem by $\emptyset \neq \bigcap C$ and the latter one by $\emptyset \neq \bigcap^{g(n)} C$, where C is from $\{\mathrm{DFA}, \mathrm{NFA}, \mathrm{AFA}\}$. Trivially, non-emptiness logspace many-one reduces to intersection non-emptiness, even to k-bounded intersection non-emptiness for constant k. The results on these problems read as follows: In [48] it was shown that $\emptyset \neq \bigcap \mathrm{DFA}$ is PSPACE-complete. Since $\emptyset \neq \bigcap \mathrm{AFA}$ can be decided within nondeterministic polynomial space, $\emptyset \neq \bigcap \mathrm{NFA}$ and $\emptyset \neq \bigcap \mathrm{AFA}$ are PSPACE-complete, too. Recently it was shown in [4] that the *infinite cardinality intersection problem*, i.e., given automata $A_1, A_2, \ldots, A_n$ from C, does there exist infinitely many words in $\bigcap_{1 \leq i \leq n} L(A_i)$ is also PSPACE-complete, for DFAs. Further PSPACE-complete problems on NFAs based on pattern and power acceptance were identified in [4].

For DFAs and NFAs these intractable intersection emptiness problems become feasible, if the number of finite automata is bounded by some constant k, but remains intractable for AFAs. More precisely, both the k-bounded intersection non-emptiness problems $\emptyset \neq \bigcap^k \mathrm{DFA}$ and $\emptyset \neq \bigcap^k \mathrm{NFA}$ are NL-complete, for each k with $k \geq 1$, by [19], and $\emptyset \neq \bigcap^k \mathrm{AFA}$ remains obviously PSPACE-complete. Moreover, for the bounded intersection non-emptiness problem in general it was shown in [49] that both $\emptyset \neq \bigcap^{g(n)} \mathrm{DFA}$ and $\emptyset \neq \bigcap^{g(n)} \mathrm{NFA}$ are complete for $\mathsf{NSPACE}(g(n) \cdot \log n)$. In particular, both $\emptyset \neq \bigcap^{\log^{k-1} n} \mathrm{DFA}$ and $\emptyset \neq \bigcap^{\log^{k-1} n} \mathrm{NFA}$ are $\mathsf{NSPACE}(\log^k n)$-complete, for $k \geq 1$. Observe, that these were the first natural complete problems for this complexity class. Finally, what can be said about the (bounded) intersection non-emptiness problem for the automata under consideration, when restricted to unary input alphabet? As a consequence of [19] and [65] both $\emptyset \neq \bigcap \text{Tally-DFA}$ and $\emptyset \neq \bigcap \text{Tally-NFA}$ are NP-complete, while $\emptyset \neq \bigcap \text{Tally-AFA}$ again remains PSPACE-complete [28]—the abbreviation of the problem instance are self-explaining. The latter result also holds for the bounded variant, even for constant k. In [49] it is briefly mentioned that $\emptyset \neq \bigcap^k \text{Tally-DFA}$ is L- and $\emptyset \neq \bigcap^k \text{Tally-NFA}$ is NL-complete. On the other hand, *completeness* results for the bounded intersection non-emptiness problem are not known for unary languages, as in the general case. Nevertheless, involved upper and lower bounds by simultaneously bounded complexity classes (time, space, and number of nondeterministic steps) were shown in [49].

Another problem closely connected to non-emptiness is the so-called *short word problem*, which was investigated in [49], too. The main idea underlying short words is that in general the shortest word accepted of an NFA can be of exponential length in the coding of this automaton. Thus, the natural question arises whether the automaton accepts words which are "short" in some sense. Regarding words of linear length as short, we can define the *short word problem* as follows: Given a finite automaton A, is there a word w of length less or equal than the coding of A, such that $w \in L(A)$? We abbreviate this problem by $\emptyset \neq C_{lin}$ and $\emptyset \neq \text{Tally-}C_{lin}$ when the automata accept unary languages, where $C \in \{\mathrm{DFA}, \mathrm{NFA}, \mathrm{AFA}\}$. Using standard methods one sees

that $\emptyset \neq \mathrm{DFA}_{lin}$ and $\emptyset \neq \mathrm{NFA}_{lin}$ are NL-complete, while $\emptyset \neq \text{Tally-DFA}_{lin}$ is L-complete and $\emptyset \neq \text{Tally-NFA}_{lin}$ is NL-complete. For AFAs it was shown in [28] that $\emptyset \neq \mathrm{AFA}_{lin}$ is NP- and $\emptyset \neq \text{Tally-AFA}_{lin}$ is P-complete. Considering the combination of the (bounded) intersection non-emptiness problem with the short word restriction leads to more interesting results. We refer to these problems as $\emptyset \neq \bigcap^{g(n)} C_{lin}$ and $\emptyset \neq \bigcap^{g(n)} \text{Tally-}C_{lin}$, respectively. The problems $\emptyset \neq \bigcap^{g(n)} \mathrm{DFA}_{lin}$ and $\emptyset \neq \bigcap^{g(n)} \mathrm{NFA}_{lin}$ are complete for simultaneously time and space bounded classes between $\mathsf{NTISP}(\mathrm{pol}\, n, \log n) = \mathsf{NL}$ and $\mathsf{NTISP}(\mathrm{pol}\, n, \mathrm{pol}\, n) = \mathsf{NP}$, namely for $\mathsf{NTISP}(\mathrm{pol}\, n, g(n) \cdot \log n)$—see [49]. For $g(n) = \log^k n$ these classes are the nondeterministic counterparts of the SC^k-hierarchy. The restriction of these problems with respect to DFAs (NFAs, respectively) to short words, always leads to L-complete (NL-complete, respectively) sets, regardless of the function $g(n)$. The corresponding problems for AFAs, namely $\emptyset \neq \bigcap^{g(n)} \mathrm{AFA}_{lin}$ and $\emptyset \neq \bigcap^{g(n)} \text{Tally-AFA}_{lin}$ are P-complete, also regardless of $g(n)$.

Now let us consider the next standard problem, the universality problem. As previously mentioned, emptiness and universality are closely related to each other by the complementation operation. The universality problem for DFAs was shown to be NL-complete [12]. For NFAs and AFAs, respectively, the problem under consideration was investigated in [1,57] and [28], respectively, in more detail. For the results on automata accepting unary languages we refer to [65] and [28]. We summarize these results in the following theorem.

Theorem 14 (Universality). *The universality problem for DFAs is* NL*-complete and for NFAs and AFAs it is* PSPACE*-complete. For automata accepting unary languages, the universality problem is* NL*-complete for DFAs,* coNP*-complete for NFAs, and* PSPACE*-complete for AFAs.*

Next we consider two variants of universality. The first one is the *union universality problem*, that is to decide for given automata $A_1, A_2, \ldots, A_n$, whether $\bigcup_{1 \leq i \leq n} L(A_i) = \Sigma^*$? Trivially, universality logspace many-one reduces to the union universality problem for any class of automata. For DFAs this problem is readily seen to be PSPACE-complete by a reduction from the intersection emptiness problem for DFAs, which was discussed in detail earlier. For NFAs and AFAs the union universality problem is PSPACE-complete, too, since it is already PSPACE-hard for a single automaton, and containment can easily be seen since NFAs and AFAs are logspace effective closed under union. Thus, further variants of this problem, comparable to variants of the intersection emptiness problem, are not worth studying. Another, not to well-known generalization of the universality problem is the bounded universality problem first studied in [12]. The *bounded universality problem* is the problem of deciding for a given finite automaton A and a unary integer n, whether $L(A) \cap \Sigma^{\leq n} = \Sigma^{\leq n}$? The *bounded non-universality problem* is defined accordingly. In [12] it was shown that the bounded universality problem for NFAs is coNP-complete, while it is NL-complete for DFAs. Thus, the complexity of non-bounded universality is significantly lower than that of the equivalence problem, which is discussed below.

Regarding the problem of computing the lexically first witness string that proves bounded non-universality for NFAs, an Δ_2^{P} upper bound, and NP- and coNP-hardness lower bounds were shown in [12]. Computing any witness string, thus dropping the lexically first criterion, leads to a problem that is computational equivalent to the bounded non-universality problem and, thus, is an NP-complete problem for NFAs. For AFAs the bounded universality is seen to be coNP-complete.

The last standard problem we are interested in, is the equivalence problem. Besides the emptiness problem, the equivalence problem is certainly one of the most important decision problems that has been investigated extensively in the literature. That equivalence is harder than emptiness is (partially) true for DFAs and NFAs, because equivalence is NL-complete for deterministic [12] and PSPACE-complete for NFAs. However, in case of AFAs equivalence remains as hard as emptiness as shown in [45]. Automata on unary input alphabet were investigated in [28,65]. As the reader may notice, universality and equivalence are computational equivalent with respect to logspace many-one reductions for the finite automata types under consideration.

Theorem 15 (Equivalence). *The equivalence problem for DFAs is* NL*-complete and for NFAs and AFAs it is* PSPACE*-complete. For automata accepting unary languages, the universality problem is* NL*-complete for DFAs,* coNP*-complete for NFAs, and* PSPACE*-complete for AFAs.*

Most of the presented results on emptiness, universality, and equivalence date back to the pioneering papers [57,64,65] and [27,36,37,38,39,62], where mostly problems on regular-like expressions were investigated. Obviously, a lower bound on the computational complexity of a problem for ordinary regular expressions implies the same lower bound for NFAs, since any regular expression of size n can be converted into an equivalent $(n+1)$-state NFA [42]. Most of these results on regular expressions are summarized in [20]—e.g., one can read the following entry, literally taken from [20], on inequivalence for regular expressions:

> "[The inequivalence for regular expressions r_1 and r_2, i.e., deciding whether $L(r_1) \neq L(r_2)$, is ...] PSPACE-complete, even if $|\Sigma| = 2$ and $L(r_2) = \Sigma^*$. In fact, PSPACE-complete if r_2 is any fixed expression representing an "unbounded" language [39]. NP-complete for fixed r_2 representing any infinite "bounded" language, but solvable in polynomial time for fixed r_2 representing any finite language. The general problem remains PSPACE-complete if r_1 and r_2 both have "star-height" k for fixed $k \geq 1$ [39], but is NP-complete for $k = 0$ ("star-free") [34,65]. Also NP-complete if one of both of r_1 and r_2 represent bounded languages (a property that can be checked in polynomial time) [39] or if $|\Sigma| = 1$ [65]. For related results and intractable generalizations, see cited references, [35], and [38]."

Here a language L is *bounded* if and only if there exist words $w_1, w_2, \ldots, w_k$, for some k, such that $L \subseteq w_1^* w_2^* \ldots w_k^*$. In [41] it was shown that boundedness

is a necessary and sufficient condition for context-free languages to be sparse. A language $L \subseteq \Sigma^*$ is *sparse*, if there exists a polynomial p such that for all n we have $|L \cap \Sigma^{\leq n}| \leq p(n)$, where $\Sigma^{\leq n}$ is the set of all words over Σ of length at most n. While boundedness for languages specified by regular expressions is easily shown to be solvable in polynomial time *via* an inductive proof [39], it is not that clear, whether this also holds for NFAs. Here the equivalence of boundedness and sparseness for context-free languages comes into play. The *sparseness problem*, i.e., given an automaton A, is $L(A)$ sparse?, was shown to be NL-complete for both DFAs and NFAs [40], and for AFAs it is PSPACE-complete. For automata accepting unary languages the problem under consideration is trivial. Hence, the boundedness problem for NFAs is efficiently solvable.

Next we summarize some results on some problems related to universality and equivalence, namely the segment equivalence and the closeness problem: (1) The *segment equivalence problem* is defined as follows: Given two automata A_1 and A_2 and n, is $L(A_1) \cap \Sigma^{\leq n} = L(A_2) \cap \Sigma^{\leq n}$? If n is coded in binary, it is called the *binary-encoded segment equivalence problem*. Segment and binary-encoded segment equivalence were studied in [40]. There it was shown that segment equivalence for DFAs is NL-complete, whereas for NFAs the problem becomes coNP-complete. As in case of ordinary equivalence one can show that the complexity of segment equivalence for AFAs is the same as for NFAs, hence coNP-complete, if the input alphabet contains at least two letters. For automata accepting unary languages it is easy to see that the segment equivalence problem is L-complete for DFAs, NL-complete for NFAs, and P-complete for AFAs. Moreover, for the binary-encoded segment equivalence problem it was shown that both NFAs and AFAs induce a PSPACE-complete problem [40]. (2) The closeness problem measures the similarity of languages in terms of the density of their symmetric difference, i.e., two languages L_1 and L_2 are *close* if and only if $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ is sparse. Thus, the *closeness problem* is to decide whether for given two automata A_1 and A_2, the symmetric difference of $L(A_1)$ and $L(A_2)$ is sparse? In [40] it was shown that the closeness problem for DFAs is NL-complete and for NFAs it is PSPACE-complete. Moreover, PSPACE-completeness also holds for the closeness problem for AFAs. Note, that the closeness problem for automata accepting unary languages is trivial.

Along the lines of development in computational complexity theory, authors began to study functional problems and classes, see, e.g., [2]. One of the most easiest functional problems for finite state devices is *census*. Here for a given finite automaton A and 1^n, one asks how many words up to length n are accepted by A? Other well known functional problems are *census of the complement*, *ranking*, *maximal word*, and *maximal relative word*—for a precise definition of these problems we refer to [2]. For DFAs and NFAs it was shown in [2] that most of these problems are complete for logarithmic space bounded counting classes like, e.g., #L, spanL, or optL, while for AFAs these problems turn out to be complete for their polynomially time bounded counterparts #P or optP [28].

We have seen that most problems for NFAs and AFAs are intractable, while some problem for DFAs are effectively solvable. In the remainder of this

subsection we consider two results of Hunt III and co-authors [27,38], which show that the above mentioned behavior on the computational complexity of DFA and NFA based problems is not accidental. It thus explains in part, why most problems for NFAs and AFAs are intractable. We feel that these nice results demand more attention, therefore we present it here.

Theorem 16 (Hardness for DFAs and NFAs Problems). *Let Σ be an alphabet with $|\Sigma| \geq 2$ and $P : 2^{\Sigma^*} \to \{0,1\}$ be any non-trivial predicate on the regular languages. Assume that the set P_{left} of all languages $\delta_x(L)$, where L is a regular language, $P(L)$ is true, and $x \in \Sigma^*$, is* not equal *to the family of all regular languages—here $\delta_x(L) = \{ y \in \Sigma^* \mid xy \in L \}$ refers to the left quotient of L with respect to the word x from Σ^*. Then the P-problem for NFAs, that is, to determine whether for a given nondeterministic finite automaton A the predicate P on $L(A)$ is true, is* PSPACE*-hard, assuming $P(\Sigma^*)$ to be true. Moreover, the corresponding P-problem for DFAs is at least* NL*-hard.*

Finally, we summarize some results on the operation problem from the computational complexity perspective. For a survey on the descriptional complexity of the operation problem for DFAs and NFAs we refer to [68] and [29,30]. Let $\circ$ be a fixed operation on languages that preserves regularity. Then the $\circ$-*operation problem* is defined as follows: Given finite automata A_1, A_2, and A_3, is $L(A_1) \circ L(A_2) = L(A_3)$? Obviously, this problem generalizes to unary language operations. It turned out that both the concatenation operation problem and the Kleene star operation for DFAs are PSPACE-complete [46]. A converse problem to the $\circ$-operation problem is the *minimum $\circ$-problem*. That is, given a finite automaton A and an integer k, are there finite automata A_1 and A_2 of the same type with $|A_1| + |A_2| \leq k$ such that $L(A_1) \circ L(A_2) = L(A)$? For DFAs this problem is NP-complete for union and intersection, and PSPACE-complete for concatenation and Kleene star. Interestingly, the minimum reverse-operation problem is shown to be solvable in polynomial time if the integer k is given in unary, although DFAs are not logspace effective closed under reversal.

3.3 Minimization of Finite Automata

The study of the minimization problem for finite automata dates back to the early beginnings of automata theory. Here we focus mainly on some recent developments related to this fundamental problem—for further reading we refer to [46] and references therein. The minimization problem is also of practical relevance, because regular languages are used in many applications, and one may like to represent the languages succinctly. The decision version of the minimization problem, for short the *NFA-to-NFA minimization problem*, is defined as follows:

– Given a *nondeterministic* finite automaton A and a natural number k in binary, that is, an encoding $\langle A, k \rangle$, is there an equivalent k-state *nondeterministic* finite automaton?

This notation naturally generalizes to other types of finite automata, for example, the DFA-to-NFA minimization problem. It is well known that for a given

n-state DFA one can efficiently compute an equivalent minimal automaton in $O(n \log n)$ time [32]. More precisely, the DFA-to-DFA minimization problem is complete for NL, even for DFAs without inaccessible states [12]. This is contrary to the nondeterministic case since the NFAs minimization problem is known to be computationally hard [46], which is also true for AFAs. The PSPACE-hardness result for NFAs was shown by a reduction from the union universality problem to the NFA-to-NFA minimization problem. For some further problems related to minimization we refer also to [24].

Theorem 17 (Minimization). *The DFA-to-DFA minimization problem is* NL*-complete, while the NFA-to-NFA minimization problem is* PSPACE*-complete, even if the input is given as a deterministic finite automaton. The AFA-to-AFA minimization problem is* PSPACE*-complete, too.*

In order to better understand the very nature of nondeterminism one may ask for minimization problems for restricted types of finite automata. Already in [46] it was shown that for the restricted class of unambiguous finite automata (UFA) some minimization problems remain intractable. To be more precise, the UFA-to-UFA and the DFA-to-UFA minimization problems are NP-complete. We mention in passing that in [12] necessary and sufficient conditions were provided to distinguish between exponential, polynomial, bounded, and k-bounded ambiguity, and it was shown that these *ambiguity problems*, i.e., determining whether the degree of ambiguity of a given NFA is exponential, polynomial, bounded, k-bounded, where k is a fixed integer, or unambiguous are all NL-complete.

Later in [52] it was shown that the minimization of finite automata equipped with a very small amount of nondeterminism is already computationally hard. To this end, a reduction from the NP-complete minimal inferred DFA problem [21,46] to the the minimization problems for multiple initial state deterministic finite automata with a fixed number of initial states (MDFA) as well as for nondeterministic finite automata with fixed finite branching has been shown. Prior to this, the MDFA-to-DFA minimization problem in general was proven to be PSPACE-complete in [31]. Here the minimal inferred DFA problem [21] is defined as follows: Given a finite alphabet Σ, two finite subsets $S, T \subseteq \Sigma^*$, and an integer k, is there an k-state DFA that accepts a language L such that $S \subseteq L$ and $T \subseteq \Sigma^* \setminus L$? Such an automaton can be seen as a consistent "implementation" of the sets S and T. Recently, the picture was completed in [8] by getting much closer to the tractability frontier for nondeterministic finite automata minimization. There a class of NFAs is identified, the so called δ-nondeterministic finite automata (δNFA), such that the minimization problem for any class of finite automata that contains δNFAs is NP-hard, even if the input is given as a DFA. Here the class of δNFAs contains all NFAs with the following properties: (i) The automaton is unambiguous, (ii) the maximal product of the degrees of nondeterminism over the states in a possible computation is at most 2, and (iii) there is at most on state q and a letter a such that the degree of nondeterminism of q and a is 2. It is worth mentioning that for every n-state δNFA there is an equivalent DFA with at most $O(n^2)$ states.

The situation for the minimization problem in general is, in fact, even worse. Recent work [23] shows that the DFA-to-NFA problem cannot be approximated within $\sqrt{n}/\mathsf{polylog}n$ for state minimization and $n/\mathsf{polylog}n$ for transition minimization, provided some cryptographic assumption holds. Moreover, the NFA-to-NFA minimization problem was classified to be inapproximable within $o(n)$, unless $\mathsf{P} = \mathsf{PSPACE}$, if the input is given as an NFA with n states [23]. That is, no polynomial-time algorithm can determine an approximate solution of size $o(n)$ times the optimum size. Even the DFA-to-NFA minimization problem remains inapproximable within a factor of at least $n^{1/3-\epsilon}$, for all $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$ [26], for alphabets of size $O(n)$, and not approximable within $n^{1/5-\epsilon}$ for a binary alphabet, for all $\epsilon > 0$. Under the same assumption, it was shown that the transition minimization problem for binary input alphabets is not approximable within $n^{1/5-\epsilon}$, for all $\epsilon > 0$. The results in [26] proved approximation hardness results under weaker (and more familiar) assumptions than [23]. Further results on the approximability of the minimization problem when the input is specified as regular expression or a truth table can be found in [23,26].

The unary NFA-to-NFA minimization problem is coNP-hard [65], and similarly as in the case of finite languages contained in Σ_2^{P}. The number of states of a minimal NFA equivalent to a given unary cyclic DFA cannot be computed in polynomial time, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log n)})$ [44]. Note that in the latter case the corresponding decision version belongs to NP. Inapproximability results for the problem in question have been found during the last years, if the input is a unary NFA: The problem cannot be approximated within $\frac{\sqrt{n}}{\ln n}$ [22], and if one requires in addition the explicit construction of an equivalent NFA, the inapproximability ratio can be raised to $n^{1-\epsilon}$, for every $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$ [23]. On the other hand, if a unary *cyclic* DFA with n states is given, the nondeterministic state complexity of the considered language can be approximated within a factor of $O(\log n)$. The picture on the unary NFA-to-NFA minimization problem was completed in [25]. Some of the aforementioned (in)approximability results, which only hold for the *cyclic* case, generalize to unary languages in general. In particular, it was shown that for a given n-state NFA accepting a unary language, it is impossible to approximate the nondeterministic state complexity within $o(n)$, unless $\mathsf{P} = \mathsf{NP}$. Observe that this bound is tight. In contrast, it is proven that the NFA-to-NFA minimization problem can be *constructively* approximated within $O(\sqrt{n})$, where n is the number of states of the given DFA. Here by *constructively approximated* we mean that we can build the nondeterministic finite automaton, instead of only approximately determining the number of states needed. This solves an open problem stated in [46] on the complexity of converting a DFA to an approximately optimal NFA in the case of unary languages.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addision-Wesley, Reading (1974)
2. Álvarez, C., Jenner, B.: A very hard log-space counting class. Theoret. Comput. Sci. 107, 3–30 (1993)

3. Ajtai, M.: Σ_1^1 formulae on finite structures. Ann. Pure. Appl. Logic 24, 1–48 (1983)
4. Anderson, T., Rampersad, N., Santean, N., Shallit, J., Loftus, J.: Detecting palindroms, patterns and borders in regular languages (2008) arXiv:0711.3183v2 [cs.CC]
5. Barrington, D.A.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. J. Comput. System Sci. 38, 150–164 (1989)
6. Barrington, D.A.M., Immerman, N., Straubing, H.: On uniformity within NC^1. J. Comput. System Sci. 41, 274–306 (1990)
7. Barrington, D.M., Thérien, D.: Finite monoids and the fine structure of NC^1. J. ACM 35, 941–952 (1988)
8. Björklund, H., Martens, W.: The tractability frontier for NFA minimization. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 27–38. Springer, Heidelberg (2008)
9. Brzozowski, J.A., Leiss, E.L.: On equations for regular languages, finite automata, and sequential networks. Theoret. Comput. Sci. 10, 19–35 (1980)
10. Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. J. ACM 21, 114–133 (1981)
11. Cho, S., Huynh, D.T.: Finite-automaton aperiodicity is PSPACE-complete. Theoret. Comput. Sci. 88, 99–116 (1991)
12. Cho, S., Huynh, D.T.: The parallel complexity of finite-state automata problems. Inform. Comput. 97, 1–22 (1992)
13. Chrobak, M.: Finite automata and unary languages. Theoret. Comput. Sci. 47, 149–158 (1986)
14. Chrobak, M.: Errata to finite automata and unary languages. Theoret. Comput. Sci. 302, 497–498 (2003)
15. Cohen, R.S., Brzozowski, J.A.: Dot-depth of star-free events. J. Comput. System Sci. 5, 1–16 (1971)
16. Eilenberg, S.: Automata, Languages, and Machines (Volume B), vol. 59-B. Academic Press, London (1976)
17. Fellah, A., Jürgensen, H., Yu, S.: Constructions for alternating finite automata. Internat. J. Comput. Math. 35, 117–132 (1990)
18. Furst, M.L., Saxe, J.B., Sipser, M.: Parity, circuits, and the polynomial-time hierarchy. Math. Systems Theory 17, 13–27 (1984)
19. Galil, Z.: Hierarchies of complete problems. Acta Inform. 6, 77–88 (1976)
20. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
21. Gold, E.M.: Complexity of automaton identification from given data. Inform. Control 37, 302–320 (1978)
22. Gramlich, G.: Probabilistic and nondeterministic unary automata. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 460–469. Springer, Heidelberg (2003)
23. Gramlich, G., Schnitger, G.: Minimizing nFA's and regular expressions. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 399–411. Springer, Heidelberg (2005)
24. Gruber, H., Holzer, M.: Finding lower bounds for nondeterministic state complexity is hard. In: H. Ibarra, O., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 363–374. Springer, Heidelberg (2006)
25. Gruber, H., Holzer, M.: Computational complexity of NFA minimization for finite and unary languages. In: Language and Automata Theory and Applications (LATA 2007), pp. 261–272. Technical Report 35/07, Universitat Rovira i Virgili, Tarragona (2007)

26. Gruber, H., Holzer, M.: Inapproximability of nondeterministic state and transition complexity assuming P $\neq$ NP. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 205–216. Springer, Heidelberg (2007)
27. Hartmanis, J., Hunt III, H.B.: The LBA problem and its importance in the theory of computing. In: SIAM AMS. Complexity of Computing, vol. 7, pp. 1–26 (1974)
28. Holzer, M.: On emptiness and counting for alternating finite automata. In: Developments in Language Theory II; at the Crossroads of Mathematics, Computer Science and Biology, pp. 88–97. World Scientific, Singapore (1996)
29. Holzer, M., Kutrib, M.: State complexity of basic operations on nondeterministic finite automata. In: Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2002. LNCS, vol. 2608, pp. 148–157. Springer, Heidelberg (2003)
30. Holzer, M., Kutrib, M.: Nondeterministic finite automata—recent results on the descriptional and computational complexity. In: Ibarra, O., Ravikumar, B. (eds.) CIAA 2008. LNCS, vol. 5148, pp. 1–16. Springer, Heidelberg (2008)
31. Holzer, M., Salomaa, K., Yu, S.: On the state complexity of k-entry deterministic finite automata. J. Autom., Lang. Comb. 6, 453–466 (2001)
32. Hopcroft, J.: An $n \log n$ algorithm for minimizing the state in a finite automaton. In: The Theory of Machines and Computations, pp. 189–196. Academic Press, London (1971)
33. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)
34. Hunt III, H.B.: On the time and tape complexity of languages. Ph.D. thesis, Cornell University, Ithaca, New York, USA (1973)
35. Hunt III, H.B.: On the time and tape complexity of languages I. In: Symposium on Theory of Computing (STOC 1973), pp. 10–19. ACM Press, New York (1973)
36. Hunt III, H.B.: Observations on the complexity of regular expressions problems. J. Comput. System Sci. 19, 222–236 (1979)
37. Hunt III, H.B., Rosenkrantz, D.J.: On equivalence and containment problems for formal languages. J. ACM 24, 387–396 (1977)
38. Hunt III, H.B., Rosenkrantz, D.J.: Computational parallels between the regular and context-free languages. SIAM J. Comput. 7, 99–114 (1978)
39. Hunt III, H.B., Rosenkrantz, D.J., Szymanski, T.G.: On the equivalence, containment, and covering problems for the regular and context-free languages. J. Comput. System Sci. 12, 222–268 (1976)
40. Huynh, D.T.: Complexity of closeness, sparseness and segment equivalence for context-free and regular languages. In: Informatik, Festschrift zum 60, Geburtstag von Günter Hotz, Teubner, pp. 235–251 (1992)
41. Ibarra, O.H., Ravikumar, B.: On sparseness, ambiguity and other decision problems for acceptors and transducers. In: Monien, B., Vidal-Naquet, G. (eds.) STACS 1986. LNCS, vol. 210, pp. 171–179. Springer, Heidelberg (1985)
42. Ilie, L., Yu, S.: Follow automata. Inform. Comput. 186, 140–162 (2003)
43. Immerman, N.: Nondeterministic space is closed under complementation. SIAM J. Comput. 17, 935–938 (1988)
44. Jiang, T., McDowell, E., Ravikumar, B.: The structure and complexity of minimal NFAs over a unary alphabet. Internat. J. Found. Comput. Sci. 2, 163–182 (1991)
45. Jiang, T., Ravikumar, B.: A note on the space complexity of some decision problems for finite automata. Inform. Process. Lett. 40, 25–31 (1991)
46. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM J. Comput. 22, 1117–1141 (1993)
47. Jones, N.: Space-bounded reducibility among combinatorial problems. J. Comput. System Sci. 11, 68–85 (1975)

48. Kozen, D.: Lower bounds for natural proof systems. In: Foundations of Computer Science (FOCS 1977), pp. 254–266. IEEE Society Press, Los Alamitos (1977)
49. Lange, K.J., Rossmanith, P.: The emptiness problem for intersections of regular languages. In: Havel, I.M., Koubek, V. (eds.) MFCS 1992. LNCS, vol. 629, pp. 346–354. Springer, Heidelberg (1992)
50. Leiss, E.: Succinct representation of regular languages by Boolean automata. Theoret. Comput. Sci. 13, 323–330 (1981)
51. Leiss, E.: Succinct representation of regular languages by Boolean automata. II. Theoret. Comput. Sci. 38, 133–136 (1985)
52. Malcher, A.: Minimizing finite automata is computationally hard. Theoret. Comput. Sci. 327, 375–390 (2004)
53. McKenzie, P., Péladeau, P., Thérien, D.: NC^1: The automata-theoretical viewpoint. Comput. Compl. 1, 330–359 (1991)
54. McNaughton, R., Papert, S.: Counter-free automata. Research monographs, vol. 65. MIT Press, Cambridge (1971)
55. Mereghetti, C., Pighizzini, G.: Optimal simulations between unary automata. SIAM J. Comput. 30, 1976–1992 (2001)
56. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: IEEE Symposium on Switching and Automata Theory (SWAT 1971), pp. 188–191. IEEE Society Press, Los Alamitos (1971)
57. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential time. In: Symposium on Switching and Automata Theory (SWAT 1972), pp. 125–129. IEEE Society Press, Los Alamitos (1972)
58. Moore, F.R.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Trans. Comput. 20, 1211–1214 (1971)
59. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. 3, 114–125 (1959)
60. Schützenberger, M.P.: On finite monoids having only trivial subgroups. Inform. Control 8, 190–194 (1965)
61. Sipser, M.: Lower bounds on the size of sweeping automata. J. Comput. System Sci. 21, 195–202 (1980)
62. Stearns, R.E., Hunt, H.B.: On the equivalence and containment problems for unambiguous regular expressions, regular grammars, and finite automata. SIAM J. Comput. 14, 598–611 (1985)
63. Stern, J.: Complexity of some problems from the theory of automata. Inform. Control 66, 163–176 (1985)
64. Stockmeyer, L.J.: The Complexity of Decision Problems in Automata Theory and Logic. Ph.D. thesis, MIT, Cambridge, Massasuchets, USA (1974)
65. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: Symposium on Theory of Computing (STOC 1973), pp. 1–9. ACM Press, New York (1973)
66. Szelepcsényi, R.: The method of forced enumeration for nondeterministic automata. Acta Inform. 26, 279–284 (1988)
67. Thérien, D.: Classification of finite monoids: the language approach. Theoret. Comput. Sci. 14, 195–208 (1981)
68. Yu, S.: State complexity of regular languages. J. Autom., Lang. Comb. 6, 221–234 (2001)

Hypothesis Spaces for Learning

Sanjay Jain*

Department of Computer Science,
National University of Singapore, Singapore 117590, Republic of Singapore
sanjay@comp.nus.edu.sg

Abstract. In this paper we survey some results in inductive inference showing how learnability of a class of languages may depend on hypothesis space chosen. We also discuss results which consider how learnability is effected if one requires learning with respect to every suitable hypothesis space. Additionally, optimal hypothesis spaces, using which every learnable class is learnable, is considered.

1 Introduction

A learning scenario can be described as follows. Consider a learner (a computable device) receiving data, one piece at a time, about some target concept (which is from a class of possible concepts). As the learner is receiving its data, it conjectures a possible description of the target concept. One may consider the learner to be successful if its sequence of conjectures converges to a correct description of the target concept.

In this paper we will be mostly concerned with language learning. A language is some recursively enumerable (r.e.) subset of a universal set. By appropriate coding, one may take the universal set to be the set of natural numbers, $\mathbb{N} = \{0, 1, 2, \ldots\}$. For learning languages, the data provided to the learner is usually the set of positive examples of the language, one element at a time, where all the elements are eventually provided and no non-elements of the language are provided. This form of data presentation is called a text for the language. This model originates from the observation that in many natural situations, such as language learning by children or in astronomy, one gets essentially only positive data. A related model of data presentation, called informant, is when the learner is presented with all elements of the universal set, appropriately classified as positive or negative with respect to the target language. The conjectures of the learner take the form of a grammar from some hypothesis space (we always assume that hypothesis space is an r.e. indexing of r.e. languages; in some cases we additionally assume that membership question for hypothesis i is decidable effectively in i — in these cases the hypothesis space is an indexed family of recursive languages). A criterion of success, as considered above, is for the sequence of grammars to converge to a grammar for the target language. This is essentially the criterion of learning first considered by Gold [16], and commonly called explanatory learning (abbreviated **TxtEx**-learning, for explanatory learning from

* Supported in part by NUS grant number R252-000-308-112.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 43–58, 2009.

text, and **InfEx**-learning, for explanatory learning from informant). Note that the learner is expected to succeed on all possible order of presentation of the elements of the target language.

Learning of one language is usually not much interesting as some learner, which always outputs the grammar for this language, will succeed. What is more interesting is whether some learner can learn all the languages from a class $\mathcal{L}$ of languages.

We now formally define the criterion described above. A *sequence* is a mapping from $\mathbb{N}$ or an initial segment of $\mathbb{N}$ to $\mathbb{N} \cup \{\#\}$. An *information sequence* is a mapping from $\mathbb{N}$ or an initial segment of $\mathbb{N}$ to $(\mathbb{N} \times \{0,1\}) \cup \{\#\}$. Content of a (information) sequence σ, denoted content(σ), is range(σ) $- \{\#\}$ ($\#$ is considered as a pause symbol, representing no data. This is useful when one considers presenting data for empty language). Let SEQ denote the set of all finite sequences. Let SEG denote the set of all finite information sequences σ such that $\{(x,y),(x,z)\} \subseteq$ content(σ) implies $y = z$ (we only care about consistent information sequences). An infinite sequence T is a *text* for a language L iff content(T) $= L$. We let T (with or without subscripts/superscripts) range over texts. An infinite information sequence I is an *informant* for a language L iff content(I) $= \{(x, \chi_L(x)) : x \in \mathbb{N}\}$, where χ_L is the characteristic function for the language L. We let I (with or without subscripts/superscripts) range over informants. $T[n]$ (respectively $I[n]$) denotes the finite sequence consisting of the first n elements in the sequence T (respectively I).

A *learning machine* (also called a learner) is an algorithmic mapping (possibly partial) from SEQ or SEG (depending on whether one is considering learning from texts or informants) to $\mathbb{N} \cup \{?\}$. A learner M converges on a text T to i (denoted $M(T)\downarrow = i$) iff for all but finitely many n, $M(T[n]) = i$. Convergence on information sequence I is defined similarly.

Here one interprets the output of the learner as an index for some language in a hypothesis space. Thus output i would represent the conjecture H_i, where $H_0, H_1, \ldots$ is the hypothesis space used by the learner. The output of ? denotes that the learner is not making a formal conjecture (this is useful when one considers some special cases, such as bounding the number of mind changes made by the learner).

Definition 1. [16] Fix a hypothesis space $\mathcal{H} = H_0, H_1, \ldots$.

(a) A learner M **TxtEx**$_{\mathcal{H}}$-identifies a language L (written: $L \in$ **TxtEx**$_{\mathcal{H}}(M)$) iff for all texts T for L, (i) $M(T[n])$ is defined for all n, and (ii) there exists an i such that $M(T)\downarrow = i$ and $H_i = L$.
(b) A learner M **TxtEx**$_{\mathcal{H}}$-identifies a class $\mathcal{L}$ of languages (written: $\mathcal{L} \subseteq$ **TxtEx**$_{\mathcal{H}}(M)$) iff it **TxtEx**$_{\mathcal{H}}$-identifies all languages in the class $\mathcal{L}$.
(c) **TxtEx**$_{\mathcal{H}} = \{\mathcal{L} : (\exists M)[M$ **TxtEx**$_{\mathcal{H}}$-identifies $\mathcal{L}\}$.

One can similarly define **InfEx**$_{\mathcal{H}}$-identification, where one replaces texts in the above definition by informants.

As the learner has seen only finitely many inputs before it converges to its final hypothesis, some form of learning must have taken place. We use the terms identify, learn, infer as synonyms for this reason.

Note that in the above model of learning, the learner does not know when it has converged to its final hypothesis. If one additionally requires this kind of ability from the learner, then the learning criterion is equivalent to finite learning, where the learner is allowed to make only one conjecture.

Definition 2. [16] Fix a hypothesis space $\mathcal{H} = H_0, H_1, \ldots$.

(a) A learner M $\mathbf{TxtFin}_{\mathcal{H}}$-identifies a language L (written: $L \in \mathbf{TxtFin}_{\mathcal{H}}(M)$) iff for all texts T for L, there exist i, n such that (i) $H_i = L$, (ii) $(\forall m < n)[M(T[m]) =?]$, and (iii) $(\forall m \geq n)[M(T[m]) = i]$.
(b) A learner M $\mathbf{TxtFin}_{\mathcal{H}}$-identifies a class $\mathcal{L}$ of languages (written: $\mathcal{L} \subseteq \mathbf{TxtFin}_{\mathcal{H}}(M)$) iff it $\mathbf{TxtFin}_{\mathcal{H}}$-identifies all languages in the class $\mathcal{L}$.
(c) $\mathbf{TxtFin}_{\mathcal{H}} = \{\mathcal{L} : (\exists M)[M \text{ } \mathbf{TxtFin}_{\mathcal{H}}\text{-identifies } \mathcal{L}\}$.

One can similarly define $\mathbf{InfFin}_{\mathcal{H}}$-identification.

Since Gold [16], various other criteria of learning have been explored in the literature, specially those which require some additional properties on the conjectures of the learner. We will consider some of them in Section 2. We refer the reader to [3,6,11,19,38,40,43] for some literature on the topic.

Note that the hypothesis space chosen for interpreting the conjectures of the learner may play a crucial role in whether the learner is successful in identifying the language. A commonly used hypothesis space is the standard acceptable programming system: $W_0, W_1, \ldots$ (see [34]). The class $\mathbf{TxtEx}$ does not depend on the exact acceptable programming system chosen, as the acceptable programming system can be translated effectively into each other. However, if one considers other programming systems such as Friedberg numbering [13] as hypothesis space or requires some other properties (such as membership question about hypotheses being decidable, or whether the hypothesis space depends on the class being learnt (for example, not allowing the hypothesis space to contain languages other than those in the class of languages under consideration)), then it may effect the classes which are learnable. This paper surveys some of the results which show how learnability depends on the type of hypothesis spaces allowed.

In Section 2 we define some commonly used criteria of learning. In section 3 we consider the special case of learning indexed families, where often the hypothesis space allowed depends on the class of languages being learnt. In section 4 we consider hypothesis spaces being restricted programming systems, such as Friedberg numberings. In section 5 we consider whether learning is at all possible if one requires learning in all (reasonable) possible hypothesis spaces. In section 6 we consider optimal hypothesis spaces in the sense that if learning a class is possible using any hypothesis space, then the class can be learnt using the given hypothesis space.

In the rest of the paper, for ease of notation, we will omit the hypothesis space from the subscript of learning criteria, and it will be implicit (the allowed hypothesis space may be constrained in some cases due to conventions of the section).

2 Some Further Criteria of Learning

Below we consider the criteria mainly for learning from texts. Similar definitions can be made for learning from informants also. Below, let $\mathcal{H} = H_0, H_1, \ldots$ be the hypothesis space used by the learner.

We first consider two generalizations of explanatory learning. The following generalization considers semantic convergence rather than syntactic convergence to the correct hypothesis by the learner. A learner M is said to *behaviourally correctly* learn (abbreviated: **TxtBc**-learn) [5,10] a language L iff for all texts T for L, for all but finitely many n, $H_{M(T[n])} = L$. One can similarly define **TxtBc** learning of a class, and **TxtBc** the set of all behaviourally correctly learnable classes. It can be shown that **TxtBc** is a strict generalization of **TxtEx**-learning if one allows arbitrary hypothesis spaces [11,5].

The following criterion is somewhere between explanatory and behaviourally correct learning. It allows the learner to eventually vacillate between finitely many correct hypotheses. A learner M is said to *vacillatory* learn (abbreviated: **TxtFex**-learn) [8] a language L iff it **TxtBc**-learns the language L and on all texts T for L, it outputs at most finitely many distinct grammars (in other words, the learner eventually vacillates between finitely many correct grammars for the language). One can similarly define **TxtFex** learning of a class, and **TxtFex** the set of all vacillatorily learnable classes.

We now turn our attention to requiring some properties that the learner (its hypotheses) must satisfy. A learner M is said to be *conservative* [2] on L if for all texts T for L, for all $m > n$, if $M(T[n]) \neq M(T[m])$, then $\text{content}(T[m]) \not\subseteq H_{M(T[n])}$. That is, M changes its hypothesis only if it finds evidence of inconsistency of its earlier conjecture. Learner M conservatively learns (**Conserv**-identifies) L if it **TxtEx**-identifies L and is conservative on L. **Conserv**-identification of a class of languages and the class **Conserv** can be defined similarly. When using acceptable numberings as hypothesis spaces, requiring learners to be conservative is a restriction on the learning capabilities of the machines [2].

A learner M is said to be *strong monotonic* [17] on L if for all texts T for L, for all $m > n$, $H_{M(T[n])} \subseteq H_{M(T[m])}$. A learner M is said to be *monotonic* [39] on L if for all texts T for L, for all $m > n$, $H_{M(T[n])} \cap L \subseteq H_{M(T[m])} \cap L$. A learner M is said to be *weak monotonic* [17] on L if for all texts T for L, for all $m > n$, if $\text{content}(T[m]) \subseteq H_{M(T[n])}$, then $H_{M(T[n])} \subseteq H_{M(T[m])}$ (that is the learner behaves strong monotonically as long as the input data does not contradict the hypothesis conjectured). The criteria of learning corresponding to the above properties being satisfied by the learner, in addition to **TxtEx**-learning the target language, are respectively called **SMon**, **Mon** and **WMon**.

Let $\overline{L}$ denote $\mathbb{N} - L$, the complement of L. [31] considered the dual of above monotonic requirements, where for *dual strong monotonic* learning of L one requires that for all texts T for L, for all $m > n$, $\overline{H_{M(T[n])}} \subseteq \overline{H_{M(T[m])}}$. Similarly, for *dual monotonic* learning of L one requires that for all texts T for L, for all $m > n$, $\overline{H_{M(T[n])}} \cap \overline{L} \subseteq \overline{H_{M(T[m])}} \cap \overline{L}$, and for *dual weak monotonic* learning of L one requires that for all texts T for L, for all $m > n$, if $\text{content}(T[m]) \subseteq H_{M(T[n])}$, then $\overline{H_{M(T[n])}} \subseteq \overline{H_{M(T[m])}}$. The criteria of learning corresponding to the above

properties being satisfied by the learner, in addition to **TxtEx**-learning the target language, are respectively called **DSMon**, **DMon** and **DWMon**.

[31] explore the relationship between the above (dual) monotonic criteria of learning.

A learner M is *consistent* [1,4] on L if for all texts T for L, for all n, content$(T[n]) \subseteq H_{M(T[n])}$. Consistency seems like a natural requirement, as if the hypothesis is not consistent, then it is obviously wrong. However, when using general hypothesis spaces such as acceptable numberings, it can be shown that requiring consistency restricts learning capabilities of the machines [4]. A learner M is *confident* [33] if it converges on every text, even if the text is for a language outside the class of languages being learnt. Confidence is restrictive: it can be shown that even simple classes, such as the class of all finite languages, cannot be learnt confidently. One can define the corresponding learning criteria for learners satisfying consistency and confidence properties (for **I**-learning) similarly. These criteria are called respectively **ConsI** and **ConfI**.

A learner M is *set-driven* [36,32] if content$(\sigma) =$ content(τ) implies $M(\sigma) = M(\tau)$. That is the output of the learner depends only on the content of the input, and not on its length or order. For acceptable programming systems as hypothesis space, it can be shown that set drivenness restricts the learning capabilities of machines [35]. A learner M is rearrangement-independent [6,14] if content$(\sigma) =$ content(τ) and $length(\sigma) = length(\tau)$, implies $M(\sigma) = M(\tau)$. That is the output of the learner depends only on the content and length of the input, and not on the order of the elements in it. Unlike most other requirements considered, rearrangement independence is not restrictive for explanatory learning [14], when one considers acceptable numberings as hypothesis spaces. One can define the corresponding learning criteria for learners satisfying set drivenness and rearrangement independence (for **I**-learning) similarly. These criteria are called s-**I** and r-**I**.

3 Learning Indexed Families

Angluin [2] considered learnability of indexed family of recursive languages. A class of languages $\mathcal{L}$ consisting of languages $L_0, L_1, \ldots$ (with the corresponding indexing) is said to be an indexed family iff there exists a recursive function f such that $f(i, x) = 1$ iff $x \in L_i$. Many of the commonly studied classes of languages, such as the class of regular languages or context free languages, are indexed families.

For learning indexed families, the hypothesis space is usually considered to be an indexed family also. Additionally, one often considers the following requirements (see [25,27]):

(a) the hypothesis space is the class being learnt (with the corresponding indexing) itself; this is called *exact* learning;
(b) the hypothesis space is *class preserving*, that is $\{H_0, H_1, \ldots\} = \{L_0, L_1, \ldots\}$; this is called class preserving learning;

(c) the hypothesis space is *class comprising*, that is $\{H_0, H_1, \ldots\} \supseteq \{L_0, L_1, \ldots\}$; this is called class comprising learning.

Note that in (b) and (c), there are several possible hypothesis spaces that might be used — if learning can be successfully done with respect to any of such hypothesis spaces, then one considers the class to be learnable according to the corresponding criterion.

We prefix E, ϵ or C (where ϵ denotes empty string) to the names of the criteria of learning to denote whether we are considering exact, class preserving or class comprising learning. This convention on criteria names is for this section only.

Lange and Zeugmann [25,27] showed that **ETxtEx** = **TxtEx** = **CTxtEx** and **ETxtFin** = **TxtFin** = **CTxtFin**. Thus, for explanatory and finite learning, choosing an appropriate hypothesis space (in the sense of exact, class preserving or class comprising) is not so crucial.

However, for monotonic learning, the choice of different kind of hypothesis spaces makes a critical difference.

Theorem 3. *The following relations hold:*

(a) [25] **ESMon** $\subset$ **SMon** $\subset$ **CSMon**.
(b) [25] **EWMon** $\subset$ **WMon** $\subset$ **CWMon**.
(c) [25] **EDWMon** $\subset$ **DWMon** $\subset$ **CDWMon**.
(d) [25] **EDSMon** = **DSMon** $\subset$ **CDSMon**.
(e) [31,26] **EMon** $\subset$ **Mon** $\subset$ **CMon**.
(f) **EDMon** $\subset$ **DMon** $\subset$ **CDMon**.

Proof of Theorem 4 in [26] shows that **EDMon** $\subset$ **DMon**. We do not know if anyone has explicitly shown that **DMon** $\subset$ **CDMon**, but it can be shown as follows. Here we do the diagonalization even against **DMon** learners using some (class preserving) r. e. indexing of languages as hypothesis space. Thus, we can consider the hypothesis of the learner as coming from an acceptable programming system, where the learner only conjectures grammars for the languages in the class being learnt. Let $\langle \cdot, \cdot \rangle$ denote some computable bijective coding from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$. Similarly, $\langle \cdot, \cdot, \cdot \rangle$ denotes some computable bijective coding from $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$.

Let $L_i = \{\langle i, 0, x\rangle : x \in \mathbb{N}\}$, $L_{i,j} = \{\langle i, 0, x\rangle : x \leq j\}$, $X_{i,j,k} = L_{i,j} \cup \{\langle i, 1, x\rangle : x \geq k\}$, and $X_{i,j,k,k'} = L_{i,j} \cup \{\langle i, 1, x\rangle : k \leq x \leq k'\} \cup \{\langle i, 2, k\rangle\}$.

Let $M_0, M_1, \ldots$ denote a recursive enumeration of all learning machines. Let T_i be the canonical text for L_i given by $T_i(j) = \langle i, 0, j\rangle$. Let $s_i > 0$ denote the first s found, if any in some standard search, such that $\text{content}(T_i[s]) \cup \{\langle i, 0, s\rangle\} \subseteq W_{M_i(T_i[s])}$. Let r_i denote the time needed to find s_i, if defined (where we assume $r_i \geq s_i$).

Let $\mathcal{L} = \{L_i : i \in \mathbb{N}$ and for all s, $\text{content}(T_i[s]) \cup \{\langle i, 0, s\rangle\} \not\subseteq W_{M_i(T_i[s])}\} \cup \{L_{i,r_i} : i \in \mathbb{N}$ and s_i is defined$\} \cup \{X_{i,s_i-1,r_i} : i \in \mathbb{N}$ and s_i is defined$\} \cup \{X_{i,s_i-1,r_i,k} : i \in \mathbb{N}$ and s_i is defined, $k \in \mathbb{N}\}$.

It is easy to verify that $\mathcal{L}$ is an indexed family. Note that if s_i is defined then L_{i,r_i} is the only language in $\mathcal{L}$ which contains $\langle i, s_i\rangle$. Thus, $\mathcal{L}$ cannot be **DMon**-identified. (If s_i is not defined, then M_i does not **TxtEx**-identify $L_i \in \mathcal{L}$. If s_i is

defined then M_i on input $T_i[s_i]$ has already conjectured a language which omitted every element in $\{\langle i,1,x\rangle : x \in \mathbb{N}\}$. Thus, it will fail to dual monotonically learn $\{X_{i,s_i-1,r_i} : i \in \mathbb{N} \text{ and } s_i \text{ is defined}\} \cup \{X_{i,s_i-1,r_i,k} : i \in \mathbb{N} \text{ and } s_i \text{ is defined}, k \in \mathbb{N}\}$.) On the other hand, with class comprising hypothesis space, one can easily dual monotonically learn the above class: initial conjecture (on input containing elements of form $\langle i,\cdot,\cdot\rangle$) would be $L_i \cup (X_{i,s_i-1,r_i} \cup \{\langle i,2,r_i\rangle\})$ (where $(X_{i,s_i-1,r_i} \cup \{\langle i,2,r_i\rangle\})$ is taken to be empty set, if s_i does not get defined). If and when s_i is defined, the learner waits until it gets $\langle i,0,s_i\rangle$, $\langle i,1,r_i\rangle$, or $\langle i,2,k\rangle$ in the input. At which point it can easily dual monotonically identify the input.

[42] gave some interesting characterization of classes which are (strong, weak) monotonically learnable in dependence of hypothesis space.

Even though **TxtEx** does not depend on whether one chooses class preserving, exact or class comprising hypothesis space, if one considers restricting the number of mind changes to a non-zero value, then learnability does depend on what kind of hypothesis space one chooses. Let $\mathbf{TxtEx}_m$ (see [11]) denote the criterion of learning where the learner is allowed at most m mind changes (here a change from ? to a proper conjecture (member of $\mathbb{N}$) is not counted as a mind change).

Theorem 4. *Suppose $m \geq 1$.*

(a) [27] $\mathbf{ETxtEx}_m \subset \mathbf{TxtEx}_m$.
(b) $\mathbf{TxtEx}_m \subset \mathbf{CTxtEx}_m$.

We do not know if anyone has explicitly shown that $\mathbf{TxtEx}_m \subset \mathbf{CTxtEx}_m$, but it can be shown as follows. Let $M_0, M_1, \ldots$ denote a recursive enumeration of all learning machines. Let $L_i = \{\langle i,x\rangle : x \in \mathbb{N}\}$, and $L^D_{i,j} = \{\langle i,x\rangle : x \leq j \text{ or } x \in D\}$. Let T_i be the canonical text for L_i given by $T_i(j) = \langle i,j\rangle$. Let $s_i > 0$ denote the first s found, if any, such that $M_i(T_i[s])$ outputs a conjecture containing $\langle i,0\rangle$. If s_i gets defined, then let r_i be the number of steps needed to find that s_i is defined (we assume $r_i \geq s_i$). If s_i is defined, then let w_i denote the least $x \geq r_i + 1$, if any, such that $\langle i,x\rangle \in W_{M_i(T_i[s_i])}$. Let r'_i denote the number of steps needed to find w_i, if defined.

If s_i is not defined, then let $\mathcal{L}_i = \{L_i\}$. If s_i is defined, but w_i does not get defined, then let $\mathcal{L}_i = \{L^{\{r_i+1\}}_{i,r_i}\}$. If s_i and w_i both get defined, then let $\mathcal{L}_i = \{L^{\{r_i+1\}}_{i,r_i}\} \cup \{L^D_{i,r_i} : card(D) \leq m+1, w_i + r'_i + 1 = min(D)\}$. Let $\mathcal{L} = \bigcup_i \mathcal{L}_i$.

It is easy to verify that $\mathcal{L}$ is an indexed family, and $\mathcal{L} \in \mathbf{CTxtEx}_m$. However $\mathcal{L} \notin \mathbf{TxtEx}_m$ (using a class preserving hypothesis space), as either s_i is not defined (and thus M_i does not **TxtEx**-identify $L_i \in \mathcal{L}_i$) or the conjecture output by M_i on $T_i[s_i]$ is not for any language in $\{L^D_{i,r_i} : card(D) \leq m+1, w_i + r'_i + 1 = min(D)\}$, which forces at least $m+1$ mind changes to learn $\{L^D_{i,r_i} : card(D) \leq m+1, w_i + r'_i + 1 = min(D)\}$.

If one considers iterative learning (where the learner's hypotheses depend only on its last hypothesis and current data, rather than all the data it has seen so far) [37,28], then [28] showed that for certain classes and particular class comprising hypothesis spaces iterative learning may outperform conservative

learning, though in general iterative learning is contained in conservative learning (for class comprising hypothesis spaces).

[31] also studied how the structure of relationship between various versions of monotonicity/dual monotonicity changes if one considers class comprising hypothesis space as opposed to class preserving/exact hypothesis spaces. For example, **CTxtFin** $\subset$ **CDMon**, though **TxtFin** $=$ **DMon**. Similarly,

- **CDWMon** $=$ **CTxtEx**, though **DWMon** $\subset$ **TxtEx**,
- and **DSMon** $\subset$ **SMon**, though **CDSMon** and **CSMon** are incomparable.

Thus, not only do the classes learnable (under a learning criterion) depend on the kind of hypothesis spaces that are allowed, but even the relationship among the learning criteria depend on what kind of hypothesis spaces are allowed.

In [29] the authors study set driven and rearrangement independent learning in dependence of hypothesis space for indexed families (hypothesis space being indexed family too). They showed that for set driven and rearrangement independent learning, the classes that can be finitely learnt does not depend on the type of hypothesis spaces allowed (among the types, exact, class preserving and class comprising).

Theorem 5. [29]
r-**ETxtFin** $=$ s-**ETxtFin** $=$ **ETxtFin** $=$ **TxtFin** $=$ **CTxtFin**.

For explanatory learning, set driven learning forms a hierarchy depending on the type of hypothesis space allowed, whereas for rearrangement independent learning, it does not depend on the type of hypothesis space allowed.

Theorem 6. [29]

(a) s-**ETxtEx** $\subset$ s-**TxtEx** $\subset$ s-**CTxtEx** $\subset$ **ETxtEx** $=$ **TxtEx** $=$ **CTxtEx**.
(b) r-**ETxtEx** $=$ r-**TxtEx** $=$ r-**CTxtEx** $=$ **ETxtEx** $=$ **TxtEx** $=$ **CTxtEx**.

For monotonic learning (all three types) we get a proper hierarchy for both set driven as well as rearrangement independent learning.

Theorem 7. [29]

(a) s-**ESMon** $\subset$ s-**SMon** $\subset$ s-**CSMon**.
(b) s-**EMon** $\subset$ s-**Mon** $\subset$ s-**CMon**.
(c) s-**EWMon** $\subset$ s-**WMon** $\subset$ s-**CWMon**.
(d) r-**ESMon** $\subset$ r-**SMon** $\subset$ r-**CSMon**.
(e) r-**EMon** $\subset$ r-**Mon** $\subset$ r-**CMon**.
(f) r-**EWMon** $\subset$ r-**WMon** $\subset$ r-**CWMon**.

We refer the reader to [41] for several other results and characterizations for learning indexed family in dependence on hypothesis spaces.

[30] considered the situation where the learner may make queries regarding certain kind of relationship between a potential hypothesis and the input language. The queries allowed are subset, superset or disjointness queries. The learner, after making a finite number of such queries, outputs a single hypothesis

which must be correct for languages in the class being learnt. They showed that the learnability of a class depends very much on whether the hypothesis space (query space) chosen is an indexed family, recursively enumerable (r.e.) family or a limiting r. e. family. [18] extended above work to learning r.e. classes of r.e. languages.

4 Special Hypotheses Spaces

In this section we revert back to learning recursively enumerable languages with respect to some fixed hypothesis spaces. Criteria **I** (such as **TxtEx**, **TxtBc**, **TxtFin**, or **TxtFex**) without a specified hypothesis space refers to using acceptable numbering as hypothesis space.

A universal numbering is a numbering which contains an index for every recursively enumerable set. Friedberg numberings [13] are universal numberings in which every r.e. language has exactly one index. Friedberg numberings can in some sense be considered "efficient" as they do not have any redundancy. **Ke**-numberings [21] are universal numberings for which grammar equivalence problem for indices is limiting recursive.

Jain and Stephan [21] considered learning using Friedberg numberings or **Ke**-numberings as hypothesis spaces. ([12] considered learning of functions using Friedberg numberings as hypothesis spaces). For a criteria **I** of learning, let **FrI** (**KeI**) denote the class of languages which can be learnt under the criteria **I** using some Friedberg numbering (some **Ke**-numbering) as hypothesis space. [21] showed that every **TxtEx**-learnable class can be learnt using some Friedberg numbering as hypothesis space. However, no single Friedberg numbering is enough to be used as hypothesis space for all **TxtEx**-learnable classes. On the other hand, for finite learning, there are classes of languages which can be finitely learnt (using acceptable numbering as hypothesis spaces), but which cannot be learnt using any Friedberg numbering as hypothesis space. On the other hand, every finitely learnable class can be learnt using some **Ke**-numbering as hypothesis space.

Theorem 8. [21] $\mathbf{FrTxtEx} = \mathbf{KeTxtEx} = \mathbf{TxtEx}$.
$\mathbf{FrTxtFin} \subset \mathbf{KeTxtFin} = \mathbf{TxtFin}$.

An interesting result shown by [21] is that a recursively enumerable class can be finitely learnt using some Friedberg numbering as hypothesis space iff it is 1–1 recursively enumerable and finitely learnable.

On the other hand, the situation changes for vacillatory and behaviourally correct learning. There exist behaviourally correctly learnable classes which cannot be behaviourally correctly learnt in any Friedberg numbering.

Theorem 9. [21] $\mathbf{FrTxtBc} \subset \mathbf{KeTxtBc}$.

It is open at this point whether every behaviourally correctly learnable class is behaviourally correctly learnable in some **Ke** numbering.

On the other hand, for vacillatory learning, there are vacillatorily learnable classes which cannot be vacillatorily learnt in any **Ke**-numbering. In particular, every class which can be vacillatorily learnt in some **Ke**-numbering is explanatorily learnable!

Theorem 10. [21] **FrTxtFex** = **KeTxtFex** = **TxtEx** ⊂ **TxtFex**.

(Here **TxtEx** ⊂ **TxtFex** was shown by [8]).

Even though every **TxtEx**-learnable class is learnable using some Friedberg numbering, the learner may not satisfy some desirable properties. For example, consider non-U-shaped learning. Non-U-shaped learning requires that a learner never abandons a correct hypotheses [7]. Every **TxtEx**-learnable class can also be learnt in a non-U-shaped way using some acceptable numbering as hypothesis space [7]. However, even some simple classes, such as the class of all finite sets, cannot be learnt in non-U-shaped way using any Friedberg numbering as hypothesis space. (However, one can do non-U-shaped learning of every **TxtEx**-learnable class using some **Ke**-numbering as hypothesis space.) Similar results hold if one considers conservative, monotonic or prudent learning (where in prudent learning [33], the learner is allowed to output conjectures only for languages which it learns).

However, for consistent learning [1,4] one can use Friedberg numbering as hypothesis space for every class of languages which can be consistently learnt using some hypothesis space.

In contrast to the power of using Friedberg numberings in general, there are some Friedberg numberings which make learning almost impossible: only **TxtEx**-learnable classes which contain finitely many infinite languages could be (explanatorily, behaviourally correctly, or vacillatorily) learnt using such Friedberg numberings as hypothesis space. Similarly, there exist Friedberg numberings, using which as hypothesis space, only inclusion free finite classes of languages can be finitely learnt (a class is inclusion free if no language in the class is included in another language in the class). We refer the reader to [21] for further results on learning using a Friedberg or **Ke**-numbering as hypothesis spaces.

5 Prescribed Learning

Until now we have been mostly concentrating on learning using some *suitable* hypothesis space, perhaps with some constraints such as being class preserving, class comprising or being a Friedberg numbering. What if one requires that the learning has to happen with respect to *every* suitable hypothesis space? This kind of situation is useful if one expects that the seller provides a learner which works based on the programming system used by any potential buyer, rather than only with the programming system used by the seller. Here one may distinguish between two cases, one where there exists a learner for each of the suitable hypothesis spaces and where one expects the same learner (with hypothesis space being a parameter) to work for all hypothesis spaces. The issue here is of being able to effectively generate a learner given a description of the suitable hypothesis space. Jain, Stephan and Ye [24,23] considered the above situation.

We say that a class $\mathcal{L}$ is *prescribed* **I** learnable, if for every hypothesis space $\mathcal{H} \supseteq \mathcal{L}$, $\mathcal{L}$ can be learnt according to the criterion **I** using $\mathcal{H}$ as hypothesis space.

We say that a class $\mathcal{L}$ is *class-preserving-prescribed* **I** learnable, if for every class preserving hypothesis space $\mathcal{H}$ (that is $\mathcal{H} = \mathcal{L}$, set wise; the indexing may be different), $\mathcal{L}$ can be learnt according to criterion **I** using $\mathcal{H}$ as hypothesis space.

We say that $\mathcal{L}$ is *uniformly* **I** learnable, if there exists an effective listing $M_0, M_1, M_2, \ldots$ of learners such that given a program i describing the hypothesis space $\mathcal{H} \supseteq \mathcal{L}$, $\mathcal{L}$ can be learnt by M_i according to criterion **I** using $\mathcal{H}$ as hypothesis space. Here we say that the program i describes the hypothesis space $\mathcal{H}$ if $\varphi_i(j, x) = 1$ iff $x \in H_j$ (where, when considering indexed family as hypothesis space, we require φ_i to be total). In above, φ_i denotes the function computed by the i-th program in some standard acceptable programming system.

One can define *uniformly class-preserving* learning similarly.

For general learnability of r.e. languages, where hypothesis spaces are r.e. classes (rather than indexed families) prescribed learning is quite weak as in some Friedberg numberings only restricted classes can be learnt. Thus, for r.e. languages one normally considers class-preservingly-prescribed (uniformly class-preserving) learning only. Note that the concept class being considered here would be r.e. classes of r.e. languages.

For finite and explanatory learning, uniform learning can very much be done.

Theorem 11. [23] *Every* **TxtFin***-learnable r.e. class of languages is also uniformly class-preservingly* **TxtFin** *learnable.*

Theorem 12. [23] *Every* **TxtEx***-learnable r.e. class of languages is also uniformly class-preservingly* **TxtEx***-learnable.*

For confident learning, there are classes which are class comprisingly confidently learnable but not class preservingly confidently learnable. So we have a restricted version of the above theorems for confident learning.

Theorem 13. [23] *Every class-preservingly confident learnable r.e. class of languages is also uniformly class-preservingly confidently learnable.*

[23] also consider behaviourally correct learning and vacillatory learning. Though these criteria are similar to explanatory learning (in semantic sense), it was shown that there are classes behaviourally correctly learnable using class-preserving hypothesis spaces but not class-preservingly-prescribed behaviourally correctly learnable. It is open at this point whether uniform and non-uniform prescribed version of class-preserving learning for behaviourally correct learning are same. Similar question for vacillatory learning is also open.

On the other hand, for conservative learning, prescribed and uniform learning are a restriction and are separated from each other.

Theorem 14. [23]

(a) The class $\{D : |D| \leq 1\}$ *is class-preservingly-prescribed conservatively but not uniformly class-preservingly conservatively learnable.*

(b) The class $\{D : |D| < \infty\}$ is class-preservingly conservatively but not class-preservingly-prescribed conservatively learnable.
(c) The class $\{D : |D| = 2 \vee (|D| = 1 \wedge D \subseteq K')\}$ is class-comprisingly conservatively but not class-preservingly conservatively learnable.

We now turn our attention to prescribed learning of indexed families. Rest of the section considers learning of indexed families only. Thus, as in Section 3, the hypothesis spaces are assumed to be indexed families.

For finite learning, prescribed and uniform learning are very restricted. However, uniform class-preserving learning can be done for all finitely learnable indexed families.

Theorem 15. [24] *Suppose $\mathcal{L}$ is a finitely learnable indexed family.*

(a) Any non-empty $\mathcal{L}$ is not uniformly finitely learnable.
(b) $\mathcal{L}$ is uniformly class-preservingly finitely learnable.
(c) $\mathcal{L}$ is prescribed finitely learnable iff the class is finite and inclusion free (that is, any two distinct languages in the class are incomparable by $\subseteq$).

For conservative learning, uniform and prescribed learning imply that (almost) all the languages in the class are cofinite.

Theorem 16. [24]

(a) If $\mathcal{L}$ is uniformly conservatively learnable then every $L \in \mathcal{L}$ is cofinite.
(b) If $\mathcal{L}$ is prescribed conservatively learnable then all but finitely many languages $L \in \mathcal{L}$ are cofinite.

Furthermore, uniformly class-preserving conservative learning and prescribed conservative learning are incomparable.

Theorem 17. [24]

(a) There exists a class $\mathcal{L}$ which is uniformly class-preservingly conservatively learnable, but not prescribed conservatively learnable.
(b) There exists a class $\mathcal{L}$ which is prescribed conservatively learnable but not uniformly class-preservingly conservatively learnable.

We now consider the effect of prescribing the hypothesis space for monotonic learning.

Theorem 18. [24]

(a) Any non-empty $\mathcal{L}$ is not uniformly strong-monotonically learnable.
(b) $\mathcal{L}$ is prescribed strong-monotonically learnable iff $\mathcal{L}$ is finite.

On the other hand, the class $\mathcal{L} = \{L_i : i \in \mathbb{N}\}$, where $L_i = \{i\}$, is uniformly monotonically learnable.

In contrast to conservative learning, for monotonic learning, uniform learnability implies that the languages in the class are finite.

Theorem 19. [24]

(a) If $\mathcal{L}$ is uniformly monotonically learnable, then $\mathcal{L}$ contains only finite sets.
(b) If $\mathcal{L}$ is prescribed monotonically learnable, then $\mathcal{L}$ contains only finitely many infinite sets.

Theorem 20. [24]

(a) There exists a class $\mathcal{L}$ which is uniformly class-preservingly strong-monotonically learnable but not prescribed monotonically learnable.
(b) There exists a class $\mathcal{L}$ which is prescribed monotonically learnable but not uniformly class-preservingly monotonically learnable.
(c) Every prescribed strong-monotonically learnable class is also uniformly class-preservingly strong-monotonically learnable.

6 Optimal Hypotheses Spaces

As we have seen, chosen hypothesis spaces play a crucial role in whether a learner is able to learn the target class of languages. In this section we consider whether there are hypothesis spaces $\mathcal{H}$ which are optimal, in the sense that any class learnable using any hypothesis space is also learnable using the hypothesis space $\mathcal{H}$. This ofcourse depends on the criterion under investigation. Furthermore, we consider whether an hypothesis space being optimal for a particular criterion implies it being optimal for some other criterion. Such studies were done by [22].

For a criteria of learning **I**, an hypothesis space $\mathcal{H}$ is said to be *optimal* if any class $\mathcal{L}$, **I**-learnable using some hypothesis space, is also **I** learnable using $\mathcal{H}$ as hypothesis space. The hypothesis space $\mathcal{H}$ is said to be *effectively optimal* for a criteria **I** if given any learner M using hypothesis space $\mathcal{H}'$, one can effectively find a learner M' using $\mathcal{H}$ as hypothesis space (for the class of languages which was **I**-learnt by M using $\mathcal{H}'$ as hypothesis space).

Clearly, all acceptable numberings are optimal. Are there other optimal numberings?

Definition 21. [22] A numbering $A_0, A_1, A_2, \ldots$ is called *nearly acceptable* iff there is a recursive function f such that $A_{f(d,e)} = W_e$ whenever $d \in W_e$.

The nearly optimal numberings are effectively optimal for explanatory, vacillatory and behaviourally correct learning. They are also optimal for finite learning, but not necessarily effectively optimal for finite learning. Note that one can easily construct nearly acceptable numberings which are not acceptable.

The effectively optimal numberings for finite, explanatory and vacillatory learning are easy to characterize. A numbering ψ is said to be K-acceptable [9] iff for any further numbering η, there exists a limiting recursive compiler [25] translating the indices of η to indices of ψ.

Theorem 22. [22] *A hypothesis space $\mathcal{H} = H_0, H_1, H_2, \ldots$ of all r.e. sets is*

(a) effectively optimal for finite learning iff $\mathcal{H}$ is acceptable;
(b) effectively optimal for explanatory learning iff $\mathcal{H}$ is K-acceptable;

(c) effectively optimal for vacillatory learning iff there is a limiting-recursive function g such that, for all d, there is an $e \leq g(d)$ with $H_e = W_d$.

The following theorem gives the relation between optimal numberings for finite, explanatory, behaviourally correct and vacillatory learning.

Theorem 23. [22]
(a) For each $\mathbf{I} \in \{\mathbf{TxtEx}, \mathbf{TxtFin}, \mathbf{TxtBc}, \mathbf{TxtFex}\}$*, there are numberings which are optimal but not effectively optimal for* $\mathbf{I}$.
(b) For any two distinct $\mathbf{I}$ *and* $\mathbf{J}$ *in* $\{\mathbf{TxtEx}, \mathbf{TxtFin}, \mathbf{TxtBc}, \mathbf{TxtFex}\}$*, there is a numbering which is optimal for* $\mathbf{I}$ *but not optimal for* $\mathbf{J}$.

In (b) above, if $\mathbf{I} \neq \mathbf{TxtFin}$ and ($\mathbf{I} \neq \mathbf{TxtEx}$ or $\mathbf{J} \neq \mathbf{TxtFex}$), then we can even take the numbering to be effective optimal for $\mathbf{I}$.

Another interesting result is that every (effectively) optimal numbering for **TxtEx** is also (effectively) optimal for consistent learning. On the other hand there are numberings which are effectively optimal for consistent learning but not optimal for finite, explanatory, vacillatory or behaviourally correct learning.

In learning with additional information, in addition to text for the language, learner is also provided with an upper bound on a grammar (in the hypothesis space) for the target language [15,20]. [22] showed that the **Ke**-numberings are exactly those hypothesis spaces which are optimal for learning with additional information.

Acknowledgements. We thank Frank Stephan for helpful discussions and 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., Smith, C.: Inductive inference: Theory and methods. Computing Surveys 15, 237–289 (1983)
4. Bārzdiņš, J.: Inductive inference of automata, functions and programs. In: Proceedings of the 20th International Congress of Mathematicians, Vancouver, pp. 455–560 (1974); (in Russian) English translation in American Mathematical Society Translations. Series 2 109, 107–112 (1977)
5. Bārzdiņš, J.: Two theorems on the limiting synthesis of functions. Theory of Algorithms and Programs 1, 82–88 (1974) (in Russian)
6. Blum, L., Blum, M.: Toward a mathematical theory of inductive inference. Information and Control 28, 125–155 (1975)
7. Baliga, G., Case, J., Merkle, W., Stephan, F., Wiehagen, R.: When unlearning helps. Information and Computation 206(5), 694–709 (2008)
8. Case, J.: The power of vacillation in language learning. SIAM Journal on Computing 28(6), 1941–1969 (1999)
9. Case, J., Jain, S., Suraj, M.: Control structures in hypothesis spaces: The influence on learning. Theoretical Computer Science 270(1–2), 287–308 (2002)

10. Case, J., Lynes, C.: Machine inductive inference and language identification. In: Nielsen, M., Schmidt, E.M. (eds.) ICALP 1982. LNCS, vol. 140, pp. 107–115. Springer, Heidelberg (1982)
11. Case, V., Smith, C.: Comparison of identification criteria for machine inductive inference. Theoretical Computer Science 25, 193–220 (1983)
12. Freivalds, R., Kinber, E., Wiehagen, R.: Inductive inference and computable one-one numberings. Zeitschr. f. math. Logik und Grundlagen d. Math. Bd. 28, 463–479 (1982)
13. Friedberg, R.: Three theorems on recursive enumeration. Journal of Symbolic Logic 23(3), 309–316 (1958)
14. Fulk, M.: Prudence and other conditions on formal language learning. Information and Computation 85, 1–11 (1990)
15. Freivalds, R., Wiehagen, R.: Inductive inference with additional information. Journal of Information Processing and Cybernetics (EIK) 15, 179–195 (1979)
16. Gold, E.M.: Language identification in the limit. Information and Control 10, 447–474 (1967)
17. Jantke, K.: Monotonic and non-monotonic inductive inference. New Generation Computing 8, 349–360 (1991)
18. Jain, S., Lange, S., Zilles, S.: A general comparision of language learning from examples and from queries. Theoretical Computer Science A 387(1), 51–66 (2007); Special Issue on Algorithmic Learning Theory (2005)
19. Jain, S., Osherson, D., Royer, J., Sharma, A.: Systems that Learn: An Introduction to Learning Theory, 2nd edn. MIT Press, Cambridge (1999)
20. Jain, S., Sharma, A.: Learning with the knowledge of an upper bound on program size. Information and Computation 102, 118–166 (1993)
21. Jain, S., Stephan, F.: Learning in Friedberg numberings. Information and Computation 206(6), 776–790 (2008)
22. Jain, S., Stephan, F.: Numberings optimal for learning. In: Györfi, L., Freund, Y., Turán, G., Zeugmann, T. (eds.) ALT 2008. LNCS (LNAI), vol. 5254, pp. 434–448. Springer, Heidelberg (2008)
23. Jain, S., Stephan, F., Ye, N.: Prescribed learning of R.E. classes. In: Hutter, M., Servedio, R., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 64–78. Springer, Heidelberg (2007)
24. Jain, S., Stephan, F., Ye, N.: Prescribed learning of indexed families. Fundamenta Informaticae 83(1–2), 159–175 (2008)
25. Lange, S., Zeugmann, T.: Language learning in dependence on the space of hypotheses. In: Proceedings of the Sixth Annual Conference on Computational Learning Theory, pp. 127–136. ACM Press, New York (1993)
26. Lange, S., Zeugmann, T.: The learnability of recursive languages in dependence on the space of hypotheses. Technical Report 20/93, GOSLER-Report, FB Mathematik und Informatik, TH Lepzig (1993)
27. Lange, S., Zeugmann, T.: Learning recursive languages with bounded mind changes. International Journal of Foundations of Computer Science 4, 157–178 (1993)
28. Lange, S., Zeugmann, T.: Incremental learning from positive data. Journal of Computer and System Sciences 53, 88–103 (1996)
29. Lange, S., Zeugmann, T.: Set-driven and rearrangement-independent learning of recursive languages. Mathematical Systems Theory 29, 599–634 (1996)
30. Lange, S., Zilles, S.: Comparison of query learning and gold-style learning in dependence of the hypothesis space. In: Ben-David, S., Case, J., Maruoka, A. (eds.) ALT 2004. LNCS (LNAI), vol. 3244, pp. 99–113. Springer, Heidelberg (2004)

31. Lange, S., Zeugmann, T., Kapur, S.: Monotonic and dual monotonic language learning. Theoretical Computer Science A 155, 365–410 (1996)
32. Osherson, D., Stob, M., Weinstein, S.: Learning strategies. Information and Control 53, 32–51 (1982)
33. Osherson, D., Stob, M., Weinstein, S.: Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists. MIT Press, Cambridge (1986)
34. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967); reprinted by MIT Press (1987)
35. Schäfer-Richter, G.: Über Eingabeabhängigkeit und Komplexität von Inferenzstrategien. PhD thesis, RWTH Aachen (1984)
36. Wexler, K., Culicover, P.: Formal Principles of Language Acquisition. MIT Press, Cambridge (1980)
37. Wiehagen, R.: Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. Journal of Information Processing and Cybernetics (EIK) 12, 93–99 (1976)
38. Wiehagen, R.: Zur Theorie der Algorithmischen Erkennung. Dissertation B, Humboldt University of Berlin (1978)
39. Wiehagen, R.: A thesis in inductive inference. In: Dix, J., Jantke, K., Schmitt, P. (eds.) NIL 1990. LNCS (LNAI), vol. 543, pp. 184–207. Springer, Heidelberg (1991)
40. Wiehagen, R., Zeugmann, T.: Learning and consistency. In: Jantke, K.P., Lange, S. (eds.) GOSLER 1994. LNCS (LNAI), vol. 961, pp. 1–24. Springer, Heidelberg (1995)
41. Zeugmann, T., Lange, S.: A guided tour across the boundaries of learning recursive languages. In: Jantke, K., Lange, S. (eds.) GOSLER 1994. LNCS (LNAI), vol. 961, pp. 190–258. Springer, Heidelberg (1995)
42. Zeugmann, T., Lange, S., Kapur, S.: Characterizations of monotonic and dual monotonic language learning. Information and Computation 120, 155–173 (1995)
43. Zeugmann, T., Zilles, S.: Learning recursive functions: A survey. Theoretical Computer Science A 397(1–3), 4–56 (2008); Special Issue on Forty Years of Inductive Inference. Dedicated to the 60th Birthday of Rolf Wiehagen

State Complexity of Nested Word Automata*

Kai Salomaa

School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Abstract. We discuss techniques to establish lower bounds for the number of states of finite automata operating on nested words. To illustrate these methods we establish a lower bound for the state complexity of Kleene star that is of a different order than the known tight state complexity bound for the star of ordinary regular languages. We survey known bounds on deterministic and nondeterministic state complexity of basic operations on regular nested word languages and discuss open problems.

1 Introduction

Finite automata operating on nested words have been introduced by Alur and Madhusudan [4,5] as a natural extension of ordinary finite automata. The symbols of a nested word are ordered in the standard way linearly, and additionally there is a recursively defined matching of occurrences of special call and return symbols. Automata on nested words provide a natural computational model for applications like XML document processing, where data has a dual linear-hierarchical structure. The class of regular nested word languages retains many desrirable properties of classical regular languages, in particular, closure and decision properties. More information on applications of nested word automata and on related models can be found e.g. in [1,2,6].

The state complexity of basic operations on regular languages has been extensively studied, see for example [15,17,19,21,22,29,31]. Generally when considering various extensions of finite automata, naturally one of the first questions to be answered could be which properties turn out to be significantly different for the extended model. Tree automata [13] are another much studied model extending ordinary finite automata. As regards Boolean operations or the extension of concatenation and Kleene star to trees, it can be expected that the state complexity results for tree automata[1] would be essentially similar as for ordinary finite automata. Interestingly, for nested word automata the lower bounds for deterministic state complexity of, for example, catenation and Kleene star turn out to be of a different order than the corresponding results for ordinary regular

* Research supported by the Natural Sciences and Engineering Research Council of Canada.

[1] However, the author is not aware of work explicitly discussing the state complexity of operations on regular tree languages.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 59–70, 2009.

languages. It can be noted that (ranked or unranked) trees can be viewed as special cases of nested words [5]. Recently, it has been established in [12] that nested word automata are equivalent to pushdown forest automata [25].

It is shown in [5] that a deterministic nested word automaton equivalent to a given nondeterministic automaton with $O(n)$ states may need 2^{n^2} states. The fact that the state complexity blow-up is larger than in the case of ordinary finite automata, intuitively, causes the possibility that the known upper bounds for the state complexity of basic operations on regular languages [29] need not, in general, hold for languages of nested words.

Even in the case of ordinary deterministic finite automata (DFA), worst-case state complexity examples for various operations are often difficult to construct and software tools, such as Grail+, are used for verifying worst-case examples [11,28,30]. The fact that DFAs can be minimized efficiently makes it possible to use software tools to find worst-case examples. The situation is essentially different for nested word automata because a minimal deterministic automaton need not be unique [3,5,24] and there is no known efficient minimization algorithm, even for deterministic nested word automata. Basic operations and determinization of visibly pushdown automata (that are essentially similar to nested word automata) have been implemented in [26]. However, the usability of the tools for verifying worst-case state complexity examples is limited due to the lack of an efficient minimization algorithm.

In order to get a better quantitative understanding of regular languages of nested words, the study of state complexity of basic regularity preserving operations has been initiated in [18,24]. Here in Section 4 we will summarize the known state complexity results for deterministic and nondeterministic nested word automata. Many questions remain open. In particular, for the deterministic state complexity of concatenation, Kleene star and reversal the known upper and lower bounds remain far apart, and the same holds for the nondeterministic state complexity of complementation.

We will discuss general techniques to establish lower bounds for the number of states of a (non)-deterministic nested word automaton. The techniques are inspired by the fooling set methods [7,14,20] used for ordinary nondeterministic finite automata (NFA). We will establish a new lower bound for the state complexity of Kleene star of regular nested word languages that is of a different order than the worst-case state complexity of Kleene star of ordinary DFAs.

2 Finite Automata on Nested Words

We assume that the reader is familiar with with finite automata and state complexity, see [27,29,31]. Here we briefly recall the definitions associated with automata on nested words. More details on nested words and their applications can be found in [4,5].

In the following Σ always denotes a finite alphabet. The *tagged alphabet* corresponding to 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 word* over Σ is a sequence of symbols of $\hat{\Sigma}$, $w = u_1 \cdots u_m$, $u_i \in \hat{\Sigma}$, $i = 1, ..., m$. We define recursively a hierarchical matching relation in a tagged word. For w as above, a call symbol $u_i \in \langle\Sigma$ *matches* a return symbol $u_j \in \Sigma\rangle$, $i < j$, if in the subsequence $u_{i+1} \cdots u_{j-1}$ every call symbol (respectively, return symbol) has a matching return symbol (respectively, call symbol). Symbol occurrences $u_i \in \langle\Sigma$ that do not have a matching return, $1 \leq i \leq m$, are *pending calls*, and $u_i \in \Sigma\rangle$ 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 word.

By a *nested word* we mean a tagged word that is associated with the usual linear ordering of symbols and the hierarchical matching relation between occurrences of call and return symbols. 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. 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, $\langle d\rangle$ matches return symbol $d\rangle$ (respectively, $c\rangle$), both occurences of $b\rangle$ are pending returns and $\langle b$ is a pending call. The word is not well-matched since it has pending calls and/or returns.

Language operations such as concatenation, Kleene star and reversal are extended in the natural way for sets of nested words. For example, the catenation of nested words w_1 and w_2 is the uniquely defined nested word where the underlying tagged word is the catenation of the tagged words corresponding to w_1 and w_2. Note that in the catenation pending calls of w_1 may match return symbol occurrences in w_2. When reversing a nested word, return symbols become call symbols and vice versa, for more details see [5]. The number of symbols $a \in \hat{\Sigma}$ occurring in a nested word w is denoted $|w|_a$.

We recall the definition of nested word automata from [5]. 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. The distinction will be useful for obtaining precise bounds for state complexity. A simplified definition of nested word automata was used in [4].

Definition 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.*

For defining the computations of an NNWA it is convenient to view a nested word $u_1 \cdots u_m$ as a directed graph where there is a *linear edge* from u_i to u_{i+1}, $i = 1, \ldots, m-1$, and additionally each pair of matching call and return symbols is connected by a *hierarchical edge*.

An NNWA 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 is a tuple $(p_1, \ldots, p_k, q)$, where $p_i \in P$, $i = 1, \ldots, k$, $k \geq 0$, 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. Note that the frontier of the computation completely determines how the computation can be continued on the remainder of the input. The NNWA A accepts a nested word w if in some nondeterministic computation it reaches the end of w in a final linear state and all hierarchical states of the computation corresponding to pending calls are final, that is, the frontier at the end of the computation is of the form $(p_1, \ldots, p_k, q)$, $q \in Q_f$, $p_i \in P_f$, $i = 1, \ldots, k$, $k \geq 0$. The nested word language recognized by A is denoted $L(A)$. Two NNWAs are said to be *equivalent* if they recognize the same language. A nested word language is *regular* if it is recognized by an NNWA.

A nested word automaton A as given in Definition 1 is said to be *deterministic* (a DNWA) if 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 that a DNWA is incomplete, that is, some values of the transition functions may be undefined. Any DNWA as can be "completed", 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. The following result from [5], see also [4], gives an upper bound for the size blow-up of determinizing an NNWA. The upper bound is tight within a multiplicative constant.

Proposition 1. *[5] 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, $n \geq 1$, recognized by an NNWA with $O(n)$ states such that any DNWA for L_n needs $\Omega(2^{n^2})$ states.*

We define the *deterministic* (respectively, *nondeterministic*) *state complexity* of a regular nested word language L, denoted $\mathrm{sc}(L)$ (respectively, $\mathrm{nsc}(L)$), as the smallest total number of states (linear states and hierarchical states) of any DNWA (respectively, NNWA) recognizing L. Naturally, a more complete descriptional complexity measure for nondeterministic automata would include

the number of transitions [9,10,17], however, here we do not consider transition complexity of NNWAs.

It should be noted that the roles played by the (numbers of) linear and hierarchical states, respectively, are different and often state complexity bounds are formulated separately for the numbers of linear and of hierarchical states. The combined value, $\mathrm{sc}(L)$ or $\mathrm{nsc}(L)$, is a convenient approximation of the descriptional complexity of L, especially in cases where precise upper and lower bounds are not known.

3 State Complexity Lower Bounds

The closure properties of regular languages of nested words under operations such as concatenation, Kleene star and reversal are established using nondeterministic nested word automata [5]. As indicated by Proposition 1, a DNWA equivalent to an n state NNWA may require more than 2^n states and, consequently, it is not clear whether the known upper bounds for the state complexity of basic operations on regular languages [29] hold for languages of nested words. The answer seems to depend on particular properties of each operation.

The regularity of nested word languages can be characterized by the finiteness of suitably defined congruence relations that extend the Myhill-Nerode right congruence to nested words, however, the number of congruence classes does not, in general, give the number of states of a minimal DNWA and the minimal DNWA need not be unique [3,5]. Note that [3] deals with *visibly pushdown automata* (VPA) and a state-minimal VPA corresponds to a DNWA that is minimized only with respect to the number of linear states. However, using essentially the same idea one can construct a nested word language that has two non-isomorphic DNWAs with a minimal total number of states.

We can develop lower bound techniques for the number of states of a (non)-deterministic nested word automaton that are inspired by the fooling set method useds for NFAs [7,14,20]. In the case of DNWAs, instead of a set of pairs of words we can use a set of nested words where each pair can be separated by a suitably chosen suffix.

Definition 2. *Let L be a nested word language over Σ.*

(i) *A finite set $S \subseteq \mathrm{NW}(\Sigma)$ is a* 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 set of pairs of nested words $F = \{(x_i, y_i) \mid i = 1, \ldots, m\}$ is said to be a* fooling set *for L if:*
   (iia) *$x_i y_i \in L$, $i = 1, \ldots, m$, and*
   (iib) *for any $1 \leq i < j \leq m$, $x_i y_j \notin L$ or $x_j y_i \notin L$.*

The set S is a k-separator set, $k \geq 0$, if each word in S has exactly k pending calls. The set F is a k-fooling set, $k \geq 0$, if each x_i has exactly k pending call symbols.

Lemma 1. [18,24] *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 $|P|^k \cdot |Q| \geq |S|$.*
(ii) *If $L(A)$ has a k-fooling set F, then $|P|^k \cdot |Q| \geq |F|$.*

The use of Lemma 1 is restricted by the requirement that all words corresponding to a k-separator or k-fooling set must have the same number of pending calls. A modified lower bound technique involving additional conditions is used in [24] to establish a tight state complexity lower bound for union and complementation.

Already in the case of ordinary regular languages it is known that the lower bounds obtained by the fooling set methods may be far removed from the actual nondeterministic state complexity of the language [16], and the same limitation naturally carries over to NNWAs. It was left open in [24] how good estimates, in the worst case, the k-separator sets provide for the size of DNWAs.

To illustrate the use of Lemma 1 we establish a lower bound for the state complexity of Kleene star that is of a different order than the worst-case state complexity of star of ordinary regular languages. (The state complexity of the Kleene star operation for nested word languages was not considered in [24].) It is well known that the state complexity of the Kleene star for DFAs is $2^{n-1} + 2^{n-2}$ [29]. We construct nested word languages that give a lower bound of $2^{\Omega(n \cdot \log n)}$ for the state complexity of star.

For $w \in \{0,1\}^*$ we denote by $\mathrm{num}(w)$ the number represented by the binary word w. Note that w may contain leading zeros. Let $\Sigma = \{0,1,a,b\}$ and let $n \in \mathbb{N}$ be arbitrary but fixed. We define

$$L_n = \{0,1,a\}^* \cup L'_n \tag{1}$$

where

$$\begin{aligned} L'_n = \{ w_1 a u_1 \langle b u_2 w_2 a u_3 b w_2 b \rangle w_1 \ \mid\ & w_1, w_2 \in \{0,1\}^n,\ u_i \in \{0,1,a\}^*, \\ & i \in \{1,3\},\ u_2 \in \{0,1,a\}^* a \cup \{\varepsilon\},\ |u_2|_a = \mathrm{num}(w_1)\ \}. \end{aligned} \tag{2}$$

Lemma 2. *The state complexity of L_n is $O(2^n)$.*

Proof. We describe the construction of a DNWA A' for the language L'_n. Afterwards we describe how the construction can be modified to obtain a DNWA for L_n.

The following description assumes that the input for A' is of the form $z = v_1 \langle b v_2 b v_3 b \rangle v_4$, $v_i \in \{0,1,a\}^*$, $i = 1, \ldots, 4$. If this is not the case, A' can easily be made to reject without exceeding the bound on the number of states. On an input z, A' checks that the prefix of z of length $n+1$ is of the form

$$w_1 a, \quad w_1 \in \{0,1\}^n. \tag{3}$$

If the prefix is not of this form, A' rejects. When reading the prefix A' stores w_1 in the linear state, and after this the computation passes by symbols of $\{0,1,a\}$ until it reaches a call symbol $\langle b$. From the call symbol the computation continues

in a linear state q_{w_1} and hierarchical state p_{w_1} that both encode the binary string w_1 of length n.

The linear computation beginning in state q_{w_1} reads symbols of $\{0, 1, a\}$ and counts $\mathrm{num}(w_1)$ occurrences of a. After the $\mathrm{num}(w_1)$'th occurrence of a, the automaton checks that the following subword is of the form $w_2 a$, $w_2 \in \{0,1\}^n$, and stores w_2 in the linear state. The computation passes by symbols of $\{0, 1, a\}$ until it encounters a b. Now the computation checks that the following n symbols equal w_2 and are followed by a return symbol $b\rangle$. Finally, the last stage of the computation verifies that the suffix following $b\rangle$ is equal to the word w_1 that is encoded by the hierarchical state p_{w_1} received at the return symbol $b\rangle$.

The linear computation of A' consists of a constant number of phases that each either read a binary word w of length n and store it in the state, or compare w symbol by symbol with some word stored in the state. This can be implemented using $O(2^n)$ linear states. The number of hierarchical states of A' is exactly 2^n.

To conclude construction, we modify A' as follows to obtain a DNWA A for the nested word language L_n. If the input consists only of symbols 0, 1, a, the DNWA A is made to accept. This case applies also when the prefix of length $n+1$ of the input is not of the form (3). This modification requires that A has to remember whether or not it has seen symbols other than 0, 1, a, and hence it is sufficient to double the number of states. □

Lemma 3. *Let $\Sigma = \{0, 1, a, b\}$. For nested word languages $L_n \subseteq \mathrm{NW}(\Sigma)$ as in (1),*

$$\mathrm{sc}(L_n^*) \in \Omega(2^{n \cdot 2^n}).$$

Proof. We denote by x_i, $1 \le i \le 2^n$, the unique word in $\{0,1\}^n$ such that $\mathrm{num}(x_i) = i - 1$. We denote by $\mathcal{F}_{2^n}$ the set of functions from $\{1, \ldots, 2^n\}$ into $\{0,1\}^n$. For $f \in \mathcal{F}_{2^n}$, we define the nested word

$$v_f = x_1 a x_2 \cdots a x_{2^n} a \langle b f(1) a f(2) a \cdots a f(2^n).$$

Consider an arbitrary $y \in \{0,1\}^n$. Since all the words x_i, $i = 1, \ldots, 2^n$, are distinct, it follows that for any x_i, $1 \le i \le 2^n$,

$$v_f byb\rangle x_i \in L_n^* \quad \text{iff} \quad x_i a \cdots x_{2^n} a \langle b f(1) a \cdots f(2^n) byb\rangle x_i \in L'_n \quad \text{iff} \quad y = f(i).$$

For verifying the first "iff" statement, recall that the subset of L_n consisting of words containing some call or return symbols is exactly the language L'_n (2). Hence $wb\rangle x_i$ can be in L_n^* only if $wb\rangle x_i$ has a suffix in L'_n.

From the above it follows that the set $S_1 = \{v_f \mid f \in \mathcal{F}_{2^n}\}$ is a 1-separator set for L_n^*. If Q and P are, respectively, the sets of linear and hierarchical states of an arbitrary DNWA B recognizing L_n^*, we have by Lemma 1 (i) that

$$|Q| \cdot |P| \ge |S_1| = 2^{n \cdot 2^n}.$$

Now the number of states of B is greater than $\max(|Q|, |P|) \ge 2^{n \cdot 2^{n-1}}$. □

As a consequence of Lemmas 2 and 3 we get:

Theorem 1. *For arbitrarily large $n \in \mathbb{N}$ there exist regular nested word languages M_n such that*

$$\mathrm{sc}(M_n) \in O(n) \quad \textit{and} \quad \mathrm{sc}(M_n^*) \in 2^{\Omega(n \cdot \log n)}.$$

The lower bound of Theorem 1 is greater than the state complexity of Kleene star of ordinary regular languages, however, it is very far from the best known upper bound. Proposition 1 and the upper bound for the nondeterministic state complexity of star [18] give only an upper bound of the form $2^{c \cdot n^2}$ (for a suitable constant c) for the state complexity of the star of an n state DNWA language.

4 Summary of State Complexity Results and Open Problems

Deterministic and nondeterministic operational state complexity has been investigated in [24] and [18], respectively, and the known upper and lower bounds are summarized in Table 1. The lower bound for the deterministic state complexity of Kleene star was established in the previous section in Theorem 1. The upper bounds for the deterministic state complexity of Kleene star and reversal and the nondeterministic state complexity of complementation follow from the construction converting an arbitrary NNWA to a DNWA [5].

When the lower and upper bounds do not coincide, in Table 1 the row element for that operation is divided into two parts. In the table, k_i, h_i, $i = 1, 2$, refer to the numbers of linear and hierarchical states, respectively, of the DNWAs (or

Table 1. Deterministic and nondeterministic state complexity. The entries (†) indicate a tight bound within an additive constant. The lower bound (‡) can be reached by the catenation of L_1 and L_2 where $\mathrm{sc}(L_1) = 2$ and $\mathrm{sc}(L_2) = n_2$. The bound ($\star$) is tight in terms of the total number of states and $\max(h_1, h_2)$ is a lower bound for the number of hierarchical states.

	Deterministic s.c.	Nondeterministic s.c.
Union	$(4k_1k_2, 4h_1h_2)^{(\dagger)}$	$(k_1 + k_2, \max(h_1, h_2) + 2)^{(\star)}$
Intersection	(k_1k_2, h_1h_2)	(k_1k_2, h_1h_2)
Complement (l.b.)	$(2k_1, 2h_1)^{(\dagger)}$	$\Omega(\sqrt{n_1!})$
(u.b.)		$O(2^{n_1^2})$
Concatenation (l.b.)	$2^{\Omega(n_2 \log n_2)}$ (‡)	$(k_1 + k_2, \max(h_1, h_2))$
(u.b.)	$O(n_1 \cdot 2^{4n_2^2})$	$(k_1 + k_2, h_1 + h_2)$
Kleene star (l.b.)	$2^{\Omega(n_1 \log n_1)}$	(k_1, h_1)
(u.b.)	$2^{O(n_1^2)}$	$(4k_1, 4h_1)$
Reversal (l.b.)	$2^{\Omega(n_1 \log n_1)}$	(k_1, h_1)
(u.b.)	$2^{n_1^2}$	

NNWAs) recognizing the argument languages. The symbols n_i, $i = 1, 2$, refer to the total number of states used by the automata for the argument languages.

More specifically, in the deterministic (respectively, nondeterministic) state complexity column of the table, a lower bound (respectively, an upper bound) for a binary operation $\odot$, $(f(k_1, k_2), g(h_1, h_2))$ indicates that for nested word languages L_i recognized by a DNWA (resp. NNWA) with k_i linear and h_i hierarchical states, $i = 1, 2$, a DNWA (resp. NNWA) for the language $L_1 \odot L_2$ needs in the worst-case at least (respectively, at most) $f(k_1, k_2)$ linear and $g(h_1, h_2)$ hierarchical states. For a unary operation, k_1 and h_1 refer, respectively, to the numbers of linear and hierarchical states of the DNWA (resp. NNWA) recognizing the single argument language.

For the deterministic state complexity of concatenation, Kleene star and reversal, as well as the nondeterministic state complexity of complementation the bounds are stated simply in terms of the total number of states, where n_1, n_2 refer to the deterministic (or nondeterministic) state complexity of the argument languages. For these operations the upper and lower bounds remain far apart and detailed formulas in terms of the numbers of linear and hierarchical states would appear unnecessarily complicated.

Table 1 gives tight results (within an additive constant) for union and intersection and for the deterministic state complexity of complement. The results for intersection are perhaps "expected", that is, they are similar to the case of ordinary regular languages, with the only difference that we need to consider separately the sets of linear and hierarchical states, respectively. The upper bounds for the deterministic state complexity of union and complement involve multiplicative constants that do not appear in corresponding bounds for ordinary regular languages. Intuitively, the constants are needed because in a deterministic nested word automaton recognizing the union of languages recognized by two different automata, the linear state has to remember (a constant amount of) information propagated along the pending hierarchical states. It should be emphasized that this is not simply a feature of a particular construction used to handle union, and the upper bound can be reached at least within an additive constant [24].

When constructing an NNWA to recognize the union of languages recognized by two different automata, the hierarchical states can be reused in both "parts" of the automaton whereas the same property does not hold for the linear states. Thus the roles played by the numbers of linear and hierarchical states in the upper bound for the nondeterministic state complexity of union appear to be different. Again the bound is known to be tight [18].

When considering ordinary regular languages, the worst-case deterministic state complexity of concatenation, Kleene star and reversal is exponential [29] and the same holds for the nondeterministic state complexity of complementation [19,22]. Thus, concatenation, Kleene star and reversal can be viewed as "hard" operations for deterministic automata while complementation is "hard"

for nondeterministic automata. The results summarized in Table 1 indicate that for each of the "four hard operations" the best known lower bound for nested word automata is of a different order than the worst-case state complexity of the same operation for ordinary regular languages. Concerning the corresponding upper bounds, the bound for concatenation [24] combines ideas of the optimal DFA construction for the concatenation of two DFA languages [29] with the construction that converts an arbitrary NNWA to a DNWA [4,5]. The remaining three upper bounds for the "hard" operations are direct corollaries of Proposition 1 from [5]. Likely the upper bounds could be slightly improved using a detailed analysis.

The upper and lower bounds for the "four hard operations" given in Table 1 remain very far apart and, thus, the main open questions on operational state complexity of nested word automata deal with closing the gaps between these bounds. It remains to be seen whether we need stronger lower bound techniques than Lemma 1, or whether up to now we have simply not been able to come up with sufficiently clever worst-case examples.

Problem 1. What is the precise deterministic state complexity of concatenation, Kleene star and reversal, and the precise nondeterministic state complexity of complementation of nested word languages.

Besides the above mentioned operations, also some other state complexity results in Table 1 are not tight. The upper and lower bound entries for the nondeterministic state complexity of concatenation and Kleene star differ by a multiplicative constant and, as discussed in [18], just by relying on Lemma 1 (ii) it seems difficult to get tight bounds for these operations. As regards nondeterministic state complexity of reversal, differing from the case of ordinary NFAs [19] the result of Table 1 does not need an additional state because the definition of NNWAs allows the use of multiple initial states.

In this paper we have concentrated on state complexity of operations on regular nested word languages. Besides closure properties, the regular nested word languages retain many of the nice decidability properties of regular languages. As regards complexity of decision problems, it is shown in [5], for example, that equivalence and inclusion of DNWAs can be decided in polynomial time and that equivalence and inclusion of NNWAs are complete for exponential time. Recall that equivalence and inclusion are PSPACE-complete for NFAs [29].

Since congruence based characterizations seem not sufficient to minimize DNWAs [3,5], minimization becomes a combinatorial question. Minimization of subclasses of visibly pushdown automata (VPA) has been investigated in [8,23]. The computation of a DNWA on a nested word can be viewed as a computation of a deterministic VPA on the underlying tagged word [5], and the minimization problems of DNWAs and VPAs are closely related. It remains open whether general DNWAs can be minimized in polynomial time. On the other hand, there is no known hardness result for the complexity of DNWA minimization.

References

1. Alur, R.: Marrying words and trees. In: Proc. of 26th ACM Symposium on Principles of Database Systems, PODS 2007, pp. 233–242 (2007)
2. 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)
3. 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)
4. Alur, R., Madhusudan, P.: Adding nesting structure to words. In: H. Ibarra, O., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 1–13. Springer, Heidelberg (2006)
5. Alur, R., Madhusudan, P.: Adding nesting structure to words. Full version of [4], www.cis.upenn.edu/~alur/Stoc04Dlt06.pdf
6. 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)
7. Birget, J.C.: Intersection and union of regular languages and state complexity. Inform. Process. Lett. 43, 185–190 (1992)
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. Domaratzki, M., Salomaa, K.: Lower bounds for the transition complexity of NFAs. J. Comput. System. Sci. 74, 1116–1130 (2008)
10. Domaratzki, M., Salomaa, K.: Transition complexity of language operations. Theoret. Comput. Sci. 387, 147–154 (2007)
11. Gao, Y., Salomaa, K., Yu, S.: The state complexity of two combined operations: Star of catenation and star of reversal. Fund. Inform. 83, 75–89 (2008)
12. Gauwin, O., Niehren, J., Roos, Y.: Streaming tree automata. Inform. Proc. Lett. 109, 13–17 (2008)
13. Gécseg, F., Steinby, M.: Tree languages. In: [27], vol. III, pp. 1–68
14. Glaister, I., Shallit, J.: A lower bound technique for the size of nondeterministic finite automata. Inform. Process. Lett. 59, 75–77 (1996)
15. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. Universal Comput. Sci. 8, 193–234 (2002)
16. Gruber, H., Holzer, M.: Finding lower bounds for nondeterministic state complexity is hard. In: H. Ibarra, O., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 363–374. Springer, Heidelberg (2006)
17. Gruber, H., Holzer, M.: Inapproximability of nondeterministic state and transition complexity assuming P $\neq$ NP. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 205–216. Springer, Heidelberg (2007)
18. Han, Y.-S., Salomaa, K.: Nondeterministic state complexity of nested word automata (September 2008) (submitted for publication)
19. Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. Internat. J. Foundations of Comput. Sci. 14, 1087–1102 (2003)
20. Hromkovič, J.: Communication Complexity and Parallel Computing. Springer, Heidelberg (1997)

21. Hromkovič, J.: Descriptional complexity of finite automata: Concepts and open problems. J. Automata, Languages and Combinatorics 7, 519–531 (2002)
22. Jirásková, G.: State complexity of some operations on binary regular languages. Theoret. Comput. Sci. 330, 287–298 (2005)
23. Kumar, V., Madhusudan, P., Viswanathan, M.: Minimization, learning, and conformance testing of boolean programs. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 203–217. Springer, Heidelberg (2006)
24. Piao, X., Salomaa, K.: Operational state complexity of nested word automata. In: Câmpeanu, C., Pighizzini, G. (eds.) Descriptional Complexity of Formal Systems, DCFS 2008, Charlottetown, Canada, pp. 194–206 (2008)
25. Neumann, A., Seidl, H.: Locating matches of tree patterns in forests. In: Arvind, V., Ramanujam, R. (eds.) FST TCS 1998. LNCS, vol. 1530, pp. 134–146. Springer, Heidelberg (1998)
26. Nguyen, H.: VPAlib: Visibly pushdown automata library (2006), http://www.emn.fr/x-info/hnguyen/vpa
27. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. I–III. Springer, Heidelberg (1997)
28. Salomaa, K., Yu, S.: On the state complexity of combined operations and their estimation. Internat. J. Foundations of Comput. Sci. 18, 683–698 (2007)
29. Yu, S.: Regular languages. In: [27], vol. I, pp. 41–110
30. Yu, S.: Grail+: A symbolic computation environment for finite-state machines, regular expressions and finite languages (2002), http://www.csd.uwo.ca/Research/grail
31. Yu, S.: State complexity: Recent results and open problems. Fund. Inform. 64, 471–480 (2005)

A Language-Based Comparison of Extensions of Petri Nets with and without Whole-Place Operations

Parosh Aziz Abdulla[1], Giorgio Delzanno[2], and Laurent Van Begin[3]

[1] Uppsala University, Sweden
parosh@it.uu.se
[2] Università di Genova, Italy
giorgio@disi.unige.it
[3] Université Libre de Bruxelles, Belgium
lvbegin@ulb.ac.be

Abstract. We use language theory to study the relative expressiveness of infinite-state models laying in between finite automata and Turing machines. We focus here our attention on well structured transition systems that extend Petri nets. For these models, we study the impact of whole-place operations like transfers and resets on nets with indistinguishable tokens and with tokens that carry data over an infinite domain. Our measure of expressiveness is defined in terms of the class of languages recognized by a given model using coverability of a configuration as accepting condition.

1 Introduction

The class of *well-structured transition systems* (wsts) [1] includes several interesting examples of infinite-state models whose expressiveness lay in between that of finite automata and that of Turing machines. Some examples of wsts are *Petri nets* [1], *transfer* and *reset nets* [2], *lossy FIFO channel systems* (LCS) [3,4], and *constrained multiset rewriting systems* (CMRS) [5]. Petri nets are a widely used model of concurrent computations. A Petri net is defined by a finite set of places containing multisets of tokens and by a finite set of transitions that define the flow of tokens among places. Each transition first consumes and then produces a fixed number of tokens in each place. Transfer/reset nets extend Petri net with *whole-place operations*, i.e., transitions that operate simultaneously on all tokens in a given set of places. In a lossy FIFO channel system places are viewed instead as unreliable FIFO channels. Finally, CMRS can be viewed as an extension of Petri nets in which tokens carry natural numbers and transitions are guarded by constraints on data attached to tokens. For all the above mentioned models, the *coverability problem* is decidable [5,3,4,2]. This decision problem is of great importance for verification of safety properties like mutual exclusion.

An interesting research question concerns the study of the relative expressiveness of well-structured models. For this purpose, it comes natural to use tools from language theory to compare the languages generated by labelled transition

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 71–82, 2009.

systems that describe the operational semantics of different models. Unfortunately, a standard notion of acceptance like *reachability* of a configuration is not adequate to obtain a fine-grained classification of wsts. For instance, with this notion of acceptance transfer/reset nets are equivalent to Turing machines. As shown in [6,7,8], a finer classification of wsts can be obtained by considering the class of languages recognized with *coverability acceptance conditions* (c-languages for short). A classification of wsts based on c-languages is particularly interesting since it can be used to extend the applicability of a decision procedure for coverability (e.g. the symbolic backward reachability algorithm in [9]) from a particular wsts model to an entire class.

In this paper we use c-languages as a formal tool to study the impact of *whole-place operations* on the expressiveness of Petri nets with black indistinguishable tokens and with tokens that carry data over an ordered domain. For this purpose, we compare the expressiveness of Petri nets, LCS, and CMRS with that of *affine well-structured nets* (AWNs) [10] and *data nets* [11]. AWNs are a generalization of Petri nets and transfer/reset nets in which the firing of a transition is split into three steps: subtraction, multiplication, and addition of black tokens. Multiplication is a generalization of transfer and reset arcs. Data nets can be viewed as a generalization of AWNs in which these steps are defined on tokens that carry data taken from an infinite, ordered domain. Conditions on data values can be used here to restrict the type of tokens on which apply whole-place operations. Although presented in a different style, a data net can be viewed as a CMRS enriched with whole-place operations.

For the above mentioned models, we prove the following results. We first show that AWNs are strictly more expressive than Petri nets and strictly less expressive than lossy FIFO channel systems. The proof of the second result exploits a non-trivial property of the class of c-languages recognized by AWNs based on Dickson's lemma [12]. We then show that, differently from nets with indistinguishable tokens, whole-place operations do not augment the expressive power of models in which tokens carry data taken from an ordered domain. The proof is based on a weak, effectively constructible encoding of data nets into CMRS that can be used to reduce the coverability problem from one model to the other. Weakness refers here to the fact that the CMRS encoding simulates a *lossy* version of data nets, i.e., data nets in which tokens may get lost. However this is enough to show that the two models define the same class of c-languages.

Our analysis has several interesting consequences. First, it can be used to give a strict classification of the expressiveness of a large class of wsts models taken from the literature. Furthermore, it shows that the symbolic backward reachability algorithm for solving the CMRS coverability problem given [5] can also be applied in presence of whole-place operations like transfer and reset of colored tokens. Finally, as discussed in the conclusions, our weak encoding of data nets into CMRS can naturally be adapted to extend the decidability of coverability to a more general definition of data nets transition than the one given in [11]. Our extensions include, for instance, generation of fresh values, a feature present in several models of concurrency like CCS and π-calculus.

Related Work. In [7,8] the authors compare the relative expressiveness of Petri nets with reset, transfer, and non-blocking arcs. A classification of infinite-state systems in terms of decidable properties is presented in [13]. The classification is extended to well-structured systems in [14]. Both classifications do not include models like CMRS and data nets. A classification of the complexity of the decision procedures for coverability of different formulations of data nets is studied in [11]. In [6] we have compared CMRS with lossy FIFO channel systems and other weaker models like relational automata. However, we have not considered whole-place operations like those in AWNs and data nets. We believe that a comparative study of all these sophisticated models can be useful to find new applications of the theory of well-structured transition systems.

Preliminary Notions. In this paper we consider extensions of finite automata defined by using labelled transition systems. A transition system $T = (S, R)$ consists of a set S of configurations and of a set R of transitions, where a transition $\xrightarrow{\rho} \subseteq S \times S$. A transition system T is said to be *well-structured* (wsts) with respect to a quasi ordering $\preceq$ on configurations iff the following conditions hold: (i) $\preceq$ is a *well-quasi ordering*, i.e., for any infinite sequence of configurations $\gamma_1\gamma_2 \ldots \gamma_i \ldots$ there exist indexes $i < j$ such that $\gamma_i \preceq \gamma_j$; (ii) T is *monotonic*, i.e., for any $\xrightarrow{\rho} \in R$, if $\gamma_1 \preceq \gamma_2$ and $\gamma_1 \xrightarrow{\rho} \gamma_3$, then there exists γ_4 s.t. $\gamma_3 \preceq \gamma_4$ and $\gamma_2 \xrightarrow{\rho} \gamma_4$.

Given a wsts T, we label each transition in R either with a symbol ℓ from an alphabet Σ or with the empty word ϵ (*silent transition*). If we associate to a wsts T an *initial* configuration γ_0 and a *final* configuration γ_{acc}, the language recognized by T with coverability acceptance (c-language for short) is defined as follows:

$$L_c(T) = \{w \in \Sigma^* \mid \gamma_0 \stackrel{w}{\Longrightarrow} \gamma \text{ and } \gamma_{acc} \preceq \gamma\}$$

where $\gamma_0 \stackrel{w}{\Longrightarrow} \gamma$ denotes a finite sequence of application of transitions such that the concatenation of their labels produces the word w. We use $L_c(\mathcal{M})$ to denote the class of c-languages recognized by instances T of a given model $\mathcal{M}$ (e.g. Petri nets, transfer nets, etc.), i.e., $L_c(\mathcal{M}) = \{L \mid \exists S \in \mathcal{M},\ L = L_c(S)\}$.

Given a wsts $T = (S, R, \preceq)$ with labels in $\Sigma \cup \{\epsilon\}$, a *lossy* version of T is a wsts $T' = (S, R', \preceq)$ for which there exists a bijection $h : R \mapsto R'$ such that $\xrightarrow{\rho} \in R$ and $\xrightarrow{h(\rho)}$ have the same label, $\xrightarrow{\rho} \subseteq \xrightarrow{h(\rho)}$ and if $\gamma \xrightarrow{h(\rho)} \gamma'$, then $\gamma \xrightarrow{\rho} \gamma''$ with $\gamma' \preceq \gamma''$. In a lossy version of a wsts, the set of reachable configurations contains configurations that are smaller than those of the original model. The following lemma then holds.

Lemma 1. *For any lossy version T' of a wsts T, we have that $L_c(T) = L_c(T')$.*

2 Whole-Place Operations in Nets with Black Tokens

In this section we use c-languages as a formal tool to compare the expressiveness of Petri nets, affine well-structured nets (AWNs) [10], and lossy FIFO channel

$$F_t = \begin{pmatrix} p & q \\ 1 & 0 \end{pmatrix} \qquad G_t = \begin{pmatrix} & p & q \\ p & 1 & 0 \\ q & 0 & 0 \end{pmatrix} \qquad H_t = \begin{pmatrix} p & q \\ 0 & 1 \end{pmatrix}$$

Fig. 1. An example of AWN transition

systems (LCS) [3,4]. AWNs are a generalization of Petri nets in which transitions admit *whole-place* operations, i.e., operations that operate simultaneously on the whole set of tokens in a given place. Examples of whole-place operations are reset (all tokens in a place are consumed) and transfer arcs (all tokens in a place are transferred to another place) [2]. Formally, an AWN consists of a finite set P of places and of a finite set T of transitions. As in Petri nets, AWN-configurations, called *markings*, are vectors in $\mathbb{N}^P$, i.e., finite multisets with symbols in P. A marking counts the current number of tokens in a given place in P. In the rest of the paper we use $[a_1, \ldots, a_n]$ to indicate a multiset with elements $a_1, \ldots, a_n$. Furthermore, for a marking M, we use $M(a)$ to denote the number of tokens in place a. Finally we use, $-$ and $+$ to denote multiset difference and union.

An AWN-transition t is defined by two vectors F_t and H_t in $\mathbb{N}^P$, and by a $\mathbb{N}^P \times \mathbb{N}^P$-matrix G_t. Intuitively, F_t defines a subtraction step (how many tokens to remove from each place), G_t defines a multiplication step (whole-place operations), and H_t defines an addition step (how many tokens are added to each place). t is enabled at marking M if $F_t \leq M$ where $\leq$ denotes marking (multiset) inclusion, i.e., $M \leq M'$ iff $M(p) \leq M'(p)$ for each $p \in P$. The firing of t at a marking M amounts to the execution of the three steps in sequence. Formally, it produces a new marking $M' = ((M - F_t) \cdot G_t) + H_t$, where $\cdot$ denotes the multiplication of vector $(M - F_t)$ and matrix G_t. As an example, let $P = \{p, q\}$ and consider the transition t in Fig. 1. This transition removes a token from p and resets the number of tokens in q to 1. For instance, from the marking $M = [p, p, q, q, q]$, i.e., the vector $(2, 3) \in \mathbb{N}^P$, we obtain the new marking $M' = [p, q]$ defined by the vector $((2, 3) - (1, 0)) \cdot G_t + (0, 1) = (1 * 1 + 3 * 0, 1 * 0 + 3 * 0) + (0, 1) = (1, 1)$.

As shown in [10], AWN are well-structured with respect to marking inclusion $\leq$. Petri nets are the subclass of AWNs in which G_t is the identity matrix, i.e., with no whole-place operations. In [7] the authors have shown that there exists a c-language $L \in L_c(Transfer\ nets)$ such that $L \notin L_c(Petri\ nets)$. Since transfer nets are a special case of AWNs, we obtain the following property.

Proposition 1. $L_c(Petri\ nets) \subset L_c(AWN)$.

To obtain a sort of upper bound on the expressive power of nets with whole-place operations, we can consider nets in which places maintain some kind of order between their tokens as in *lossy FIFO channel systems* (LCS). A LCS is a tuple (Q, C, N, δ), where Q is a finite set of control states, C is a finite set of channels, N is a finite set of messages, δ is a finite set of transitions, each of which is of the form (q_1, Op, q_2) where $q_1, q_2 \in Q$, and Op is a mapping from channels to channel operations. For any $c \in C$ and $a \in N$, an operation $Op(c)$ is either a *send* operation $!a$, a *receive* operation $?a$, the *empty* test $\epsilon?$, or the *null* operation *nop*. A configuration γ is a pair (q, w) where $q \in Q$, and w is a mapping from C to N^*

giving the content of each channel. The initial configuration γ_{init} of $\mathcal{F}$ is the pair (q_0, ε) where $q_0 \in Q$, and ε denotes the mapping that assigns the empty sequence ϵ to each channel. The (strong) transition relation (that defines the semantics of machines with *perfect* FIFO channels) is defined as follows: $(q_1, w_1) \xrightarrow{\sigma} (q_2, w_2)$ if and only if $\sigma = (q_1, Op, q_2) \in \delta$ such that, for all $c \in C$, if $Op(c) = !a$, then $w_2(c) = w_1(c) \cdot a$; if $Op(c) = ?a$, then $w_1(c) = a \cdot w_2(c)$; if $Op(c) = \epsilon?$ then $w_1(c) = \epsilon$ and $w_2(c) = \epsilon$; if $Op(c) = nop$, then $w_2(c) = w_1(c)$. Now let $\preceq_l$ be the well-quasi ordering on LCS configurations defined as: $(q_1, w_1) \preceq_l (q_2, w_2)$ if and only if $q_1 = q_2$ and $\forall c \in C : w_1(c) \preceq_w w_2(c)$, where $\preceq_w$ indicates the subword relation. We introduce then the weak transition relation $\stackrel{\sigma}{\Longrightarrow}$ that defines the semantics of LCS: we have $\gamma_1 \stackrel{\sigma}{\Longrightarrow} \gamma_2$ iff there exists γ_1' and γ_2' s.t. $\gamma_1' \preceq_l \gamma_1$, $\gamma_1' \xrightarrow{\sigma} \gamma_2'$, and $\gamma_2 \preceq_l \gamma_2'$. Thus, $\gamma_1 \stackrel{\sigma}{\Longrightarrow} \gamma_2$ means that γ_2 is reachable from γ_1 by first losing messages from the channels and reaching γ_1', then performing a transition, and, thereafter losing again messages from channels. As shown in [3,4], LCS are well-structured w.r.t. $\preceq_l$. The following theorem then holds.

Theorem 1. $L_c(AWN) \subset L_c(LCS)$.

Proof

(1) We first prove the inclusion $L_c(AWN) \subseteq L_c(LCS)$. Assume an AWN $W = (P, T, F, G, H)$ with $P = \{p_1, \ldots, p_n\}$. We build a LCS $\mathcal{F} = (Q, C, N, \delta)$ such that $L_c(W) = L_c(\mathcal{F})$. W.l.o.g. we assume that channels can be non-empty in the initial configuration of a LCS. The set of channels is defined as $C = P \cup P'$ where P' (auxiliary channels) contains a primed copy of each element in P. The set of messages N contains the symbol $\bullet$ (a representation of a black token). Assume that $q_0 \in Q$ is the initial state of $\mathcal{F}$. Then, a marking M is encoded as a LCS configuration $enc(M)$ with state q_0 and in which channel $p_i \in P$ contains the word $\bullet^{m_i}$ containing $m_i = M(p_i)$ occurrences of symbol $\bullet$ for $i \in \overline{n}$, and all channels in P' are empty (we define $\overline{n}$ as $[1, \ldots, n]$).

For each transition t with label ℓ, we need to simulate the three steps (subtraction, multiplication, and addition) that correspond to F_t, G_t and H_t. Subtraction and addition can be simulated in a straightforward way by removing/adding the necessary number of tokens from/to each channel. The multiplication step is simulated as follows. For each $i \in \overline{n}$, we first make a copy of the content of channel p_i in the auxiliary channel p_i'. Each copy is defined by repeatedly moving a symbol from p_i to p_i' and terminates when p_i becomes empty. When the copy step is terminated, we start the multiplication step. For each $i \in \overline{n}$, we remove a message $\bullet$ from p_i' and add as many $\bullet$'s to channel p_j as specified by $G_t(p_i, p_j)$ for $j \in \overline{n}$. This step terminates when the channels $p_1', \ldots, p_n'$ are all empty. For an accepting AWN-marking M_f, the accepting LCS-configuration is $enc(M_f)$.

The following properties then hold: *i)* We first notice that $M \leq M'$ iff $enc(M) \preceq_l enc(M')$; *ii)* Furthermore, if $M_0 \stackrel{w}{\Longrightarrow} M_1$ in W, then $enc(M_0) \stackrel{w}{\Longrightarrow} enc(M_1)$ in $\mathcal{F}$; *iii)* Finally, since $\bullet$ symbols may get lost in $\mathcal{F}$, if $enc(M_0) \stackrel{w}{\Longrightarrow} enc(M_1)$ then there exists M_2 such that $M_0 \stackrel{w}{\Longrightarrow} M_2$ and $M_1 \leq M_2$. Since we consider languages with coverability acceptance, $L_c(W) = L_c(\mathcal{F})$ immediately follows from properties *(i),(ii)*, *(iii)* and Lemma 1.

(2) We prove now that $L_c(LCS) \not\subseteq L_c(AWN)$. For this purpose, we exhibit a language in $L_c(LCS)$ and prove that it cannot be recognized by any AWN.

Fix a finite alphabet $\Sigma = \{a, b, \sharp\}$ and let $\mathcal{L} = \{w\sharp w' |\ w \in \{a,b\}^* \text{ and } w' \preceq_w w\}$. It is easy to define a LCS that accepts the language L: we first put w in a lossy channel and then remove one-by-one all of its messages. Thus, we have that $\mathcal{L} \in L_c(LCS)$. We now prove that there is no AWN that accepts $\mathcal{L}$. Suppose it is not the case and there exists a AWN N, with (say) n places, that recognizes $\mathcal{L}$ with initial marking M_{init} and accepting marking M_f.

For each $w \in \{a,b\}^*$, there is a marking M_w such that $M_{init} \stackrel{w\sharp}{\Longrightarrow} M_w \stackrel{w}{\Longrightarrow} M$ and $M_f \leq M$ (otherwise $w\sharp w$ would not be in $L_c(N)$). Consider the sequences $w_0, w_1, w_2, \ldots$ and $M_{w_0}, M_{w_1}, M_{w_2}, \ldots$ of words and markings defined as follows:

- $w_0 := b^n$;
- If $M_{w_i} = (m_1, \ldots, m_n)$ then $w_{i+1} := a^{m_1}\, b\, a^{m_2}\, b \cdots b\, a^{m_n}$, for $i = 0, 2, \ldots$

We observe that (a) $w_0 \not\preceq_w w_i$ for all $i > 0$, since w_0 contains n occurrences of b, while w_i contains only $n-1$ occurrences of b; and (b) for any $i < j$, $M_{w_i} \leq M_{w_j}$ iff $w_{i+1} \preceq_w w_{j+1}$. By Dickson's lemma [12], there are $i < j$ such that $M_{w_i} \leq M_{w_j}$. Without loss of generality, we can assume that j is the smallest natural number satisfying this property. Remark that we have that $w_i \not\preceq_w w_j$. Indeed, $w_0 \not\preceq_w w_j$ for any $j > 0$ by (a), and in the case of $i > 0$ we have by (b) that $w_i \not\preceq_w w_j$ since $M_{w_{i-1}} \not\leq M_{w_{j-1}}$. Since $M_{w_i} \leq M_{w_j}$, by monotonicity of AWNs, we have that $M_{w_i} \stackrel{w_i}{\Longrightarrow} M$ with $M_f \leq M$ implies that $M_{w_j} \stackrel{w_i}{\Longrightarrow} M'$ with $M_f \leq M \leq M'$. Hence, $M_{init} \stackrel{w_j\sharp w_i}{\Longrightarrow} M'$ and $w_j\sharp w_i \in L_c(N) = \mathcal{L}$, which is a contradiction. □

By combining Prop. 1 and Theorem 1 we obtain the following strict classification.

$$L_c(Petri\ nets) \subset L_c(AWN) \subset L_c(LCS)$$

As a corollary, we have that transfer/reset nets are strictly less expressive than LCSs.

3 Whole-Place Operations in Nets with Colored Tokens

In this section we study the impact of whole-place operations on the expressiveness of well-structured colored Petri nets like CMRS [5] and data nets [11]. CMRS is an extension of Petri nets in which tokens are labelled with natural numbers. For a fixed number of places P, we represent a token in place p with value v as the term $p(v)$. A CMRS configuration is a multiset of ground terms like $[p(1), p(3), q(4)]$ (we recall that markings are multisets over P, i.e., a special case of CMRS configurations). We use *P-terms* to denote terms associated to colored tokens. CMRS transitions are defined in terms of conditional multiset rewriting rules of the form $L \rightsquigarrow R : \Psi$ where L and R are terms with variables that describe colored tokens and Ψ is a condition over such variables. Conditions are expressed by a finite conjunction of constraints in the following form: $x + d < y$, $x \leq y$, $x = y$, $x < d$, $x > d$, $x = d$ where x, y are variables appearing in L and/or

$$s = \begin{array}{|cc|cc|cc|cc|}\hline e_1 & & e_2 & & e_3 & & e_4 & \\ p & q & p & q & p & q & p & q \\ 3 & 2 & 5 & 1 & 2 & 10 & 2 & 2 \\ \hline \end{array} \quad s' = \begin{array}{|cc|cc|cc|cc|}\hline e_1 & & e_2 & & e_3 & & e_4 & \\ p & q & p & q & p & q & p & q \\ 29 & 28 & 5 & 1 & 25 & 1 & 2 & 2 \\ \hline \end{array}$$

$$F_t = \begin{array}{|cc|cc|cc|}\hline R_0 & & S_1 & & R_1 & \\ p & q & p & q & p & q \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline \end{array} \quad H_t = \begin{array}{|cc|cc|cc|}\hline R_0 & & S_1 & & R_1 & \\ p & q & p & q & p & q \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \hline \end{array}$$

$$G_t = \begin{array}{cc|cc|cc|cc|}\multicolumn{2}{c|}{} & R_0 & & S_1 & & R_1 & \\ \multicolumn{2}{c|}{} & p & q & p & q & p & q \\ \hline R_0 & p & 1 & 0 & 3 & 0 & 0 & 0 \\ & q & 3 & 1 & 0 & 0 & 0 & 0 \\ \hline S_1 & p & 0 & 0 & 1 & 0 & 0 & 0 \\ & q & 2 & 0 & 0 & 0 & 0 & 0 \\ \hline R_1 & p & 0 & 0 & 0 & 0 & 1 & 0 \\ & q & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \end{array}$$

Fig. 2. Two data net markings (s and s') and a transition t with arity 1

R and $d \in \mathbb{N}$ is a constant. A rule r is enabled at a configuration c if there exists a valuation of the variables Val ($Val(x) \in \mathbb{N}$) such that $Val(\Psi)$ is satisfied and $c \geq Val(L)$. Firing r at c leads to a new multi-set $c' = c - Val(L) + Val(R)$, where $Val(L)$, resp. $Val(R)$, is the multi-set of ground terms obtained from L, resp. R, by replacing each variable x by $Val(x)$. As an example, consider the CMRS rule:

$$\rho = [p(x)\ ,\ q(y)] \quad \rightsquigarrow \quad [q(z)\ ,\ r(x)\ ,\ r(w)] \quad : \ \{x + 2 < y\ ,\ x + 4 < z\ ,\ z < w\}$$

A valuation which satisfies the condition is $Val(x) = 1$, $Val(y) = 4$, $Val(z) = 8$, and $Val(w) = 10$. Thus, to fire t on $c = [p(1), p(3), q(4)]$ we first remove $p(1)$ and $q(4)$ and then add the new tokens $q(8)$, $r(1)$, and $r(1)$, producing the configuration $c' = [p(3), q(8), r(1), r(10)]$.

The coverability problem for CMRS is decidable for an ordering $\preceq_c$ that extends multi-set inclusion by taking into consideration the relative "gaps" among the values on different tokens [5]. We come back to this point later.

It is important to remark that CMRS rules does not provide whole-place operations (the semantics is defined using rewriting applied to sub-multisets of tokens). Despite of it, in [6] we show that colors and gap-order conditions is enough to obtain a model that is strictly more powerful than LCS. By combining this property with Theorem 1, we have that

$$L_c(Petri\ nets) \subset L_c(AWN) \subset L_c(LCS) \subset L_c(CMRS)$$

A natural research question now is whether whole-place operations add power to models like CMRS or not. To answer this question, instead of defining a new version of CMRS, we compare its expressiveness with that of *data nets* [11]. *Data nets* are an extension of AWNs in which tokens are colored with data taken from a generic infinite domain D equipped with a linear ordering $\prec$. As discussed in [11], for coverability we can equivalently consider dense or discrete orderings. A data net has a finite sets of places P and transitions T. A data net marking s is a multiset of tokens that carry (linearly ordered) data in D, i.e., s is a finite sequence of vectors in $\mathbb{N}^P \setminus \{\mathbf{0}\}$, where $\mathbf{0}$ is the vector that contains only 0's. Each

index i in the sequence s corresponds to some $d_i \in D$ (data values that occur in some token) such that $i \leq j$ if and only if $d_i \prec d_j$; $s(i)(p)$ is the number of tokens with data d_i in place p. In Fig. 2 we show two examples of configurations, namely, s and s', for a data net with places $P = \{p, q\}$. The data in D that occur in tokens in s and s' are $e_1 \prec e_2 \prec e_3 \prec e_4$.

Transitions like that in Fig. 2 are defined by vectors (of vectors) F_t and H_t and by a matrix G_t that define resp. subtraction, addition, and multiplication of colored tokens. Vectors and matrices are indexed by *regions* defined by the partitioning of the set of tokens $R(\alpha_t) = (R_0, S_1, R_1, \ldots, S_k, R_k)$ associated to the arity $\alpha_t = k$ of the rule. The arity is used to select k data values $d_1, \ldots, d_k$ either *fresh* or occurring in the current configuration. Region S_i is defined as $\{d_i\}$. Regions R_i's are used to define whole-place operations (e.g. transfers) for tokens whose data are not in $\{d_1, \ldots, d_k\}$. R_0 contains all data $d : d \prec d_1$ in s, R_i contains all $d : d_i \prec d \prec d_{i+1}$ in s for $i : 1, \ldots, k-1$, and R_k contains all $d : d_k \prec d$ in s. To illustrate, consider the marking s and the rule t with has arity 1 both defined in Fig. 2. The partitioning is defined here as $R(\alpha_t) = \{R_0, S_1, R_1\}$. Let us assume that t (non-deterministically) partitions the data in s as follows $R_0 = \{e_1, e_2\}$, $S_1 = \{e_3\}$, and $R_1 = \{e_4\}$, its firing is defined as follows.

Subtraction. F_t specifies the number of tokens with data $d_1, \ldots, d_k$ that have to be removed, for each place in P, from the current configuration s. t is enabled if places have enough tokens to remove. In our example p contains two tokens with value e_3, and F_t specifies that one token with value e_3 must be removed. Thus, t is enabled in s. The subtraction step produces an intermediate configurations s_1 obtained from s by removing one token with data e_3 from place p.

Multiplication. G_t specifies whole-place operations on the regions in $R(\alpha_t)$. In our example the third column of G_t defines the effect of multiplication on the number of tokens with data e_3 in place p in s_1. Specifically, we add to the tokens in place p with value e_3 (1 in position S_1, p, S_1, p in G_t), three new tokens with value e_3 for each token with value in R_0 that lay into place p in s_1 (3 in position R_0, p, S_1, p in G_t). Thus, the total number of tokens with value e_3 in p becomes $(3 + 5) * 3 + 1 = 25$. Furthermore, since the fourth column has only zeroes, all tokens with data e_3 are removed from place q (a reset restricted to all tokens with value e_3 in q). The first column of G_t defines the effect on the tokens with values in R_0 in place p. Specifically, for each $d \in R_0$, we add to place p three tokens with value d for each token with the same value laying into q in s_1 (3 in position R_0, q, R_0, p in G_t); two tokens with data d for each token with data e_3 in q (2 in position S_1, q, R_0, p in G_t). Thus, the total number of tokens with value e_1 in p is now $3 + 3 * 2 + 2 * 10 = 29$ and that for value e_2 in p is now $5 + 3 * 1 + 2 * 10 = 28$. The other columns of G_t leave the same tokens as those in the corresponding regions and places in s_1. We use s_2 to refer to the resulting intermediate configuration.

Addition. H_t specifies the number of tokens that are added, for each place, region, and data to the configuration s_2 to obtain the successor configuration s'.

In our example, we simply add one token with data e_3 to place q. Finally, the new configuration s' is given in Fig. 2.

It is important to remark that whole-place operations are uniformly applied to each data value in a region. Whole-place operations between region R_i and R_j as well as subtractions from a region R_i are forbidden. Furthermore, in case of whole-place operations from R_i to S_j (or vice versa) tokens may change data value (e.g. all tokens with data $d \in R_i$ in p are moved to place q with value d_j), whereas in operations within a single region R_i tokens do not change data value.

As proved in [11], data nets are well-structured with respect to the well-quasi ordering $\preceq_d$ defined on markings as follows. Let $Data(s)$ be the set of data values that occur in a marking s. Then, $s_1 \preceq_d s_2$ iff there exists an injective function $h : Data(s_1) \mapsto Data(s_2)$ such that (i) h is monotonic and (ii) $s_1(d)(p) \leq s_2(h(d))(p)$ for each $d \in Data(s_1)$ and $p \in P$. In other words we compose subword ordering (condition (i)) with multiset inclusion (condition (ii)).

3.1 CMRS, Petri Data Nets, and Data Nets

Data nets without whole place operations (i.e. in which G_t is the identity matrix) are called *Petri data nets*. Petri data nets defined on a domain with a single data value d are equivalent to Petri nets. Furthermore, as discussed in [11], it is possible to effectively build an encoding of CMRS into Petri data nets such that coverability in CMRS can be reduced to coverability into Petri data nets. Indeed, the well-quasi ordering $\preceq_c$ used in CMRS is basically the same as that used in Data nets (the only technical difference is due to the presence of constants in conditions of CMRS rules). Thus, we have that

$$L_c(CMRS) = L_c(Petri\ data\ nets) \subseteq L_c(Data\ nets)$$

We show next that the inclusion is not strict, and that Petri data nets, CMRS, and data nets have all the same expressive power. To prove this result, we have to show that for each Data nets $\mathcal{D}$ we can effectively build a Petri data net or a CMRS $\mathcal{S}$ such that $L_c(\mathcal{S}) = L_c(\mathcal{D})$. Since CMRS rules have a format similar to a (logic) programming language, we find more convenient to describe the encoding in CMRS.

Configurations. Given a multi-set M with symbols in P and a value or variable x, we use M^x to denote the multi set of P-terms such that $M^x(p(x)) = M(p)$ (=number of occurrences of p in M) for each $p \in P$, and $M^x(p(y)) = 0$ for any $y \neq x$ and $p \in P$.

Now assume an initial data net marking s_0 with data $d_1 \prec \ldots \prec d_n$. We build a CMRS representation of s_0 by non-deterministically selecting n natural numbers $v_1 < \ldots < v_n$ strictly included in some interval $[f, l]$. P-terms with parameter v_i represent tokens with data d_i in place p. Formally, we generate the representation of s_0 by adding to $\mathcal{S}$ a rule that rewrites an initial zero-ary term *init* as follows

$$[init] \ \rightsquigarrow \ [first(f), last(l)] + \textstyle\sum_{i:1,\ldots,n} M_i^{x_i} \ : \ f < x_1 < \ldots < x_n < l \quad (init)$$

Here M_i is the multiset $s_0(d_i)$ for each $i \in \overline{n}$. The non-determinism in the choice of $f, l, x_1, \ldots, x_n$ makes the CMRS representation of s_0 independent from specific parameters assumed by terms.

Transitions are encoded by CMRS rules that operate on the values in $[f, l]$ used in the representation of a marking. Most of the CMRS rule are based on left-to-right traversals of P-terms with parameters in $[f, l]$.

Consider a transition t with $\alpha_t = k$. We first define a (silent) CMRS-rule that implements the subtraction step of t:

$$[first(f), last(l)] + F_t(S_1)^{x_1} + \ldots + F_t(S_k)^{x_k} \quad \rightsquigarrow \qquad (subtract)$$
$$[\imath_0(f), \imath_1(x_1), ..., \imath_k(x_k), \imath_{k+1}(l), new_t] : \ f < x_1 < ... < x_k < l$$

Here new_t is a nullary predicate (with no data) that indicates the action to be performed next. In the *subtract* rule we non-deterministically associate a value, represented by variable x_i in the above defined rule, to region S_i. The selection is performed by removing (from the current configuration) the multiset $F_t(S_i)^{x_i}$ that contains $F_t(S_i, p)$ occurrences of $p(x_i)$ for each $p \in P$. The association between value x_i and region S_i is maintained by storing x_i in a $\imath_i$-term (introduced in the right-hand side of the rule). If $F_t(S_i, p) = 0$ for any $p \in P$, then a value x_i may be associated to a data d_i not occurring in the current marking (i.e. selection of fresh data is a special case). Furthermore, by removing both the $first$- and the $last$-term, we disable the firing of rules that encode other data net transitions. The values $x_1, \ldots, x_k$ stored in $\imath_1$-,...,$\imath_k$-terms play the role of pointers to the regions $S_1, \ldots, S_k$. We refer to them as to the set of α_t-*indexes*. The parameters of terms in $[f, l]$ associated to the other regions $R_0, \ldots, R_k$ are called *region-indexes*.

To simulate the multiplication step we proceed as follows. We first make a copy of the multiset of P-terms with parameters $v_1, \ldots, v_n$ in $[f, l]$ by copying each p-term with parameter v_i in a $\overline{p}$-term with parameter w_i such that $f' < w_1 < \ldots < w_n < l'$ and $[f', l']$ is an interval to the right of $[f, l]$, i.e., $l < f'$. The new_t-term in the *subtract* rule is used to enable a set of (silent) CMRS rules (omitted for brevity) that create the copy-configuration. During the copy we add a ✓-term for any *visited* region index. These terms are used to remember region indexes whose corresponding $\overline{P}$-terms are all removed in the multiplication step (e.g. when all tokens with data $d \in R_i$ are removed).

For instance, $[p(v_1), p(v_2), p(v_2), q(v_3)]$ with $f < v_1 < v_2 < v_3 < l$ is copied as $[\overline{p}(w_1), ✓(w_1), \overline{p}(w_2), \overline{p}(w_2), ✓(w_2), \overline{q}(w_3)✓(w_3)]$ for some w_1, w_2, w_3 such that $f < l < f' < w_1 < w_2 < w_3 < l'$. The copy process is implemented by a left-to-right scan of the values that represent data. The scan uses a predicate as a pointer to the current value to consider. The pointer is moved to the right by non-deterministically jumping to a larger value (CMRS conditions cannot specify the "next" value). Thus, during the traversal we may forget to copy some token. This is the first type of loss we find in our encoding. Notice that lost tokens have parameters strictly smaller that f'.

The simulation of the multiplication step operates on the copy-configuration only (with $\overline{P}$-terms only). The intuition behind its definition is as follows. We first consider all α_t-indexes of $\overline{P}$-terms from left to right. For each α_t-index v_i,

we proceed as follows. We first select and remove a term $\overline{p}(v_i)$ (encoding a given token). We compute then the effect of the whole-place operation on the entire set of α_t-indexes (including v_i itself). More specifically, for an α_t-index v_j we add $G_t(S_i, p, S_j, q)$ occurrences of the term $q(v_j)$ to the current CMRS configuration. The use of P- and $\overline{P}$-terms with parameters in the same interval allows us to keep track of tokens still to transfer ($\overline{P}$-terms) and tokens already transferred (P-terms). We then consider all remaining indexes by means of a left-to-right traversal of region-indexes in the current configuration. During the traversal, we add new P-terms with region-indexes as parameters as specified by G_t. During this step, we may forget to transfer some $\overline{P}$-term. This is the second type of loss we find in the encoding. After this step we either consider the next token with α_t-index v_i or we move to the next α_t-index.

After the termination of the whole-place operations for terms with α_t-indexes, we have to simulate the transfer of $\overline{P}$-terms with region-indexes. For each such an index, we transfer tokens within the same region-index or to an α_t-index. To simulate these operations we scan region-indexes from left-to-right to apply the matrix G_t. Furthermore, we mark visited region-indexes using ✓-terms. The ✓-terms are used in the simulation of the addition step.

As a last step we add tokens to α_t-indexes and visited region-indexes as specified by H_t. For α_t-indexes, we need a single rule that applies the matrix H_t. For region-indexes, we traverse from left-to-right the current configuration and apply H_t to each marked (with a ✓-term) region-index w. As mentioned before, the ✓-term allows us to apply H_t to regions emptied by the multiplication step. All the rules are silent except the last rule used to encode addition whose label is the same as that of t.

During the traversal, we may ignore some (marked) region-index. This is the last type of loss in our encoding. The new configuration is the final result of the simulation of the transition. Due to the possible losses in the different simulation steps, we may get a representation of a data net configuration smaller than the real successor configuration.

To formalize the relation between a data net $\mathcal{D}$ and its CMRS encoding $\mathcal{E}(\mathcal{D})$, for a configuration s with data $d_1 \prec \ldots \prec d_k$ we use $s^{\boldsymbol{v}}$ to denote the CMRS representation with indexes $\boldsymbol{v} = (v_1, \ldots, v_k)$. For configurations s_0, s_1, s, we have that *(i)* if $s_0 \overset{w}{\Longrightarrow} s_1$ in $\mathcal{D}$, then there exists $\boldsymbol{v}$ such that $[init] \overset{w}{\Longrightarrow} {s_1}^{\boldsymbol{v}}$ in $\mathcal{E}(\mathcal{D})$. Furthermore, *(ii)* if $[init] \overset{w}{\Longrightarrow} c$ in $\mathcal{E}(\mathcal{D})$ and $s^{\boldsymbol{v}} \preceq_c c$ for some $\boldsymbol{v}$, then there exists s_1 such that $s_0 \overset{w}{\Longrightarrow} s_1$ in $\mathcal{D}$ with $s \preceq_d s_1$. Finally, suppose that the accepting data net marking is a sequence $M_1 \ldots M_k$ of k vectors (multi-sets) over $\mathbb{N}^P$. Then, we add a silent CMRS rule

$$[first(f), last(l)] + \sum_{i \in \{1,\ldots,k\}} M_i^{x_i} \rightsquigarrow [acc] \ : \ f < x_1 < x_2 < \ldots < x_k < l, x = 0$$

where *acc* is a fresh (with arity zero) predicate. By adding this rule, the accepting CMRS configuration can be defined as the singleton $[acc]$. From properties $(i), (ii)$ and Lemma 1, we have the following result.

Theorem 2. *$L_c(Data\ nets) = L_c(CMRS)$.*

4 Conclusions

By combining the results in the present paper with the relation between LCS and CMRS describe in [6], we obtain the following classification of well-structured extensions of Petri nets

$$L_c(Petri\ nets) \subset L_c(AWN) \subset L_c(LCS) \subset L_c(CMRS) = L_c(Data\ nets)$$

This classification reveals a different impact of whole-place operations on nets with black and colored tokens: they augment the expressive power of basic models like Petri nets, but they can be simulated in extended models in which tokens carry ordered data.

We believe that our analysis can also be applied to extend the scope of the decidability results given in [11] to more general well-structured systems obtained by relaxing some of the constraints in the definition of data nets transitions. We plan to explore this research direction in future work.

References

1. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! TCS 256(1-2), 63–92 (2001)
2. Dufourd, C., Finkel, A., Schnoebelen, P.: Reset nets between decidability and undecidability. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 103–115. Springer, Heidelberg (1998)
3. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Information and Computation 127(2), 91–101 (1996)
4. Cécé, G., Finkel, A., Iyer, S.P.: Unreliable channels are easier to verify than perfect channels. Information and Computation 124(1), 20–31 (1996)
5. Abdulla, P.A., Delzanno, G.: On the coverability problem for constrained multiset rewriting. In: Proc. AVIS 2006, an ETAPS 2006 workshop (2006)
6. Abdulla, P.A., Delzanno, G., Van Begin, L.: Comparing the expressive power of well-structured transition systems. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 99–114. Springer, Heidelberg (2007)
7. Finkel, A., Geeraerts, G., Raskin, J.F., Van Begin, L.: On the ω-language expressive power of extended petri nets. TCS 356, 374–386 (2006)
8. Geeraerts, G., Raskin, J.F., Van Begin, L.: Well-structured languages. Acta Informatica 44(3-4), 249–288 (2007)
9. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.K.: General decidability theorems for infinite-state systems. In: Proc. LICS 1996, pp. 313–321 (1996)
10. Finkel, A., McKenzie, P., Picaronny, C.: A well-structured framework for analysing petri net extensions. Information and Computation 195(1-2), 1–29 (2004)
11. Lazić, R.S., Newcomb, T., Ouaknine, J., Roscoe, A.W., Worrell, J.B.: Nets with tokens which carry data. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 301–320. Springer, Heidelberg (2007)
12. Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. Amer. J. Math. 35, 413–422 (1913)
13. Henzinger, T.A., Majumdar, R., Raskin, J.F.: A classification of symbolic transition systems. ACM Trans. Comput. Log. 44(1), 1–32 (2005)
14. Bertrand, N., Schnoebelen, P.: A short visit to the sts hierarchy. ENTCS 154(3), 59–69 (2006)

Minimal Union-Free Decompositions of Regular Languages*

Sergey Afonin and Denis Golomazov

Lomonosov Moscow State University, Institute of Mechanics, Moscow, Russia
serg@msu.ru

Abstract. A regular language is called *union-free* if it can be represented by a regular expression that does not contain the union operation. Every regular language can be represented as a finite union of union-free languages (the so-called *union-free decomposition*), but such decomposition is not necessarily unique. We call the number of components in the minimal union-free decomposition of a regular language *the union width* of the regular language. In this paper we prove that the union width of any regular language can be effectively computed and we present an algorithm for constructing a corresponding decomposition. We also study some properties of union-free languages and introduce a new algorithm for checking whether a regular language is union-free.

1 Introduction

Regular expressions are a natural formalism for the representation of regular languages. It is well known that there exist regular languages that can be represented by infinitely many equivalent regular expressions, and a number of "canonical" forms of regular expressions representing a given regular language have been proposed in the literature, such as concatenative decomposition [1,2] and union-free decomposition (see e.g. [3]). This paper is devoted to the task of finding a minimal union-free decomposition of a regular language. A language is called *union-free* if it can be represented by a regular expression without the usage of the union operation. For example, the language represented by the expression $(a + b^*)^*$ is union-free because there exists an equivalent expression $(a^*b^*)^*$. Union-free languages have been introduced under the name "star-dot regular" languages by J. Brzozowski in [4].

Every regular expression r can be transformed into a regular expression r' in which union operations appear only on the "top level" of the expression, i.e., it takes the following form: $r' = r_1 + \ldots + r_m$, and the regular expressions $r_1, \ldots, r_m$ do not contain the "+" operator (see [3]). This means that every regular language can be represented as a finite union of union-free languages. But this decomposition is not necessarily unique: for example, $(a + b)^* = (a^*b^*)^* = \{\varepsilon\} + a^*ba^* + b^*ab^*$, and these are two different union-free decompositions of the language $(a+b)^*$. We

* The research presented in this paper was partially supported by the RFBR grant number 09-01-00822-a.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 83–92, 2009.

call the minimal number of components in such a representation *the union width* of the regular language and a corresponding decomposition (that is not necessarily unique) is called *a minimal union-free decomposition* of a regular language. In this paper we present an algorithm that computes the union width and constructs a minimal union-free decomposition of a regular language.

The union width of a regular language and corresponding decompositions may be considered as canonical representations of a regular expression, as well as a complexity measure of a regular language [5], similar to the restricted star height.

Union-free decompositions play an important role in the algorithm for checking membership of a regular language in a rational subset of a finitely generated semigroup of regular languages with respect to concatenation as a semigroup product. In order to check such membership one should verify that at least one of the distance automata corresponding to the components of an arbitrary union-free decomposition of a certain regular language is limited. We do not go into the details here (see [6]). We will just mention that checking the limitedness property is PSPACE-complete [7], thus, the number of components in a union-free decomposition is an important parameter influencing membership-checking complexity.

The problem of constructing union-free decompositions of regular languages has also practical applications. In particular, regular languages can be used for a description of the syntactic structure of a programming language [8]. The concatenation operation corresponds to the sequential continuation, the Kleene star corresponds to loops, and the union operation corresponds to branching. In this context, union-free languages represent sequences of operators that do not contain conditional transitions. Minimal union-free decompositions of regular languages may be useful for simplifying and normalizing such descriptions.

The main result of the current paper is that the union width of a regular language L can be effectively computed, and we present an algorithm that constructs the corresponding decomposition. This result is achieved by using the combinatorial technique we have adopted from [2]. First we take the set $C(L)$ of all maximum finite concatenations of letters and certain star languages derived from the automaton of L. We prove that $C(L)$ is a finite set and that for every union-free decomposition of L there exists a union-free decomposition of L of the same number of elements that consists of languages from $C(L)$. Consequently, there exists a minimal union-free decomposition that consists of languages from $C(L)$. Thus we can obtain it by examining all the subsets of the finite set $C(L)$.

We also present an algorithm for checking whether a given regular language is union-free. This decidability result is already known (see Theorem 1 below) but it is based on reduction to the computationally expensive problem of checking limitedness of distance automata, which is PSPACE-complete.

The paper has the following structure: Section 2 provides some basic definitions, Section 3 presents results concerning general properties of union-free languages, Section 4 is devoted to the algorithm for finding a minimal union-free decomposition of a regular language, and Section 5 contains conclusions and ideas for the further work.

2 Preliminaries

Let $\Sigma = \{a_1, \ldots, a_n\}$ be a finite alphabet, $L \subseteq \Sigma^*$ be a regular language and $V = \langle \Sigma, Q, q_0, F, \varphi \rangle$ be the corresponding minimal deterministic finite automaton, where $Q = \{q_1, \ldots, q_m\}$ is the set of all states of V, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\varphi : Q \times \Sigma \to Q$ is the transition function of the automaton. Let $M \subseteq \Sigma^*$, $q_1 \in Q$. The definition of the function φ is extended as follows: $\varphi(q_1, M) = \{q \in Q \mid \exists \alpha = \alpha_1\alpha_2 \cdots \alpha_p \in M : \varphi(\cdots\varphi(\varphi(q_1, \alpha_1), \alpha_2), \ldots, \alpha_p) = q\}$.

An ordered list of states $\{q_1,\ldots,q_m\}$ $(q_i \in Q)$ is called *a path marked with a word* $w \in \Sigma^*$ iff $w = a_1 \cdots a_{m-1}$ and $\varphi(q_i, a_i) = q_{i+1}$, $i = 1, \ldots, m-1$. A path in an automaton is called *cycle-free* iff it starts at the initial state q_0, ends at a final state $q_f \in F$ and does not contain any cycles, i.e., there is no state occurring in the list more than once. It should be noted that by "a cycle-free path in a language" we actually mean a word in the language that is represented by a cycle-free path in the minimal automaton associated with the language.

A language $W \subseteq \Sigma^*$ is called *a star language* iff $W = V^*$ for some $V \subseteq \Sigma^*$. A language L is called *union-free* iff it can be represented by a regular expression that contains the star and concatenation operations only, i.e., it takes the following form:

$$L = S_{01}^* S_{02}^* \cdots S_{0k_0}^* u_1 S_{11}^* \cdots S_{1k_1}^* u_2 \cdots S_{l-1,1}^* \cdots S_{l-1,k_{l-1}}^* u_l S_{l,1}^* \cdots S_{l,k_l}^*, \quad (1)$$

where S_{ij} are regular languages, $u_1, \ldots, u_l$ are non-empty words, and $l \geqslant 0$. We call (1) *a general form* of a union-free language and denote it $GF(L)$.

Let L be a union-free language. We denote $\mathrm{tsw}(L)$ the shortest word in L. Proposition 1 shows that the definition is correct, i.e., there cannot exist two different words of minimum length.

Definition 1. *Let L be a regular language. Then a representation $L = L_1 \cup L_2 \cup \cdots \cup L_k$ is called* a union-free decomposition *of L iff L_i is a union-free language for all $i = 1, \ldots, k$. The decomposition is called* minimal *iff there is no other union-free decomposition of L with fewer elements.*

Theorem 1 (K. Hashiguchi [9]). *Let L be a regular language, $T \subseteq \{\cdot, \cup, ^*\}$ be a subset of the rational language operations (concatenation, union, and star), and $M = \{M_1, \ldots, M_n\}$ be a finite set of regular languages. Then it is decidable whether L can be constructed from elements of M using a finite number of operations from T.*

As an immediate corollary we obtain that it is decidable whether a regular language L is union-free, by taking singleton languages as M and $T = \{\cdot, ^*\}$.

Let B be a subset of the set of states Q of the automaton $V = \langle \Sigma, Q, q_0, F, \varphi \rangle$. The set of words $\{x \in \Sigma^* \mid \forall q \in B, \varphi(q, x) \in B\}$ is denoted $\mathrm{str}(B)$.

Lemma 1 (J.A. Brzozowski, R. Cohen [2]). *Let $B \subseteq Q$. Then* $\mathrm{str}(B)$ *is a regular star language.*

3 Union-Free Languages

In this section some common properties of union-free languages are studied. In particular, we present an algorithm for checking whether a regular language is union-free.

First, we adduce an example of a union-free language. Its associated finite automaton is shown in Fig. 1(a). We believe that it is not a simple task to recognize a union-free language by looking at the automaton. For example, the well-known Kleene algorithm constructs a regular expression that contains three union operations on the top level. The language can be represented as $M = S_1^* b S_2^* a S_3^*$ where S_1, S_2, and S_3 are shown in Fig. 1(b), 1(c), and 1(d), respectively (the initial states of these automata are marked by 1).

In this section we assume that $\Sigma = \{a_1, \ldots, a_n\}$ is a finite alphabet, $L \subseteq \Sigma^*$ is a regular language and $V = \langle \Sigma, Q, q_0, F, \varphi \rangle$ is its associated deterministic finite automaton.

Proposition 1. *Let L be a union-free language. Then* $\mathrm{tsw}(L)$ *is meaningfully defined, i.e., if u and v are shortest words in L then $u = v$.*

Proof. Suppose $u = u_1 \cdots u_l$, $v = v_1 \cdots v_l$. Consider $GF(L)$. It should have the following form:

$$L = S_{01}^* S_{02}^* \cdots S_{0k_0}^* u_1 S_{11}^* \cdots S_{1k_1}^* u_2 \cdots S_{l-1,1}^* \cdots S_{l-1,k_{l-1}}^* u_l S_{l,1}^* \cdots S_{l,k_l}^*.$$

Since $v \in L$ and length of v is equal to that of u, $v_i = u_i$ for $i = 1, \ldots, l$, hence $u = v$. □

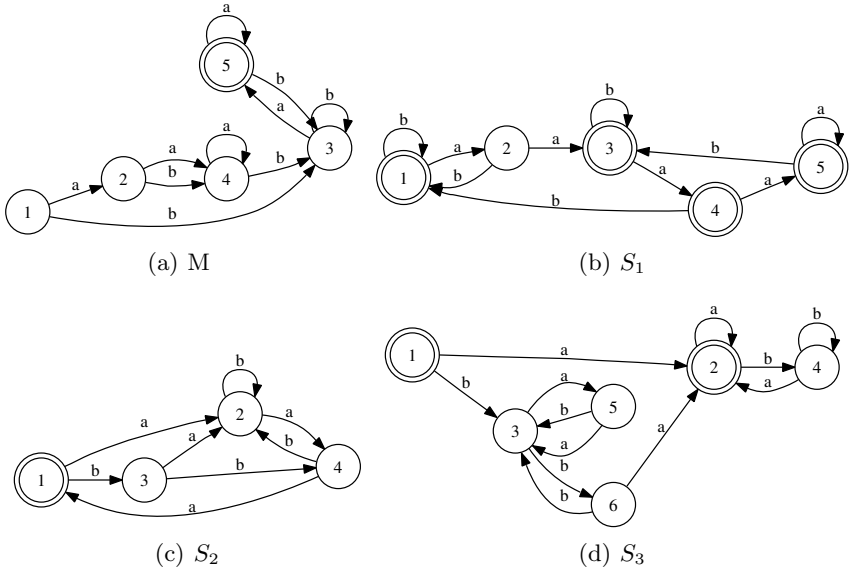


Fig. 1. Example of the union-free language $M = S_1^* b S_2^* a S_3^*$

Remark 1. Obviously, the word $u_1 \cdots u_l$ in the general form of a union-free language L is equal to $\mathrm{tsw}(L)$.

Definition 2. *Let* $2^Q = \{B_1, \ldots, B_k\}$. *We denote*

$$B(L) = \{\mathrm{str}(B_1), \ldots, \mathrm{str}(B_k)\}.$$

By Lemma 1, $B(L)$ is the finite set of regular star languages that can be constructed for every regular language. We now show that for every representation of a subset of L as a product of a prefix language, a star language and a suffix language the star language can be replaced with a language from $B(L)$. Thus we can extend every subset of L by replacing all star languages within it with star languages from the fixed set $B(L)$.

Lemma 2. *Let* $M \subseteq L$. *Then for every representation* $M = PR^*T$ *there exists a language* $D \in B(L)$ *so that* $M \subseteq PDT \subseteq L$.

Proof. First it should be noted that we do not consider the automaton associated with M and work only within the automaton for L.

Given a representation $M = PR^*T$ we define

$$G = \{q \in Q \mid \exists\, w \in P,\ \varphi(q_0, w) = q\}.$$

Then we denote $\widehat{G} = \{q \in Q \mid \forall w \in T\ \ \varphi(q, w) \in F\}$. Obviously, $G \subseteq \widehat{G} \subseteq Q$. We define $D = \mathrm{str}(\widehat{G})$. Taking any words $p \in P$ and $r \in R^*$, we obtain that $\varphi(q_0, p) \in G \subseteq \widehat{G}$ and $\varphi(q_0, pr) \in \widehat{G}$, because $\varphi(prt) \in F$ for all $t \in T$. This means that $\varphi(q, r) \in \widehat{G}$ for all $q \in \widehat{G}$, $r \in R^*$. Hence, $R^* \subseteq D \in B(L)$ and $M \subseteq PDT$. $PDT \subseteq L$, because we extend the language R^* to the language D working within the same unmodified automaton for L and therefore cannot obtain a language that contains more words than L does. □

Corollary 1. *For every representation* $L = PR^*T$ *there exists a language* $D \in B(L)$ *so that* $L = PDT$.

Proof. We consider $M = L$ and apply Lemma 2. Then $L \subseteq PDT \subseteq L$ hence $L = PDT$. □

Definition 3. *We denote* $C(L)$ *a set of all maximal finite concatenations of languages from* $B(L)$ *and letters such that every concatenation is a subset of* L. *Maximal means that if* C_1, C_2 *are such finite concatenations and* $C_2 \subseteq C_1$ *then we include only the language* C_1 *in* $C(L)$.

Lemma 3. *Let* $M \subseteq L$, $B_1 \subseteq Q$, $B_2 \subseteq Q$ *and* $M = P\,\mathrm{str}(B_1)\,\mathrm{str}(B_2)T$, *where* P *and* T *are regular languages. Then* $\varphi(q_0, P\,\mathrm{str}(B_1)) \subset \varphi(q_0, P\,\mathrm{str}(B_1)\,\mathrm{str}(B_2))$ *or there exists* $B_3 \subseteq Q$ *such that* $M \subseteq P\,\mathrm{str}(B_3)T \subseteq L$.

Proof. We denote three sets of states: $D_1 = \varphi(q_0, P)$, $D_2 = \varphi(q_0, P\,\mathrm{str}(B_1))$, and $D_3 = \varphi(q_0, P\,\mathrm{str}(B_1)\,\mathrm{str}(B_2))$. Since $\mathrm{str}(B_1)$ and $\mathrm{str}(B_2)$ contain the empty word, $D_1 \subseteq D_2 \subseteq D_3$. We also obtain that $\mathrm{str}(B_1) \subseteq \mathrm{str}(D_2)$, because the

set $\varphi(q_0, P\,\mathrm{str}(B_1)\,\mathrm{str}(B_1)) = \varphi(q_0, P\,\mathrm{str}(B_1)) = D_2$. Suppose $D_2 = D_3$. This means that $\mathrm{str}(B_2) \subseteq \mathrm{str}(D_2)$. Therefore, $\mathrm{str}(B_1)\,\mathrm{str}(B_2) \subseteq \mathrm{str}(D_2)$ and $M = P\,\mathrm{str}(B_1)\,\mathrm{str}(B_2)T \subseteq P\,\mathrm{str}(D_2)T$. Finally, we take $B_3 = D_2$. $P\,\mathrm{str}(B_3)T \subseteq L$ because we have not modified the automaton for L and still work within it. □

Lemma 4. *$C(L)$ is a finite set.*

Proof. Every element in $C(L)$ is a concatenation of star languages and letters. As already mentioned, all the letters concatenated form the shortest word in the language represented by the concatenation. First we limit the number of letters in every concatenation by $|Q| - 1$. We do that as follows: if a concatenation $L_i \in C(L)$ contains more than $|Q| - 1$ letters, we show that there is a language $M_i \in C(L)$ such that $L_i \subseteq M_i$, and we come to a contradiction with the definition of $C(L)$. We show how to effectively construct the language M_i given the language L_i. Suppose $L_i \in C(L)$ and its general form contains more than $|Q| - 1$ letters. This means that $\mathrm{tsw}(L_i)$ contains cycles in the automaton for L. Then $\mathrm{tsw}(L_i) = u_1 v_1 \cdots u_h v_h$, where $u_j \in \Sigma^*$, $v_j \in \Sigma^+$ and every $v_j = v_{j_1} \cdots v_{j_{l_j}}$ $(j = 1, \ldots, h)$ represents a cycle in the path $u_1 v_1 \cdots u_h v_h$ in the minimal automaton for L (and every u_j does not contain any cycles). This means that $L_i = L_{u_1} L_{v_1} \cdots L_{u_h} L_{v_h}$ where languages L_{u_j} and L_{v_j} are parts of the general form of L_i corresponding to the words u_j, v_j, respectively. For example, $L_{v_1} = v_{11} S_{p1} \cdots S_{pk_p} v_{12} S_{p+1,1} \cdots S_{p+1,k_{p+1}} v_{13} \cdots v_{1l_1}$. Then we define the language M_i as $M_i = L_{u_1}(L_{v_1})^* \cdots L_{u_h}(L_{v_h})^*$. First, $L_i \subset M_i$. Second, $\mathrm{tsw}(M_i) = u_1 \cdots u_h$ and $u_1 \cdots u_h$ represents a cycle-free path in L. Third, $M_i \subseteq L$, because it has been constructed within the automaton for L. This means that $L_i \subset M_i \subseteq L$ and we come to a contradiction, since $C(L)$ contains only maximal languages.

Now we prove that there is only a limited number of star languages between every pair of adjacent letters in every concatenation $M \in C(L)$. For every representation $M = P\,\mathrm{str}(B_1) \cdots \mathrm{str}(B_k)T$ we apply Lemma 3 and obtain that either $\varphi(q_0, P\,\mathrm{str}(B_1)) \subset \varphi(q_0, P\,\mathrm{str}(B_2)) \subset \ldots \subset \varphi(q_0, P\,\mathrm{str}(B_k))$ or we can replace the language M with the language M' such that $M \subseteq M'$ and $M' = P\,\mathrm{str}(D_1) \cdots \mathrm{str}(D_l)T$ and

$$\varphi(q_0, P\,\mathrm{str}(D_1)) \subset \varphi(q_0, P\,\mathrm{str}(D_2)) \subset \ldots \subset \varphi(q_0, P\,\mathrm{str}(D_l)).$$

In this case $M \notin C(L)$. We conclude that every element in $C(L)$ can be written as a concatenation with not more than $|Q| - 1$ star languages between every pair of adjacent letters. These two limitations complete the proof. □

Corollary 2. *Let $M \in C(L)$. Then* $\mathrm{tsw}(M)$ *represents a cycle-free path in the automaton associated with L.*

Proof. Since $M \subseteq L$, $\mathrm{tsw}(M) \in L$. Suppose $\mathrm{tsw}(M)$ contains cycles in the automaton associated with L. Then applying the procedure described in the proof of Lemma 4 (which constructs the language M_i using the language L_i), we obtain a language $M' \in C(L)$ such that $M \subset M'$. This means that $M \notin C(L)$ and we get a contradiction. □

Corollary 3. $|C(L)| \leqslant c|Q|\left(2^{|Q|}\right)^{|Q|-1}$, *where* c *is the number of cycle-free paths in the automaton associated with* L.

Proof. First, we fix a cycle-free path in the automaton associated with L (c possibilities). Then we fix a position of star languages: since there are not more than $|Q|-1$ letters, we have $|Q|$ possibilities (because star languages can appear before the first letter and after the last one). Then we choose not more than $|Q|-1$ languages from $B(L)$ (every language can appear more than once), having $\left(2^{|Q|}\right)^{|Q|-1}$ possibilities. Finally, we multiply all three expressions and come to the statement of the corollary. □

Remark 2. In order to construct $C(L)$ given a language L, we simply take all possible concatenations that contain letters that being concatenated form cycle-free paths in the automaton for L and that contain not more than $|Q|-1$ languages from $B(L)$ between every pair of letters. Finally, we exclude the languages that are not subsets of L and the languages that are subsets of other languages from the set.

Lemma 5. *Let* L *be a regular language and* $M \subseteq L$ *be a union-free language. Then there exists a language* $C_M \in C(L)$ *such that* $M \subseteq C_M$.

Proof. We take every star language $S_{i,j}^*$ from the general form of M. Thus $M = PS_{i,j}^*T$ where

$$P = S_{01}^* \cdots S_{0k_0}^* a_1 \cdots S_{i1}^* \cdots S_{i,j-1}^*$$

and $T = S_{i,j+1}^* \cdots S_{i,k_i} a_{i+1} \cdots a_l S_{l,1}^* S_{l,2}^* \cdots S_{l,k_l}^*$. Then we apply Lemma 2 and derive that $M \subseteq P\,\mathrm{str}(B_k)T$ where $B_k \in B(L)$. Thus we extended the "unknown" language $S_{i,j}^*$ to the known language $\mathrm{str}(B_k)$ from the set $B(L)$. After applying the procedure of extension to each star language in the general form for M, we get a language C_M that is a finite concatenation of languages from $B(L)$ and letters and also an extension of M. Hence $M \subseteq C_M$ and $C_M \in C(L)$. □

Theorem 2. *Let* L *be a regular language. Then* L *is a union-free language iff* $L \in C(L)$.

Proof

Necessity. We consider $M = L$ and apply Lemma 5. Then there exists a language $C_L \in C(L)$ such that $L \subseteq C_L$. But since all languages from $C(L)$ are subsets of L, $L = C_L$ and hence $L \in C(L)$.

Sufficiency. Suppose $L \in C(L)$ and L is a non-union-free language. Then it cannot be represented as a finite concatenation from $C(L)$ because every concatenation from $C(L)$ only consists of union-free languages (languages from $B(L)$ and letters), and we come to a contradiction. □

4 Union-Free Decomposition

Theorem 3. *Let* L *be a regular language. Then there exists an algorithm that results in a minimal union-free decomposition of* L: $L = L_1 \cup L_2 \cup \cdots \cup L_k$ *(the algorithm is described within the proof).*

Proof. To construct a minimal union-free decomposition, we examine all the subsets of $C(L)$ and choose the subset containing a minimum number of languages (among all the subsets) which being added up are equal to L. It should be noted that there is at least one subset containing languages which being added up are equal to L, because there exists at least one union-free decomposition of L and to every component of the decomposition we can apply Lemma 5, thus obtaining a decomposition of L into languages from $C(L)$. The final step is to prove that the decomposition obtained is minimal, i.e., there exists no decomposition containing fewer elements than the one we got. Suppose we have such a decomposition $L = N_1 \cup N_2 \cup \cdots \cup N_p$, $p < k$. We take each language N_i $(i = 1, \ldots, p)$ and apply Lemma 5 to it, getting a union-free language $C_{N_i} \in C(L)$ such that $N_i \subseteq C_{N_i}$. Thus we get the new decomposition $L = C_{N_1} \cup C_{N_2} \cup \cdots \cup C_{N_p}$, $p < k$ and every language C_{N_i} belongs to the set $C(L)$. But since we have already examined all the subsets of $C(L)$, we have examined the subset $\{C_{N_1}, \ldots, C_{N_p}\}$ too, and we must have chosen this subset for the minimal decomposition. This contradiction completes the proof. □

Unfortunately, the algorithm for constructing a minimal union-free decomposition of a given regular language L is computationally expensive since it requires checking all the subsets of the set $C(L)$, which can contain up to $c|Q| \left(2^{|Q|}\right)^{|Q|-1}$ elements, where c is the number of cycle-free paths in the automaton associated with L (see Corollary 3). We believe that there exist more effective algorithms that result in a minimal union-free decomposition of a given regular language. Some ideas on creating such an algorithm are given below.

A promising way of constructing minimal union-free decompositions can be developed using the technique of cutting maximum star languages introduced in [2]. In short, the technique is as follows. Let L be a regular language. The equation $L = X^*L$ is proved to have the unique maximal solution X_0 (w.r.t. inclusion). Moreover, the equation $L = X_0^*Y$ is proved to have the unique minimal solution Y_0. To construct a minimal union-free decomposition of L we solve these two equations and obtain the language Y_0. Then we apply the same procedure to the language Y_0 and get the minimal language Y_1 such that $L = X_0^*X_1^*Y_1$. If the process ends (and it is an open problem, see [10]) we either obtain the language $Y_m = \{\varepsilon\}$ (and thus the union-free decomposition $L = X_0^*X_1^* \cdots X_m^*$) or get a language Y_m such that the equation $Y_m = X^*Y_m$ has no non-trivial solutions. In the latter case we check whether all the words in the language Y_m start with the same letter. If it is the case and, for example, all the words in Y_m start with a, we write $Y_m = aY'_m$, $L = X_0^*X_1^* \cdots X_m^*aY'_m$ and apply the procedure described above to the language Y'_m (solve the equation $Y'_m = X^*Y'_m$ etc.). If it is not, and there are words in Y_m that start with different letters, e.g. $a_1, \ldots, a_n$, we can write $Y_m = Y_{m_1} \cup \cdots \cup Y_{m_n}$ so that every language $Y_{m_1}, \ldots, Y_{m_n}$ contains only words starting with the same letter a_i, $1 \leqslant i \leqslant n$. Then we write $L = X_0^*X_1^* \cdots X_m^*Y_{m_1} \cup X_0^*X_1^* \cdots X_m^*Y_{m_2} \cup \cdots \cup X_0^*X_1^* \cdots X_m^*Y_{m_n}$ and apply the procedure described above to every language $Y_{m_1}, \ldots, Y_{m_n}$. If the process ends, thus we obtain the union-free decomposition of the language L, which is likely to be minimal, but this is yet to be proved. As already mentioned, an-

other open problem connected with this technique is that the described process of "cutting stars" has not yet been proved to always be finite (see [10]).

5 Conclusions and Further Work

In this paper we have presented an algorithm for constructing a minimal union-free decomposition of a regular language. The algorithm includes an exhaustive search but we tend to think that there exist more effective algorithms that solve the problem.

We have also studied some common properties of union-free languages. In particular, we have presented the new algorithm for checking whether a given language is union-free which can be more effective than the one existing in the field (see [9,7]).

There are some other interesting questions connected with the problems considered. For instance, whether languages that form a minimal union-free decomposition are pairwise disjoint (as sets of words). If it is not always the case, one can consider minimal union-free decompositions that consist of pairwise disjoint languages and ways of constructing them.

Another open problem is connected with star height. Given a star height of a regular language is it possible to construct a minimal union-free decomposition that consists of languages of the same star height?

Acknowledgements. The authors would like to thank the anonymous reviewers for valuable comments. The first author would also like to thank Benedek Nagy for drawing attention to the problem.

References

1. Paz, A., Peleg, B.: On concatenative decompositions of regular events. IEEE Transactions on Computers 17(3), 229–237 (1968)
2. Brzozowski, J., Cohen, R.: On decompositions of regular events. Journal of the ACM 16(1), 132–144 (1969)
3. Nagy, B.: A normal form for regular expressions. In: Eighth International Conference on Developments in Language Theory, CDMTCS Technical Report 252, CDMTCS, Auckland, pp. 51–60 (2004)
4. Brzozowski, J.: Regular expression techniques for sequential circuits. PhD thesis, Princeton University, Princeton, New Jersey (1962)
5. Ehrenfeucht, A., Zeiger, P.: Complexity measures for regular expressions. In: STOC 1974: Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, pp. 75–79. ACM, New York (1974)
6. Afonin, S., Khazova, E.: Membership and finiteness problems for rational sets of regular languages. International Journal of Foundations of Computer Science 17(3), 493–506 (2006)
7. Leung, H., Podolskiy, V.: The limitedness problem on distance automata: Hashiguchi's method revisited. Theoretical Computer Science 310(1–3), 147–158 (2004)

8. Nagy, B.: Programnyelvek elemeinek szintaktikus lersa norml formban (syntactic description of the elements of the programming languages in a normal form). In: IF 2005, Conference on Informatics in Higher Education, Debrecen (2005)
9. Hashiguchi, K.: Representation theorems on regular languages. Journal of Computer and System Sciences 27, 101–115 (1983)
10. Brzozowski, J.: Open problems about regular languages. In: Book, R.V. (ed.) Formal Language Theory, Santa Barbara, CA, Univ. of CA at Santa Barbara, pp. 23–47. Academic Press, New York (1980)
11. Nagy, B.: Union-free languages and 1-cycle-free-path-automata. Publicationes Mathematicae Debrecen 68, 183–197 (2006)

Commutative Regular Shuffle Closed Languages, Noetherian Property, and Learning Theory

Yohji Akama

Mathematical Institute, Tohoku University, Sendai Miyagi, Japan, 980-8578
(Japan Science and Technology Agency)
akama@m.tains.tohoku.ac.jp

Abstract. To develop computational learning theory of commutative regular shuffle closed languages, we study finite elasticity for classes of (semi)group-like structures. One is the class of $A\mathbb{N}^d + F$ such that A is a matrix of size $e \times d$ with nonnegative integer entries and F consists of at most k number of e-dimensional nonnegative integer vectors, and another is the class $\mathcal{X}_k^d$ of $A\mathbb{Z}^d + F$ such that A is a square matrix of size d with integer entries and F consists of at most k number of d-dimensional integer vectors (F is repeated according to the lattice $A\mathbb{Z}^d$). Each class turns out to be the elementwise unions of k-copies of a more manageable class. So we formulate "learning time" of a class and then study in general setting how much "learning time" is increased by the elementwise union, by using Ramsey number. We also point out that such a standpoint can be generalized by using Noetherian spaces.

1 Introduction

For words u, v on an alphabet Σ, the *shuffle product* of u and v is, by definition,

$$u \diamond v = \{\, w \ : \exists n \geq 1 \exists u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n \in \Sigma^* \\ w = u_1 v_1 u_2 v_2 ... u_n v_n,\ u = u_1 u_2 \cdots u_n \text{ and } v = v_1 v_2 \cdots v_n \,\}$$

A language L is said to be *shuffle closed* if it is closed under shuffle product. L is said to be *commutative* if $xuvy \in L \iff xvuy \in L$ for all $u, v, x, y \in \Sigma^*$. Let $\mathbb{N}$ be $\{0, 1, 2, \ldots\}$. *Parikh mapping* is $\Psi : \Sigma^* \to \mathbb{N}^{\#\Sigma}$ such that $\Psi(w) = (n_1, \ldots, n_r)$ where n_i is the number of occurrences of s_i in w. $\Psi(L)$ is the set of $\Psi(w)$ such that $w \in L$. According to [1], for every commutative regular shuffle closed language L, $\Psi(L) \subseteq \mathbb{N}^{\#\Sigma}$ is a semigroup [1], and if L moreover contains the empty word ϵ, then

$$L = \Psi^{-1}(\bigcup_{i=1}^{n} A_i \mathbb{N}^{\#\Sigma} + f_i) \tag{1}$$

for some $n \in \mathbb{N}$, some square matrices A_i's of size $\#\Sigma$ with nonnegative integer entries and some $\#\Sigma$-dimensional nonnegative integer vectors f_i's. Thus computational learning theory for commutative regular shuffle closed languages is

[1] In this paper, semigroups are always understood commutative.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 93–104, 2009.

related to that for semigroups, and conditions where the learnabilities of each of two classes $\mathcal{C}_1, \mathcal{C}_2$ implies that of their elementwise union

$$\mathcal{C}_1 \,\widetilde{\cup}\, \mathcal{C}_2 := \{L_1 \cup L_2 \ ; \ L_1 \in \mathcal{C}_1, L_2 \in \mathcal{C}_2\}.$$

A typical learning process of semigroups is as follows: "Teacher" chooses a positive integer m and presents all the elements of a semigroup $\langle m \rangle := \{nm \ ; \ n = 0, 1, 2, 3, \ldots\}$ one by one to "Learner", where "Learner" submits a hypothesis for m whenever he receives a number from "Teacher." If the hypotheses $h_1, h_2, \ldots$ converge to m, then we say that "Learner" learns m. But for each updating of hypothesis, "Learner" cannot know whether his hypothesis is true or not. If "Learner" always submits a hypothesis which respect all the numbers presented so far, then he learns m because an ascending chain $\langle h_1 \rangle \subseteq \langle h_2 \rangle \subseteq \cdots$ is eventually stationary (i.e. $\langle h_t \rangle = \langle h_{t+1} \rangle = \cdots$ for some t). In other words, "Learner" can learn $\langle m \rangle$ because of *Noetherian property* and the "learning time" $t = \#\{i \ ; \ h_{i-1} \neq h_i\}$ is bounded by the length of a strictly ascending chain from $\langle m \rangle$. However Noetherian property [2,3] for semigroups, which is defined after Noetherian property for semigroup-rings, happens to be unsatisfactory from computational learning viewpoint, because it is satisfied by a semigroup $(\mathbb{N}^2, +)$ which turns out to have an infinite strictly ascending chain of subgroups in $\mathbb{N}^2$. In computational learning theory, a property similar to Noetherian property was independently introduced as a sufficient condition for a class of languages to be learnable [4]. The property is called a *finite elasticity*. The computational learning theory for Noetherian rings and similar algebraic structures are developed by Ventsov and Stephan [5]. We develop learning theory for algebraic structures and language classes by taking following proposition seriously:

Proposition 1 (Motoki-Shinohara-Wright [4]). *If two classes $\mathcal{C}_1, \mathcal{C}_2$ of sets have finite elasticities, so does $\mathcal{C}_1 \,\widetilde{\cup}\, \mathcal{C}_2$.*

By using the Proposition, the finite elasticity was proved for the class of extended pattern languages (e.g. $\{ww \ ; \ w \in \Sigma^*\}$ is an extended pattern language), and the same property will be proved for the class of commutative regular shuffle closed languages in this paper. Also we study finite elasticity for classes of (semi)group-like structures, such as a class $Lattice_d$ of d-dimensional integer lattices and a class

$$\mathcal{X}_k^d := \{M + F \ ; \ M \in Lattice_d, \ F \subseteq \mathbb{Z}^d, \#F \leq k\} \quad (k \geq 1) \tag{2}$$

of *integer crystalline structures* (F is repeated by a lattice M like a physical crystalline material). The former class $Lattice_d$ is studied from viewpoint of another computational learning setting [6]. The latter class $\mathcal{X}_k^d$ is a subclass of the elementwise union of k-copies of more manageable classes. So we formulate "learning time" and then evaluate from upward the "learning time" of the elementwise union in terms of those of the constituents classes, by using Ramsey number $R(l, m, 2)$. We also point out that such a standpoint can be generalized by using Noetherian spaces.

2 Commutative Regular Shuffle Closed Languages

For $\alpha, \beta, p \in \mathbb{N}^r$, we write $\alpha \geq \beta$ if each component of α is not less than the corresponding component of β, and we write $\alpha \overrightarrow{\equiv} \beta \pmod{p}$ if $\alpha \geq \beta$ but each component of α is congruent to the corresponding component of β modulo the corresponding component of p.

Proposition 2 ([1, Proposition 5.3.4, 5.3.5]). *Let $L \subseteq \Sigma^*$ such that L contains the empty word ϵ. Then L is a commutative regular shuffle closed language if and only if L is represented as*

$$L = \bigcup_{u \in F} \Psi^{-1}\Psi\left(u\left(a_1^{\delta_{u1}p_1}\right)^* \cdots \left(a_r^{\delta_{ur}p_r}\right)^*\right) \tag{3}$$

where

1. *$Y = \{a_1, \ldots, a_r\}$ is an r-size subset of Σ for some positive integer r, F is a finite language over Y with $\epsilon \in F$, and $\delta_u = (\delta_{u1}, \ldots, \delta_{ur})$ belongs to $\{0,1\}^r$ for any $u \in F$. Moreover,*
2. *$p = (p_1, \ldots, p_r)$ belongs to $\{1, 2, \ldots.\}^r$,*
   (a) *there exists $q \in \mathbb{N}^r$ such that for all $u \in F$ $q \geq p$ and $q \geq \Psi(u)$, and*
   (b) *for any $u, v \in F$, there is $w \in F$ such that $\delta_u, \delta_v \leq \delta_w$ and $\Psi(uv) \overrightarrow{\equiv} \Psi(w) \pmod{(\delta_{w1}p_1, \ldots, \delta_{wr}p_r)}$.*

For the class of commutative regular shuffle closed languages with the shuffle base being finite, the characterization is similar but δ_u's are always $(1, \ldots, 1)$. Here shuffle base *of L is defined as $(L \setminus \{\epsilon\}) \setminus \big((L \setminus \{\epsilon\}) \diamond (L \setminus \{\epsilon\})\big)$.*

By above proposition, every commutative regular shuffle closed language is written as (1), while every commutative regular shuffle closed languages with finite shuffle base is written as $\Psi^{-1}(A\mathbb{N}^{\#\Sigma} + F)$ for some square matrix A of size $\#\Sigma$ with nonnegative integer entries and for some finite set $F \subset \mathbb{N}^{\#\Sigma}$. Here '+' stands for *Minkowski sum(i.e., elementwise addition).*

Definition 1. *For any positive integer k, let $\mathcal{CRS}_k$ be the class of commutative regular shuffle closed languages L such that $L \ni \epsilon$ and $\#F$ of (3) is less than or equal to k. Let $f\mathcal{CRS}_k$ be the class of $L \in \mathcal{CRS}_k$ such that the shuffle base of L is finite.*

3 Computational Learning Theory: Finite Elasticity and Wright's Theorems

In this section, we review *finite elasticity*, a sufficient condition for a class to be learnable, formulate the "learning time," and then evaluate how much "learning time" is increased by elementwise union by using Ramsey numbers.

First of all, we review one of basic settings of computational learning theory. An *indexed family of recursive language(i.f.r.l.)* is any $\mathcal{C} = \{L_i \ ; \ i \in \mathbb{N}\}$ such that there is a computable function $f : \mathbb{N} \times \Sigma^* \to \{0,1\}$ where $f(i,w) = 1$ if

$L_i \ni w$ and $f(i,w) = 0$ otherwise. We say that this i is a *representative* of L_i or i *explains* L_i. For example, a class $\{L(G)\ ;\ G$ is a regular grammar$\}$ and the class $\left\{\{nm\ ;\ n \in \mathbb{N}\right\}\ ;\ m \in \mathbb{N}\}$ are indexed families of recursive languages. We say that an i.f.r.l. $\mathcal{C}$ is *learnable from positive data* by an algorithm A if for any $L \in \mathcal{C}$ and for any enumeration $x_1, x_2, \ldots$ of L, the algorithm A computes a hypothesis h_n for L on each initial segment $x_1, \ldots, x_n$ $(n \geq 1)$ and the hypotheses $h_1, h_2, \ldots$ tends to a representative of L in the limit of n. An algorithm said to be *consistent*, if $L_{h_n} \supseteq \{x_1, \ldots, x_n\}$ for every n.

The number of mind-changes taken by A with respect to the presentation $x_1, x_2, \ldots$ is defined as $\#\{i\ ;\ h_{i-1} \neq h_i\}$. The maximum over L and the presentations of L is called the mind-change complexity of $\mathcal{C}$ by A, and the maximum over any consistent algorithm that learns $\mathcal{C}$ is called the *mind-change complexity for* $\mathcal{C}$. For example, let us consider the case where "Teacher" teaches "Learner" multiples but "Teacher" always begins with presenting a number x_1 less than n as an example of a multiple, while "Learner" always submits a hypothesis consistent with numbers presented so far. Then the hypotheses $h_1, h_2, \ldots$ submitted by "Learner" are such that each h_i divides h_{i+1}, so the number of the mind-change taken by "Learner" is at most $\log_2 n$. By a similar reasoning, the mind-change complexity of $\{\langle m \rangle \cap [0,n]\ ;\ m \in \mathbb{N}\}$ is at most $\log_2 n$. "Learner" is said to be *consistent*, if every hypothesis "Learner" submits is consistent with data presented so far.

Let $\mathbb{N}_* = \{0 < 1 < 2 < \cdots < n < \cdots < *\}$.

Definition 2 (finite elasticity, dimension). *We say a class $\mathcal{C}$ of sets has an* elasticity *of length $\alpha \in \mathbb{N}_*$, if*

$$\exists\{L_n \in \mathcal{C}\ ;\ 1 \leq n < \alpha\}\ \exists\{t_\gamma\ ;\ 0 \leq \gamma < \alpha\}$$
$$\forall\beta.\ 1 \leq \beta < \alpha\ \Rightarrow\ \{t_\gamma\ ;\ 0 \leq \gamma < \beta\} \subseteq L_\beta \not\ni t_\beta.$$

We say $\mathcal{C}$ has an infinite elasticity, *if $\alpha = *$. Otherwise, we say $\mathcal{C}$ has a* finite elasticity. *When and only when $\mathcal{C}$ has finite elasticity, we define the* dimension $\dim(\mathcal{C}) \in \mathbb{N} \cup \{\infty\}$ *as the supremum of α such that $\mathcal{C}$ has an elasticity of length $\alpha < *$, that is,*

$$\dim(\mathcal{C}) := \sup\left\{\alpha\ ;\ \begin{array}{l}\exists\{L_n \in \mathcal{C}\ ;\ 1 \leq n < \alpha\}\ \exists\{t_\gamma\ ;\ 0 \leq \gamma < \alpha\} \\ \forall\beta.\ 1 \leq \beta < \alpha\ \Rightarrow\ \{t_\gamma\ ;\ 0 \leq \gamma < \beta\} \subseteq L_\beta \not\ni t_\beta\end{array}\right\}.$$

Remark 1. If $\mathcal{C}$ has a finite elasticity, then for any pair of "Teacher" and "Consistent Learner," in a following game between them, the "Consistent Learner" always wins after submitting at most $\dim(\mathcal{C})$ numbers of elements of $\mathcal{C}$.

In the course of the game, the "Teacher" presents a datum w while the "Consistent Learner" chooses from $\mathcal{C}$ an element L that contains the data presented so far. "Teacher" tries to present a datum that the "Consistent Learner" can no more choose such an element from $\mathcal{C}$. "Consistent Learner" wins if the "Teacher" can no more present such a datum. We can easily see that $\mathcal{C}$ has a finite elasticity if "Consistent Learner" can always win.

$$\begin{array}{lccccc} \text{“Teacher”} & t_0 & t_1 & t_2 \cdots & & t_{\alpha-1} \\ \text{“Consistent Learner”} & L_1 & L_2 & \cdots & L_{\alpha-1} & \end{array}$$

Clearly, if $\mathcal{C}$ has a finite elasticity then so does any subclass of $\mathcal{C}$.

Proposition 3 ([4]). *For every i.f.r.l. $\mathcal{C}$, if $\mathcal{C}$ has a finite elasticity, it is learnable from positive data.*

Theorem 1. *For any i.f.r.l. $\mathcal{C}$, if $\dim(\mathcal{C}) < \infty$, then $\mathcal{C}$ is learnable from positive data by a consistent learning algorithm and the mind-change complexity of $\mathcal{C}$ is bounded from above by $\dim(\mathcal{C})$.*

Now let us derive an upper bound of $\dim(\mathcal{C} \mathbin{\widetilde{\cup}} \mathcal{D})$ from a Ramsey-type proof of Proposition 1. First recall *Ramsey* numbers.

Proposition 4 ([7, Section 4.2]). *For every $l, m \in \mathbb{Z}_{\geq 1}$ there exists k such that any edge-coloring with red and black for the complete graph of size k has either a red complete subgraph of size l or a black complete subgraph of size m. Such minimum k, denoted by $R(l, m, 2)$, is called a Ramsey number of l, m, and is bounded from above by $\binom{m+l-2}{l-1} \leq c4^{\max(l,m)}(\max(l,m))^{-1/2}$ for some constant c.*

Theorem 2. *For any classes $\mathcal{C}$ and $\mathcal{D}$, if $\dim(\mathcal{C})$ and $\dim(\mathcal{D})$ are finite, then*

$$\dim(\mathcal{C} \mathbin{\widetilde{\cup}} \mathcal{D}) < \dim(\mathcal{C}) \cdot \dim(\mathcal{D}) \cdot R(\dim(\mathcal{C}) + 1,\ \dim(\mathcal{D}) + 1,\ 2).$$

Proof. Suppose there are a sequence $t_0, \ldots, t_{n-1}$, a sequence $L_1, \ldots, L_{n-1}$ of $\mathcal{C}$ and a sequence $M_1, \ldots, M_{n-1}$ of $\mathcal{D}$ such that

$$\{t_0, \ldots, t_{i-1}\} \subseteq L_i \cup M_i \not\ni t_i \quad (1 \leq \forall i < n). \tag{4}$$

We have only to show that $n < \dim(\mathcal{C}) \cdot \dim(\mathcal{D}) \cdot R(\dim(\mathcal{C}) + 1, \dim(\mathcal{D}) + 1,\ 2)$. It is easy to see:

Fact 1

1. $L_{\nu_1} = \cdots = L_{\nu_k} \quad (1 \leq \nu_1 < \cdots < \nu_k < n) \ \Rightarrow\ k \leq \dim(\mathcal{D})$.
2. $M_{\nu_1} = \cdots = M_{\nu_k} \quad (1 \leq \nu_1 < \cdots < \nu_k < n) \ \Rightarrow\ k \leq \dim(\mathcal{C})$.

Proof. The first assertion is proved as follows: By (4),

$$\{t_{\nu_1}, \ldots, t_{\nu_{i-1}}\} \setminus L_{\nu_i} \subseteq M_{\nu_i} \not\ni t_{\nu_i} \quad (2 \leq \forall i \leq k).$$

The leftmost term $\{t_{\nu_1}, \ldots, t_{\nu_{i-1}}\} \setminus L_{\nu_i}$ is just $\{t_{\nu_1}, \ldots, t_{\nu_{i-1}}\}$. Otherwise there is a positive integer j less than i such that $t_{\nu_j} \in L_{\nu_i} = L_{\nu_j}$, which contradicts (4). The situation is illustrated as a following game between “Teacher” and “Consistent Learner”:

$$\begin{array}{lccccc} \text{“Teacher”} & t_{\nu_1} & t_{\nu_2} \cdots & t_{\nu_{k-1}} & & t_{\nu_k} \\ \text{“Consistent Learner”} & & M_{\nu_2} \quad \cdots & & M_{\nu_k} & \end{array} .$$

Therefore $k \leq \dim(\mathcal{D})$. The second assertion is proved similarly as the first assertion.

Fact 2. *If $L_i (i = 1, \ldots, n-1)$ is pairwise distinct and $M_i (i = 1, \ldots, n-1)$ is pairwise distinct, then $n < R(\dim(\mathcal{C}) + 1,\ \dim(\mathcal{D}) + 1,\ 2)$.*

Proof. Consider a complete graph G with the vertices being $0, \ldots, n-1$. For any edge $\{i, j\}(i \neq j)$, color it by red if $0 \leq i < j \leq n-1$ and $t_i \in L_j$, while color it by black otherwise.

Assume $n \geq R(\dim(\mathcal{C})+1,\ \dim(\mathcal{D})+1, 2)$. By Ramsey's theorem, the colored complete graph G has either a red clique of size $\dim(\mathcal{C})+1$ or a black clique of size $\dim(\mathcal{D})+1$. When a red clique of size $\dim(\mathcal{C})+1$ exists, write it as $\{u_0 < \cdots < u_{\dim(\mathcal{C})}\}$. Then we have $\{t_{u_0}, \ldots, t_{u_i}\} \subseteq L_{u_{i+1}} \not\ni t_{u_{i+1}} \quad (0 \leq \forall i < \dim(\mathcal{C}))$, which contradicts the definition of $\dim(\mathcal{C})$.

Otherwise, a black clique of size $\dim(\mathcal{D})+1$ exists, so we write it as $\{u_0 < \cdots < u_{\dim(\mathcal{D})}\}$. Then we have $\{t_{u_0}, \ldots, t_{u_i}\} \cap L_{u_{i+1}} = \emptyset \quad (0 \leq \forall i < \dim(\mathcal{D}))$. Because $(u_i)_i$ is in ascending order and (4) holds, we have $\{t_{u_0}, \ldots, t_{u_i}\} \subseteq L_{u_{i+1}} \cup M_{u_{i+1}} \not\ni t_{u_{i+1}}$, so $\{t_{u_0}, \ldots, t_{u_i}\} \subseteq M_{u_{i+1}} \not\ni t_{u_{i+1}} \quad (0 \leq i < \dim(\mathcal{D}))$, which contradicts the definition of $\dim(\mathcal{D})$. This completes the proof of Fact 2.

By the previous Fact 1, in the sequence (4), each L_i occurs at most $\dim(\mathcal{D})$ times and each M_i occurs at most $\dim(\mathcal{C})$ times, so we have the desired consequence of Theorem 2.

4 Finite Elasticity of Semigroups, Commutative Regular Shuffle Closed Languages and Pattern Languages

In this section we give an elementary proof of the finite elasticity of $\mathcal{CRS}_k$ of Definition 1 and that of extended pattern languages, by considering Parikh mapping and the class of finite generated semigroups of $\mathbb{N}^e$.

A monoid $(\mathbb{N}^2, +)$ does not have any infinite strictly ascending chain of submonoideals, by Dickson's Lemma. As a semigroup, $(\mathbb{N}^2, +)$ is Noetherian in a following sense:

Proposition 5 ([2],[3]). *For every semigroup S, following two conditions are equivalent: (a) S is finitely generated. (b) The lattice of congruences on S satisfies ascending chain condition with respect to inclusion. I.e., any ascending chain of congruences on S with respect to inclusion is eventually stationary.*

S is said to be Noetherian *if one of the two conditions (a) or (b) holds.*

But $(\mathbb{N}^2, +)$ has an infinite strictly ascending chain of sub-semigroups.

Theorem 3

1. *For any positive irrational r, $M_r := \{\, {}^t(x, y) \in \mathbb{N}^2 \ ;\ y < rx \,\}$ is a sub-semigroup of $(\mathbb{N}^2, +)$ but not finitely generated.*
2. *A semigroup $(\mathbb{N}^2, +)$ has an infinite strictly ascending chain of finitely generated free sub-semigroups.*

Proof

1. Assume M_r is finitely generated. Then it becomes $\begin{pmatrix} k_1^{(1)} & \cdots & k_n^{(1)} \\ k_1^{(2)} & \cdots & k_n^{(2)} \end{pmatrix} \mathbb{N}^n$ for some positive integer n and some $k_j^{(i)} \in \mathbb{N}$ $(i = 1, 2;\ j = 1, \ldots, n)$. So

for any ${}^t(x, y) \in M_r$ with $x \neq 0$, y/x is less than or equal to the $q := \max_{1 \leq j \leq n, k_j^{(1)} \neq 0} k_j^{(2)}/k_j^{(1)} \in \mathbb{N}$. But because r is irrational, there exist two positive integers x and y such that $q < y/x < r$, and $(x, y) \in M_r$. Contradiction.

2. M_r is written as an infinite set $\{\boldsymbol{a}_i\}_i \subset \mathbb{N}^2$ and has an ascending chain of sub-semigroups $S_1 \subseteq S_2 \subseteq S_3 \subseteq \cdots$ where $S_i := \langle \boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_i \rangle$ $(i = 1, 2, 3, \ldots)$. If $S_n = S_{n+1} = S_{n+2} = \cdots$, then M becomes S_n which is finitely generated. This contradicts the first assertion of the theorem.

We can easily construct a series of matrices $A_n = \begin{pmatrix} k_1^{(1)} & \cdots & k_n^{(1)} \\ k_1^{(2)} & \cdots & k_n^{(2)} \end{pmatrix}$ such that $A_n \mathbb{N}^n \subsetneq A_{n+1} \mathbb{N}^{n+1}$ from a continued fraction of an irrational number. However if the number of columns(i.e., the number of generators) is bounded, then there is no infinite strictly ascending chain of sub-semigroups of $(\mathbb{N}^2, +)$, as we see below.

Theorem 4. *For all positive integers e and d,*

$$\mathcal{SL}_{e,d} := \{A\mathbb{N}_d \ ; \ A \in \mathrm{M}_{e,d}(\mathbb{N})\} \qquad (\mathbb{N}_d := \mathbb{N}^{d-1} \times \{1\})$$

has a finite elasticity.

Proof. Assume that the set has an infinite elasticity, that is,

$$\begin{aligned} &\exists A_1, A_2, \ldots \in \mathrm{M}_{e,d}(\mathbb{N}).\ \exists \boldsymbol{w}_0, \boldsymbol{w}_1, \ldots \in \mathbb{N}^e. \\ &\forall n \geq 1. \{\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{n-1}\} \subseteq A_n \mathbb{N}_d \not\ni \boldsymbol{w}_n \ . \end{aligned} \tag{5}$$

For positive integers i, j and any matrix A and vector w, let $(A)_{i,j}$ be the i-th row j-th column component and $(w)_i$ be the i-th component. Clearly

$$\forall n \geq 1.\ (A_n)_{1,d} \leq (\boldsymbol{w}_0)_1 \tag{6}$$

We will derive a contradiction by induction on d.

If $d = 1$, then $A_n \mathbb{N}_d$ is a singleton for every n, so $\boldsymbol{w}_0 = \boldsymbol{w}_1 = \cdots$, which contradicts (5).

Consider the case where $d > 1$. A matrix obtained from $A \in \mathrm{M}_{e,d}(\mathbb{N})$ by removing the j-th column is denoted by ${}^j(A) \in \mathrm{M}_{e,d-1}(\mathbb{N})$.

Fact 3. $\forall k \in \mathbb{N} \exists \nu(k) > k \exists j(k) \in [1,\ d-1].\ \{\boldsymbol{w}_0, \ldots, \boldsymbol{w}_k\} \subset {}^{j(k)}\left(A_{\nu(k)}\right)\ \mathbb{N}_{d-1}$.

($\because$) Let $m(k) := \max\left\{(\boldsymbol{w}_l)_i \ ; \ 0 \leq l \leq k, 1 \leq i \leq e\right\} + 1$. Then there is $n > (m(k))^{d \times e}$ such that

$$\exists i \in [1, e] \exists j \in [1, d].\ m(k) \leq (A_n)_{i,\,j}. \tag{7}$$

Otherwise there are at most $(m(k))^{d \times e}$ number of A_n which components are all less than $m(k)$. But $\{A_1, A_2, A_3, \ldots\}$ is infinite, a contradiction. We can easily see that there are infinitely many n such that (7) holds.

But by (6) and the definition of $m(k)$, we have $j \leq d-1$. Let us choose an n such that (7) and $n > \max\Big(\big(m(k)\big)^{d\times e}, k\Big)$, and denote it by $\nu(k) > \max\Big(\big(m(k)\big)^{d\times e}, k\Big)$.

For any $l \in [0,k]$, if $\boldsymbol{w}_l$ is $A_{\nu(k)}\boldsymbol{u}$ for some $\boldsymbol{u} \in \mathbb{N}_d$ with $(u)_j \neq 0$, then by (7) and the definition of $m(k)$, we have $(\boldsymbol{w}_l)_i \geq (A_{\nu(k)})_{i,j} > (\boldsymbol{w}_l)_i$, a contradiction. So $(\boldsymbol{u})_j = 0$. This completes the proof of Fact 3.

Then we have a following contradiction to the induction hypothesis.

Fact 4. *There are an infinite series of e-dimensional vectors $\boldsymbol{w}_{\nu^i(0)}$ $(i \geq 0)$ and an infinite series of $e \times (d-1)$ integer matrices ${}^{j\left(\nu^{n-1}(0)\right)}\left(A_{\nu^n(0)}\right)$ $(n \geq 1)$ which cause an infinite elasticity: $\left\{\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{\nu^{n-1}(0)}\right\} \subseteq {}^{j\left(\nu^{n-1}(0)\right)}\left(A_{\nu^n(0)}\right)\ \mathbb{N}_{d-1} \not\ni \boldsymbol{w}_{\nu^n(0)}$. $(n \geq 1)$.*

($\because$) For $n \geq 1$, because $\nu(x) > x$ and (5), we have $\left\{\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{\nu^{n-1}(0)}\right\} \subseteq A_{\nu^n(0)}\mathbb{N}_d \not\ni \boldsymbol{w}_{\nu^n(0)}$. By Fact 3, $\left\{\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{\nu^{n-1}(0)}\right\} \subseteq {}^{j\left(\nu^{n-1}(0)\right)}\left(A_{\nu^n(0)}\right)\ \mathbb{N}_{d-1}$. On the other hand, because ${}^{j\left(\nu^{n-1}(0)\right)}\left(A_{\nu^n(0)}\right)\ \mathbb{N}_{d-1} \subseteq A_{\nu^n(0)}\mathbb{N}_d$, we have ${}^{j\left(\nu^{n-1}(0)\right)}A_{\nu^n(0)}\ \mathbb{N}_{d-1} \not\ni \boldsymbol{w}_{\nu^n(0)}$. This completes the proof of Fact 4.

Now let us return to the proof of the theorem. By induction hypothesis, $\mathcal{SL}_{e,d-1}$ has a finite elasticity, which contradicts Fact 4. So $\mathcal{SL}_{e,d}$ has a finite elasticity. Q.E.D.

Recall that $\mathcal{CRS}_k$ is the class of commutative regular shuffle closed languages L such that $L \ni \epsilon$ and $\#F$ of (3) is less than or equal to k.

Corollary 1. *For any positive integer k, $\mathcal{CRS}_k$ has a finite elasticity.*

Proof. For any subclass $\mathcal{C}$ of the powerset of $\mathbb{N}^{\#\Sigma}$, let $\Psi^{-1}(\mathcal{C})$ be $\{\Psi^{-1}(L)\ ;\ L \in \mathcal{C}\}$. Then $\mathcal{CRS}_1 = \Psi^{-1}(\mathcal{SL}_{\#\Sigma,\#\Sigma+1})$. By the previous theorem, $\mathcal{CRS}_1$ has a finite elasticity, because $w \in L \iff \Psi(w) \in \Psi(L)$ for any $L \in \Psi^{-1}(\mathcal{C})$. So the elementwise union of k-copies of $\mathcal{CRS}_1$ has a finite elasticity by Proposition 1, and contains $\mathcal{CRS}_k$. This prove the corollary.

The class $\mathcal{SL}_{1,v+1}$ of Theorem 4 is also important in establishing the finite elasticity of extended pattern languages.

Definition 3. *The* extended pattern language *of a pattern $p \in Pat^v_\Sigma := \big(\Sigma \cup \{x_1, \ldots, x_v\}\big)^*$ is the set* $\mathrm{eLang}_\Sigma(p)$ *of $\sigma(p)$ with σ being any homomorphism from Pat^v_Σ to Σ^* such that $\sigma(s) = s$ for every $s \in \Sigma$. Then $\mathcal{EPAT}^v_\Sigma$ is defined as the class of* $\mathrm{eLang}_\Sigma(p)$ *such that $p \in Pat^v_\Sigma$.*

For every $p \in Pat^v_\Sigma$, the set of length of w's in $\mathrm{eLang}_\Sigma(p)$ belongs to $\mathcal{SL}_{1,v+1}$, because p_n induces a matrix $A_n \in \mathrm{M}_{1,v+1}(\mathbb{N})$ such that $(A_n)_{1,v+1}$ is the number of occurrences of non-variable symbols in p_n while $(A_n)_{1,j}$ the number of occurrences of x_j in p_n.

Theorem 5 (Wright[8]). *For any $v \in \mathbb{N}$, $\mathcal{EPAT}^{v}_{\Sigma}$ has a finite elasticity.*

Proof. By induction on $v \geq 0$. We derive a contradiction from an assumption

$$\exists p_1, p_2, \ldots \in Pat^{d}_{\Sigma}\ \exists u_0, u_1, \ldots \in \Sigma^{*}$$
$$\forall n \geq 1. \{u_0, \ldots, u_{n-1}\} \subseteq \mathrm{eLang}_{\Sigma}(p_n) \not\ni u_n \ .$$

We can prove that

3' $\forall k \in \mathbb{N}\, \exists \nu(k) > k\, \exists j(k) \in [1, , d].\ \{u_0, \ldots, u_k\} \subseteq \mathrm{eLang}_{\Sigma}(p_{\nu(k)}[x_{j(k)} := \epsilon])$.
4' There are $\left\{u_{\nu^i(0)}\right\}_{i \geq 0} \subseteq \Sigma^{*}$ and $\left\{p_{\nu^n(0)}[x_{j(\nu^{n-1}(0))} := \epsilon]\right\}_{n \geq 1} \subseteq Pat^{d-1}_{\Sigma}$ that cause an infinite elasticity.

In this way, we can derive contradiction against the inductive hypothesis.

Proposition 6 ([9]). *Let $\#\Sigma \geq 2$. Then $\mathrm{eLang}_{\Sigma}(x_1x_1x_2x_2x_3x_3)$ does not have* tell-tale set *with respect to $\mathcal{EPAT}^{\infty}_{\Sigma} := \bigcup_{v \geq 0} \mathcal{EPAT}^{v}_{\Sigma}$, so $\mathcal{EPAT}^{\infty}_{\Sigma}$ is not learnable by Angluin [10]. Thus $\mathcal{EPAT}^{\infty}_{\Sigma}$ has an infinite elasticity.*

Interestingly, the class of finitely generated sub-semigroup of $(\mathbb{N}, +)$ has a finite elasticity.

Theorem 6. *For all positive integers e and d, $\left\{A\mathbb{N}^d \backslash \{\mathbf{0}\}\ ;\ A \in \mathrm{M}_{e,d}(\mathbb{N}), d \geq 1\right\}$ has a finite elasticity if and only if $e = 1$.*

Proof. (If part) Assume the contrary. Then, there are infinite sequences $\{d_i \in \mathbb{N}\ ;\ i > 0\}$, $\{A_i \in \mathrm{M}_{1,d_i}(\mathbb{N})\ ;\ i > 0\}$ and $\{w_i \in \mathbb{N}\ ;\ i \geq 0\}$ such that each $A_i\mathbb{N}^{d_i} \setminus \{0\}$ contains $w_0, \ldots, w_{i-1}$ but not $\boldsymbol{w}_i$. The free semigroup generated by $w_0, w_1, \ldots$ will be written as $\langle w_0, w_1, \ldots \rangle$.

We observe that

$$\forall k > 0\, \forall w > 0\, \exists l > k.\ \left(\langle w_0, \ldots, w_k \rangle \subseteq A_l\mathbb{N}^{d_l} \not\ni w_l \text{ and } w_l > w\right). \tag{8}$$

Consider a quotient set Q obtained from $\langle w_0, \ldots, w_k \rangle \subset \mathbb{N}$ by the congruence modulo w_0. Then the cardinality of the quotient set is

$$c(k) := \#\left\{i \in [0, w_0) \cap \mathbb{N}\ ;\ (\mathbb{N}w_0 + i) \cap \langle w_0, \ldots, w_k \rangle \neq \emptyset\right\}.$$

Let $\{u_1, \ldots, u_{c(k)}\} = \{\min q\ ;\ q \in Q\}$. By (8) with k being a and w being $\max_{1 \leq j \leq c(k)} u_j$, we have for any j, $u_j < w_l \notin \langle w_0, \ldots, w_k \rangle = \bigcup_{j=1}^{c(k)} (\mathbb{N}w_0 + u_j)$. Because of the minimality of u_j, classes $\mathbb{N}w_0 + w_l$, $\mathbb{N}w_0 + u_j$ $(1 \leq j \leq c(k))$ are mutually disjoint. Thus $c(k) < c(l) \leq w_0$. By repeating this argument, $c(m) = w_0$ for some m. Because $w_0 \in \langle w_0, \ldots, w_m \rangle$, a set $\mathbb{N} \setminus \langle w_0, \ldots, w_m \rangle$ is finite. But it should contain an infinite set $\{w_i\ ;\ i > m\}$. Contradiction.

(Only if-part) When $e = 2$, then the class has an infinite elasticity, because of the proof of Theorem 3. The infinite elasticity for $e = 2$ induces that for $e > 2$, by considering matrices with the 3rd to e-th rows being all zero.

Actually, a semigroup generated from positive integers a and b has a following structure:

Lemma 1. *For all $a, b \in \mathbb{N}$ such that $\gcd(a, b) = d \geq 1$, if $n \geq n_0 := ab/d - (a+b) + d$ (actually, $n_0 \geq ab/d - (a+b) + 1$) and d divides n then $n \in \mathbb{N}a + \mathbb{N}b$.*

5 Integer Crystalline Structure: Learning Algorithm as Geometric Algorithm

The class $\mathcal{X}_k^d$ of integer crystalline structures, introduced in (2), has many examples.

- The arrangement of atoms in a body centered crystal, a face centered crystal, and a diamond crystal are $\begin{pmatrix} \frac{-1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{-1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{-1}{2} \end{pmatrix} \mathbb{Z}^3$, $K = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} \mathbb{Z}^3$, $K + \{0,\ {}^t(1/4, 1/4, 1/4)\}$.
- The vertex sets of periodic tilings(e.g. Archimedean tilings [11]). For example, the vertex set of Archimedean (3.6.3.6) tiling(i.e. the Kagome lattice) is $H(2\mathbb{Z}^2 + F)$ where $H = \begin{pmatrix} 1 & 1/2 \\ 0 & \sqrt{3}/2 \end{pmatrix}$ and $F = \{\, {}^t(0,0),\ {}^t(0,1),\ {}^t(1,0)\}$, while the vertex set of Archimedean $(3^4.6)$ tiling is the matrix H multiplied by the Minkowski sum of $\begin{pmatrix} 1 & 3 \\ 2 & -1 \end{pmatrix} \mathbb{Z}^2$ and F, where F consists of ${}^t(1,0)$, ${}^t(1,1)$, ${}^t(2,0)$, ${}^t(2,1)$, ${}^t(3,0)$, and ${}^t(3,1)$.

Theorem 7. *$\mathcal{X}_k^d$ has finite elasticity.*

Proof. $\mathcal{X}_1^d$ has a finite elasticity. Otherwise $\exists P_1, P_2, \ldots \in \mathcal{X}_1^d\ \exists \boldsymbol{w}_0, \boldsymbol{w}_1, \ldots\ \forall n \geq 1. \{\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{n-1}\} \subseteq P_n \not\ni \boldsymbol{w}_n$. So for each $n \geq 1$ $\{\boldsymbol{0}, \boldsymbol{w}_1 - \boldsymbol{w}_0, \ldots, \boldsymbol{w}_{n-1} - \boldsymbol{w}_0\} \in P_n - \boldsymbol{w_0} \not\ni \boldsymbol{w}_{n-1} - \boldsymbol{w}_0$. Because any translated lattice containing the origin is actually a lattice, we have $P_n - \boldsymbol{w}_0$ is a lattice. Thus we have an infinite strictly ascending chain of $\mathbb{Z}$-submodules of a $\mathbb{Z}$-modules $\mathbb{Z}^d$, which contradicts that $\mathbb{Z}^d$ is a Noetherian $\mathbb{Z}$-module (It is well-known that for any Noetherian ring A, M is a Noetherian A-modules if and only if M is finitely generated.) By the finite elasticity of $\mathcal{X}_1^d$, the elementwise union of k copies of $\mathcal{X}_1^d$ has a finite elasticity, from which a subclass $\mathcal{X}_k^d$ has too.

Theorem 8. *The union $\mathcal{X}_\infty^d$ of all $\mathcal{X}_k^d$ over all $k > 0$ is not learnable from positive data; hence has an infinite elasticity.*

Proof. An infinite subset of $\mathbb{Z}^d$ belongs to $\mathcal{X}_\infty^d$, and any finite subset F of $\mathbb{Z}^d$ does so too because $F = O\mathbb{Z}^d + F \in \mathcal{X}_\infty^d$ with O being the zero matrix. That is, $\mathcal{X}_\infty^d$ is superfinite. By [12], $\mathcal{X}_\infty^d$ is not learnable from positive data. Thus $\mathcal{X}_\infty^d$ cannot have finite elasticity.

A fast learning algorithm of $\mathcal{X}_1^d$ can be implemented by using a fast algorithm to compute Hermite normal form.

Algorithm 1

- input: $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots,$
- procedure: From $\boldsymbol{x}_2 - \boldsymbol{x}_1$, $\boldsymbol{x}_3 - \boldsymbol{x}_1, \cdots$, compute the $\mathbb{Z}$-independent basis $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_d$ that spans $\{\boldsymbol{x}_i - \boldsymbol{x}_1\ ;\ i \geq 2\}$, by using Hermite normal form. Set A by $[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_d]$, and set $\boldsymbol{f}$ by $\boldsymbol{x}_2 - \boldsymbol{x}_1$.

One of a learning algorithm of $\mathcal{X}_k^1$ is the following:

Algorithm 2. The input is the data $x_1, \ldots, x_n \in \mathbb{Z}$ $(n > 1)$ presented so far. They are all in some $D \in \mathcal{X}_k^1$. The output is a pair of $a \in \mathbb{Z}$ and a finite $F \subset \mathbb{Z}$ such that $a\mathbb{Z} + F \in \mathcal{X}_k^1$ is minimal among $\left\{C \in \mathcal{X}_k^1 \ ; \ C \supseteq \{x_1, \ldots, x_n\}\right\}$. As the hypothesis of a, the algorithm computes the maximum range of $k+1$ *adjacent points* bounded by the minimum range of $k+2$ adjacent points.

An idea to generalize the algorithm for $\mathcal{X}_k^d$ is consider the convex hull of $k+1$ points, but it is a difficult problem in general to compute convex hull of points in high dimension. We hope some geometric and combinatorial argument inherent to $\mathcal{X}_k^d$ helps.

6 Noetherian Spaces

We prove that if the dimension of a Noetherian space $\mathcal{X}$ is d, then any ascending chain among unions of two irreducible closed sets has length at most d^2 times diagonal Ramsey number of $d+1$.

We review a basic notions of Noetherian spaces according to [13]. A topological space $\mathcal{X}$ is said to be irreducible if $\mathcal{X} \neq \emptyset$ and one of the following equivalent conditions (1)–(4) holds, so that all of them hold: (1) $\mathcal{X} = F_1 \cup F_2$ for closed sets F_1 and F_2 in $\mathcal{X}$ implies $\mathcal{X} = F_1$ or $\mathcal{X} = F_2$. (2) Every open set $O \neq \emptyset$ in $\mathcal{X}$ is dense in $\mathcal{X}$. (3) The intersection of a finite number of nonempty open subsets in $\mathcal{X}$ is not empty. (4) Every open set in $\mathcal{X}$ is connected. A subset E of a topological space X is *irreducible* if it is irreducible with the relative topology. We say a topological space $\mathcal{X}$ is a *Noetherian space* if for every descending chain $F_1 \supseteq F_2 \supseteq \cdots$ of closed subsets in $\mathcal{X}$, there exists n such that $F_n = F_{n+1} = \cdots$, which is equivalent to the condition that every family of closed sets in $\mathcal{X}$ has a minimal element with respect to inclusion. For example, for an algebraically closed field Ω containing a field of characteristic 0, the n-fold direct product Ω^n with Zariski topology is a Noetherian space. For a Noetherian topological space $\mathcal{X}$, we define the *dimension* $\dim \mathcal{X}$ to be the supremum of all integers n such that there exists a sequence $Z_0 \subsetneq Z_1 \subsetneq \cdots \subsetneq Z_n$ of irreducible closed sets in $\mathcal{X}$.

By the union of two topological spaces $\mathcal{X} = (X, \mathcal{O}(\mathcal{X}))$ and $\mathcal{Y} = (Y, \mathcal{O}(\mathcal{Y}))$, we mean a topological space $\mathcal{Z} = (X \cup Y, \mathcal{O}(\mathcal{Z}))$ such that $\mathcal{O}(\mathcal{X}) \cup \mathcal{O}(\mathcal{Y})$ is a subbase of $\mathcal{O}(\mathcal{Z})$.

Let $\mathcal{X}_1$ and $\mathcal{X}_2$ be Noetherian spaces of dimensions $\dim \mathcal{X}_1, \dim \mathcal{X}_2 < \infty$. Then we have following facts. If $\mathcal{Z}$ is the disjoint union of $\mathcal{X}_1$ and $\mathcal{X}_2$, then $\mathcal{Z}$ is a Noetherian space of dimension $\max(\dim \mathcal{X}_1, \ \dim \mathcal{X}_2)$. If $\mathcal{Z}$ is the direct product of $\mathcal{X}_1$ and $\mathcal{X}_2$, then $\mathcal{Z}$ is a Noetherian space of dimension $\dim \mathcal{X}_1 + \dim \mathcal{X}_2$. If $\mathcal{Z}$ is the intersection of $\mathcal{X}_1$ and $\mathcal{X}_2$, then $\mathcal{Z}$ is a Noetherian space of dimension $\min(\dim \mathcal{X}_1, \ \dim \mathcal{X}_2)$.

Proposition 7 (Irreducible component decomposition [13]). *Let $\mathcal{X}$ be a Noetherian space. For any closed set F in $\mathcal{X}$ there exist a positive integer r and irreducible closed sets $Z_1, \ldots, Z_r$ such that $F = \bigcup_{i=1}^{r} Z_i$. The condition $Z_i \not\subseteq$*

$Z_j (i \neq j)$ uniquely determines $Z_1, \ldots, Z_r$, which are the totality of irreducible components of F.

The situation of Theorem 2 also appears when we consider the length of ascending chain of closed sets with 2 irreducible components.

Theorem 9. *Let $\mathcal{X}$ be a Noetherian space of dimension d. If $Z_1 \cup Z'_1 \supsetneq Z_2 \cup Z'_2 \supsetneq \cdots \supsetneq Z_n \cup Z'_n$ and all of Z_i's and Z'_i's are irreducible closed sets in $\mathcal{X}$, then n is less than or equal to $d^2 R(d+1,\ d+1,\ 2)$.*

The proof of previous Theorem also established a following version of Proposition 1.

Theorem 10. *If neither $\mathcal{X}$ nor $\mathcal{Y}$ admits an infinite ascending chain of irreducible closed subsets, then the union does not either.*

References

1. Ito, M.: Algebraic theory of automata and languages. World Scientific Publishing Co. Inc., River Edge (2004)
2. Gilmer, R.: Commutative semigroup rings. Chicago Lectures in Mathematics. University of Chicago Press, Chicago (1984)
3. Howie, J.M.: An introduction to semigroup theory. L.M.S. Monographs, vol. 7. Academic Press [Harcourt Brace Jovanovich Publishers], London (1976)
4. Motoki, T., Shinohara, T., Wright, K.: The correct definition of finite elasticity: corrigendum to identification of unions. In: COLT 1991: Proceedings of the fourth annual workshop on Computational learning theory, p. 375. Morgan Kaufmann Publishers Inc., San Francisco (1991)
5. Stephan, F., Ventsov, Y.: Learning algebraic structures from text. Theoret. Comput. Sci. 268(2), 221–273 (2001); Algorithmic learning theory (Otzenhausen 1998)
6. Helmbold, D., Sloan, R., Warmuth, M.K.: Learning integer lattices. SIAM J. Comput. 21(2), 240–266 (1992)
7. Graham, R.L., Rothschild, B.L., Spencer, J.H.: Ramsey theory, 2nd edn. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons, Inc., New York (1980); A Wiley-Interscience Publication
8. 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)
9. Reidenbach, D.: A non-learnable class of e-pattern languages. Theor. Comput. Sci. 350(1), 91–102 (2006)
10. Angluin, D.: Inductive inference of formal languages from positive data. Inform. Control 45(2), 117–135 (1980)
11. Grünbaum, B., Shephard, G.C.: Tilings and patterns. A Series of Books in the Mathematical Sciences. W. H. Freeman and Company, New York (1989)
12. Gold, E.M.: Language identification in the limit. Inform. Control 10(5), 447–474 (1967)
13. Kimura, T.: Introduction to prehomogeneous vector spaces. Translations of Mathematical Monographs, vol. 215. American Mathematical Society, Providence (2003); Translated from the Japanese original by Makoto Nagura and Tsuyoshi Niitani and revised by the author (2003)

Matching Trace Patterns with Regular Policies

Franz Baader[1,⋆], Andreas Bauer[2], and Alwen Tiu[2]

[1] TU Dresden, Germany
baader@inf.tu-dresden.de
[2] The Australian National University
{baueran,alwen.tiu}@rsise.anu.edu.au

Abstract. We consider policies that are described by regular expressions, finite automata, or formulae of linear temporal logic (LTL). Such policies are assumed to describe situations that are problematic, and thus should be avoided. Given a trace pattern u, i.e., a sequence of action symbols and variables, were the variables stand for unknown (i.e., not observed) sequences of actions, we ask whether u potentially violates a given policy L, i.e., whether the variables in u can be replaced by sequences of actions such that the resulting trace belongs to L. We also consider the dual case where the regular policy L is supposed to describe all the admissible situations. Here, we want to know whether u always adheres to the given policy L, i.e., whether all instances of u belong to L. We determine the complexity of the violation and the adherence problem, depending on whether trace patterns are linear or not, and on whether the policy is assumed to be fixed or not.

1 Introduction

Regular languages (defined by regular expressions, finite automata, or temporal logics such as LTL) are frequently used in Computer Science to specify the wanted or unwanted behaviour of a system [1,8,3,4]. Such specifications are not only employed in the design phase of the system, where one may try to verify that every execution trace of the system respects the specification [5]. They can also be used in the deployment phase to monitor the actual system trace and raise an alarm if the specification is violated by the behaviour of the system [7,2]. In many of these applications, one actually employs ω-regular languages to describe the infinite behaviour of a reactive system, but in our intended application domain, considering finite sequences of actions (called traces in the following) appears to be more appropriate.

In online trading systems, like eBay, formal policies can be used to describe sequences of actions that indicate potentially malicious, dishonest, or fraudulent behaviour (see, e.g., [14]). Of course, it is not always easy to define potentially problematic behaviour without creating too many false positives. But even if such a definition is available, detecting that the actual trace indeed violates the given policy is non-trivial due to the fact that the administrator of an online

⋆ Supported by NICTA, Canberra Research Lab.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 105–116, 2009.

trading platform may not be able to observe all the relevant user actions. For example, payments made through a third-party institution, shipments of goods, etc., cannot be directly observed. Our approach for modelling this situation is that we use regular languages to describe policies, but instead of traces we consider trace patterns, i.e., traces with variables, where the variables stand for unknown sequences of actions. More formally, assume that L is a policy (formally defined as a regular language) describing undesirable sequences of actions.[1] We say that a given trace w violates the policy L if $w \in L$. Checking for a violation is thus just an instance of the word problem for regular languages. If, instead of a trace, we only have a trace pattern, then detecting violations of the policy becomes more complicated. For example, consider the trace pattern $abXaY$, where a, b are action symbols and X, Y are variables. This trace pattern says: all we know about the actual trace is that it starts with ab, is followed by some trace u, which is followed by a, which is in turn followed by some trace v. Given such a trace pattern, all traces that can be obtained from it by replacing its variables with traces (i.e., finite sequences of actions) are possibly the actual trace. The policy L is potentially violated if one of the traces obtained by such a substitution of the variables by traces belongs to L. In our example, $abXaY$ potentially violates $L = (ab)^*$ since replacing X by ab and Y by b yields the trace $ababab \in L$. The trace pattern $abXaY$ is linear since every variable occurs at most once in it. We can also consider non-linear trace patterns such as $abXaX$, where different occurrences of the same variable must be replaced by the same trace. The underlying idea is that, though we do not know which actions took place in the unobserved part of the trace, we know (from some source) that the same sequence of actions took place. It is easy to see that the policy $L = (ab)^*$ is not potentially violated by $abXaX$.

In this paper, we will show that the complexity of deciding whether a given trace pattern potentially violates a regular policy depends on whether the trace pattern is linear or not. For linear trace patterns, the problem is decidable in polynomial time whereas for non-linear trace patterns the problem is PSpace-complete. If we assume that the policy is fixed, i.e., its size (more precisely, the size of a finite automaton or regular expression representing it) is constant, then the problem can be solved in linear time for linear trace patterns and is NP-complete for non-linear trace patterns. We also consider the dual case where the regular policy L is supposed to describe all the admissible situations. Here, we want to know whether u always adheres to the given policy L, i.e., whether all instances of u belong to L. For the case of a fixed policy, the adherence problem is coNP-complete for arbitrary trace patterns and linear for linear trace patterns. If the policy is not assumed to be fixed, however, then the adherence problem is PSpace-complete both for linear and non-linear trace patterns. Finally, we consider the case where the policy is given by an LTL formula. If the policy is not assumed to be fixed, then the violation and the adherence problem are PSpace-complete both for linear and non-linear trace patterns. For the case of a

[1] Using regular languages of *finite* words to express policies means that we only monitor safety properties [11]. For more general notions of policies, see [20].

fixed policy, the violation (adherence) problem is NP-complete (coNP-complete) for non-linear patterns and it can be solved in linear time for linear patterns.

2 Preliminaries

In the following, we consider finite alphabets Σ, whose elements are called *action symbols*. A *trace* is a (finite) word over Σ, i.e., an element of Σ^*. A *trace pattern* is an element of $(\Sigma \cup \mathcal{V})^*$, i.e., a finite word over the extended alphabet $\Sigma \cup \mathcal{V}$, where $\mathcal{V}$ is a finite set of *trace variables*. The trace pattern u is called *linear* if every trace variable occurs at most once in u. A *substitution* is a mapping $\sigma : \mathcal{V} \to \Sigma^*$. This mapping is extended to a mapping $\widehat{\sigma} : (\Sigma \cup \mathcal{V})^* \to \Sigma^*$ in the obvious way, by defining $\widehat{\sigma}(\varepsilon) = \varepsilon$ for the empty word ε, $\widehat{\sigma}(a) = a$ for every action symbol $a \in \Sigma$, $\widehat{\sigma}(X) = \sigma(X)$ for every trace variable $X \in \mathcal{V}$, and $\widehat{\sigma}(uv) = \widehat{\sigma}(u)\widehat{\sigma}(v)$ for every pair of non-empty trace patterns u, v. A *policy* is a regular language over Σ. We assume that such a policy is given either by a regular expression or by a (non-deterministic) finite automaton. For our complexity results, it is irrelevant which of these representations we actually use.

Definition 1. *Given a trace pattern u and a policy L, we say that u* potentially violates *L (written $u \lesssim L$) if there is a substitution σ such that $\widehat{\sigma}(u) \in L$. The* violation problem *is the following decision problem:*

Given: *A policy L and a trace pattern u.*
Question: *Does $u \lesssim L$ hold or not?*

If the trace pattern u in this decision problem is restricted to being linear, then we call this the linear violation problem*.*

We assume that the reader is familiar with regular expressions and finite automata. Given a (non-deterministic) finite automaton $\mathcal{A}$, states p, q in $\mathcal{A}$, and a word w, we write $p \to_{\mathcal{A}}^{w} q$ to say that there is a path in $\mathcal{A}$ from p to q with label w. The set of labels of all paths from p to q is denoted by $L_{p,q}$. The following problem turns out to be closely connected to the violation problem. The *intersection emptiness problem* for regular languages is the following decision problem:

Given: Regular languages $L_1, \ldots, L_n$.
Question: Does $L_1 \cap \ldots \cap L_n = \emptyset$ hold or not?

This problem is PSpace-complete [13,10], independent of whether the languages are given as regular expressions, non-deterministic finite automata, or deterministic finite automata.

3 The Linear Violation Problem

Assume that u is a linear trace pattern and L is a regular language. Let the trace pattern u be of the form $u = u_0 X_1 u_1 \ldots X_m u_m$ where $u_i \in \Sigma^*$ $(i = 0, \ldots, m)$ and $X_1, \ldots, X_m$ are distinct variables. Obviously, we have

$$u \lesssim L \text{ iff } u_0 \Sigma^* u_1 \ldots \Sigma^* u_m \cap L \neq \emptyset.$$

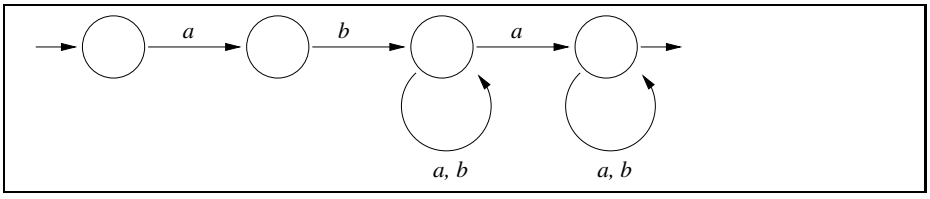


Fig. 1. A non-deterministic finite automaton accepting $ab\Sigma^*a\Sigma^*$

If n is the length of $u_0u_1 \dots u_m$, then we can build a non-deterministic finite automaton $\mathcal{A}$ accepting the language $u_0\Sigma^*u_1 \dots \Sigma^*u_m$ that has $n+1$ states. For example, given the linear trace pattern $abXaY$ from the introduction, we consider the language $ab\Sigma^*a\Sigma^*$, where $\Sigma = \{a, b\}$. Fig. 1 shows a non-deterministic finite automaton with 4 states accepting this language. In addition, there is a non-deterministic finite automaton $\mathcal{B}$ accepting L such that the number of states ℓ of $\mathcal{B}$ is polynomial in the size of the original representation for L.[2] By constructing the product automaton of $\mathcal{A}$ and $\mathcal{B}$, we obtain a non-deterministic finite automaton accepting $u_0\Sigma^*u_1 \dots \Sigma^*u_m \cap L$ with $(n+1) \cdot \ell$ states. Thus, emptiness of this language can be tested in time linear in $(n+1) \cdot \ell$, and thus in time polynomial in the size of the input u, L of our linear violation problem. If the policy is assumed to be fixed, then ℓ is a constant, and thus emptiness of the automaton accepting $u_0\Sigma^*u_1 \dots \Sigma^*u_m \cap L$ can be tested in time linear in the length of u.

Theorem 1. *The linear violation problem can be solved in polynomial time. If the policy is assumed to be fixed, it can even be solved in time linear in the length of the input trace pattern.*

4 The General Violation Problem

Allowing also the use of non-linear patterns increases the complexity.

Theorem 2. *The violation problem is PSpace-complete.*

Proof. PSpace-hardness can be shown by a reduction of the intersection emptiness problem for regular languages. Given regular languages $L_1, \dots, L_n$, we construct the trace pattern $u_n := \#X\#X \dots \#X\#$ of length $2n+1$ and the policy $L(L_1, \dots, L_n) := \#L_1\#L_2 \dots \#L_n\#$. Here X is a variable and $\#$ is a new action symbol not occurring in any of the words belonging to one of the languages $L_1, \dots, L_n$. Obviously, both u_n and (a representation of) $L(L_1, \dots, L_n)$ can be constructed in time polynomial in the size of (the representation of) $L_1, \dots, L_n$. To be more precise regarding the representation issue, if we want to show PSpace-hardness for the case where the policy is given by a regular expression (a non-deterministic finite automaton, a deterministic finite automaton), then we assume

[2] In fact, it is well-known that, given a regular expression r for L, one can construct a non-deterministic finite automaton accepting L in time polynomial in the size of r.

that the regular languages $L_1, \ldots, L_n$ are given by the same kind of representation. It is easy to see that the following equivalence holds:

$$L_1 \cap \ldots \cap L_n \neq \emptyset \text{ iff } u_n \lesssim L(L_1, \ldots, L_n).$$

Thus, we have shown that the intersection emptiness problem for regular languages can be reduced in polynomial time to the violation problem. Since the intersection emptiness problem is PSpace-complete [13] (independent of whether the regular languages are given as regular expressions, non-deterministic finite automata, or deterministic finite automata), this shows that the violation problem is PSpace-hard (again independent of the chosen representation).

To show *membership* of the violation problem *in PSpace*, consider the violation problem for the trace pattern u and the policy L. Let n be the length of u and $\mathcal{A}$ a non-deterministic finite automaton accepting L. For $i \in \{1, \ldots, n\}$, we denote the symbol in $\Sigma \cup \mathcal{V}$ occurring at position i in u with u_i, and for every variable X occurring in u, we denote the set of positions in u at which X occurs with P_X, i.e., $P_X = \{i \mid 1 \leq i \leq n \wedge u_i = X\}$.

It is easy to see that $u \lesssim L$ iff there is a sequence $q_0, \ldots, q_n$ of states of $\mathcal{A}$ such that the following conditions are satisfied:

1. q_0 is an initial state and q_n is a final state;
2. for every $i \in \{1, \ldots, n\}$, if $u_i \in \Sigma$, then $q_{i-1} \rightarrow_{\mathcal{A}}^{u_i} q_i$;
3. for every variable X occurring in u, we have $\bigcap_{i \in P_X} L_{q_{i-1}, q_i} \neq \emptyset$.[3]

Based on this characterisation of "$u \lesssim L$" we can obtain a PSpace decision procedure for the violation problem as follows. This procedure is non-deterministic, which is not a problem since NPSpace = PSpace by Savitch's theorem [19]. It guesses a sequence $q_0, \ldots, q_n$ of states of $\mathcal{A}$, and then checks whether this sequence satisfies the Conditions 1–3 from above. Obviously, the first two conditions can be checked in polynomial time, and the third condition can be checked within PSpace since the intersection emptiness problem for regular languages is PSpace-complete [13]. □

Alternatively, we could have shown membership in PSpace by reducing it to the known PSpace-complete problem of *solvability of word equations with regular constraints* [21,17]. Due to limited space and the fact that the algorithm for testing solvability of word equations with regular constraints described in [17] is rather complicated and "impractical," we do not describe this reduction here.

Let us now consider the complexity of the violation problem for the case where the policy is assumed to be fixed. In this case, the NPSpace algorithm described in the proof of Theorem 2 actually becomes an NP algorithm. In fact, guessing the sequence of states $q_0, \ldots, q_n$ can be realized using polynomially many binary choices (i.e., with an NP algorithm), testing Conditions 1 and 2 is clearly polynomial, and testing Condition 3 becomes polynomial since the size of $\mathcal{A}$, and thus of non-deterministic finite automata accepting the languages L_{q_{i-1}, q_i}, is constant.

[3] Recall that $L_{p,q}$ denotes the set of words labeling paths in $\mathcal{A}$ from p to q.

Theorem 3. *If the policy is assumed to be fixed, then the violation problem is in NP.*

The matching NP-hardness result of course depends on the fixed policy. For example, if $L = \Sigma^*$, then we have $u \lesssim L$ for all trace patterns u, and thus the violation problem for this fixed policy can be solved in constant time. However, we can show that there are policies for which the problem is NP-hard. Given a fixed policy L, the *violation problem for* L is the following decision problem

Given: A trace pattern u.
Question: Does $u \lesssim L$ hold or not?

Theorem 4. *There exists a fixed policy such that the violation problem for this policy is NP-hard.*

Proof. To show *NP-hardness*, we use a reduction from the well-known NP-complete problem 3SAT [10]. Let $C = c_1 \wedge \ldots \wedge c_m$ be an instance of 3SAT, and $\mathcal{P} = \{p_1, \ldots, p_n\}$ the set of propositional variables occurring in C. Every 3-clause c_i in C is of the form $c_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$, where the $l_{i,j}$ are literals, i.e., propositional variables or negated propositional variables. In the corresponding violation problem, we use the elements of $\mathcal{V} := \{P_i \mid p_i \in \mathcal{P}\}$ as trace variables, and as alphabet we take $\Sigma := \{\#, \neg, \vee, \wedge, \top, \bot\}$. The positive literal p_i is encoded as the trace pattern $\#P_i\#$ and the negative literal $\neg p_i$ as $\neg\#P_i\#$. For a given literal l, we denote its encoding as a trace pattern by $\iota(l)$. 3-Clauses are encoded as "disjunctions" of the encodings of their literals, i.e., $c_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$ is encoded as $\iota(c_i) = \iota(l_{i,1}) \vee \iota(l_{i,2}) \vee \iota(l_{i,3})$, and 3SAT-problems are encoded as "conjunctions" of their 3-clauses, i.e., if $C = c_1 \wedge \ldots \wedge c_m$, then $\iota(C) = \iota(c_1) \wedge \ldots \wedge \iota(c_m)$.

Our fixed policy describes all situations that can make a 3-clause true. To be more precise, consider $\iota(c) = \iota(l_1) \vee \iota(l_2) \vee \iota(l_3)$ for a 3-clause $c = l_1 \vee l_2 \vee l_3$. If we replace the trace variables in c by either $\top$ or $\bot$, then we get a trace of the form $w_1 \vee w_2 \vee w_3$ where each w_i belongs to the set

$$K := \{\#\top\#,\ \#\bot\#,\ \neg\#\top\#,\ \neg\#\bot\#\}.$$

Intuitively, replacing the trace variable P_i by $\top$ ($\bot$) corresponds to replacing the propositional variable p_i by true (false). Thus, a substitution σ that replaces trace variables by $\top$ or $\bot$ corresponds to a propositional valuation v_σ. The valuation v_σ makes the 3-clause c true iff $\widehat{\sigma}(\iota(c)) = w_1 \vee w_2 \vee w_3$ is such that there is an $i, 1 \leq i \leq 3$, with $w_i \in \{\#\top\#, \neg\#\bot\#\}$. For this reason, we define

$$T := \{w_1 \vee w_2 \vee w_3 \mid \{w_1, w_2, w_3\} \subseteq K \text{ and there is an } i, 1 \leq i \leq 3, \text{ with } w_i \in \{\#\top\#, \neg\#\bot\#\}\}.$$

To make a conjunction of 3-clauses true, we must make every conjunct true. Consequently, we define our fixed policy L as $L_{3SAT} := (T\wedge)^*T$. Since T is a finite language, L_{3SAT} is obviously a regular language. NP-hardness of the violation problem for L_{3SAT} is an immediate consequence of the (easy to prove) fact that C is satisfiable iff $\iota(C) \lesssim L_{3SAT}$. □

5 The Adherence Problem

Instead of using regular languages to describe traces that are viewed as being problematic, one could assume that a regular policy L describes all the admissible situations. In this case, we want to know whether u always adheres L.

Definition 2. *Given a trace pattern u and a policy L, we say that u* always adheres to *L (written $u \models L$) if $\widehat{\sigma}(u) \in L$ holds for all substitutions σ. The* adherence problem *is the following decision problem:*
Given: *A policy L and a trace pattern u.*
Question: *Does $u \models L$ hold or not?*
If the trace pattern u in this decision problem is restricted to being linear, then we call this the linear adherence problem*.*

Obviously, we have $u \models L$ iff not $u \lesssim \Sigma^* \setminus L$, which shows that the adherence problem and the complement of the violation problem can be reduced to each other. If the policy L is assumed to be fixed, then these reductions are linear in the length of u. Thus, we obtain the following corollary to our Theorems 1, 3, and 4.

Corollary 1. *Assume that the policy is fixed. Then, the linear adherence problem can be solved in time linear in the length of the input trace pattern. In addition, the general adherence problem is in coNP, and there exists a policy such that the general adherence problem for this policy is coNP-hard.*

Another case for which the above reductions are linear is if the policy is given by a *deterministic* finite automaton. Thus, our Theorems 1 and 2 yield the following corollary.

Corollary 2. *Assume that the policy is given by a deterministic finite automaton. Then, the linear adherence problem can be solved in polynomial time, and the general adherence problem is PSpace-complete.*

If the policy is neither fixed nor given by a deterministic finite automaton, then the reductions between the adherence problem and the complement of the violation problem are not polynomial since they involve the (potentially exponential) construction of the complement automaton for a non-deterministic finite automaton. In fact, in this case there cannot be a polynomial time reduction between the two problems since the adherence problem is intractable even for linear trace patterns, for which the violation problem is tractable.

Lemma 1. *The linear adherence problem is PSpace-hard if the policy is given by a non-deterministic finite automaton or a regular expression.*

Proof. Consider the linear trace pattern X and an arbitrary regular language L over the alphabet Σ. Obviously, we have $X \models L$ iff $L = \Sigma^*$. The problem of deciding whether a regular language (given by a regular expression or a non-deterministic finite automaton) is the universal language Σ^* or not is PSpace-complete [10]. Consequently, the adherence problem is PSpace-hard even for linear trace patterns. □

Obviously, this PSpace lower bound then also holds for the general adherence problem. Next, we show that a matching PSpace upper bound holds for the general adherence problem, and thus for the linear one as well.

Lemma 2. *The adherence problem is in PSpace if the policy is given by a non-deterministic finite automaton or a regular expression.*

Proof. Since PSpace is a deterministic complexity class, it is sufficient to show that the complement of the adherence problem is in PSpace. Thus, given the trace pattern u of length n and the policy L, we want to decide whether $u \models L$ does not hold. As noted above, this is the same as deciding whether $u \lesssim \Sigma^* \setminus L$. Basically, we will use the PSpace decision procedure for the violation problem described in the proof of Theorem 2 to decide this problem. However, we cannot explicitly construct the automaton for $\Sigma^* \setminus L$ from the one for L since the size of this automaton might be exponential in the size of the automaton (or regular expression) for L. Instead, we construct the relevant parts of this automaton on-the-fly.

Let $\mathcal{A}$ be a non-deterministic finite automaton accepting L that has k states, and $\mathcal{B}$ the deterministic automaton for $\Sigma^* \setminus L$ constructed in the usual way from $\mathcal{A}$, i.e., the states of $\mathcal{B}$ are all the subsets of the set of states of $\mathcal{A}$, the initial state of $\mathcal{B}$ is the set of initial states of $\mathcal{A}$, the final states of $\mathcal{B}$ are the sets not containing any final state of $\mathcal{A}$, and $P \to^a_{\mathcal{B}} Q$ iff $Q = \{q \mid \exists p \in P\colon p \to^a_{\mathcal{A}} q\}$. Although the size of $\mathcal{B}$ is exponential in the size of $\mathcal{A}$, every single state of $\mathcal{B}$ can be represented using linear space. Also, deciding whether a given state of $\mathcal{B}$ is the initial state or a final state requires only polynomial space, and the same is true for constructing, for a given state P of $\mathcal{B}$ and $a \in \Sigma$, the unique state Q such that $P \to^a_{\mathcal{B}} Q$.

For $i \in \{1, \ldots, n\}$, we again denote the symbol in $\Sigma \cup \mathcal{V}$ occurring at position i in u with u_i, and for every variable X occurring in u, we denote the set of positions in u at which X occurs with P_X, i.e., $P_X = \{i \mid 1 \leq i \leq n \wedge u_i = X\}$. Then $u \lesssim \Sigma^* \setminus L$ iff there is a sequence $Q_0, \ldots, Q_n$ of states of $\mathcal{B}$ such that the following conditions are satisfied:

1. Q_0 is the initial state and Q_n is a final state of $\mathcal{B}$;
2. for every $i \in \{1, \ldots, n\}$, if $u_i \in \Sigma$, then $Q_{i-1} \to^{u_i}_{\mathcal{B}} Q_i$;
3. for every variable X occurring in u, we have $\bigcap_{i \in P_X} L_{Q_{i-1}, Q_i} \neq \emptyset$.

Obviously, this characterisation yields the desired PSpace decision procedure for $u \lesssim \Sigma^* \setminus L$ if we can show that, for each variable X, the non-emptiness of $\bigcap_{i \in P_X} L_{Q_{i-1}, Q_i}$ can be decided by an NPSpace procedure.

Let $P_X = \{i_1, \ldots, i_m\}$, and $I_j := Q_{i_j - 1}, F_j := Q_{i_j}$ for $j = 1, \ldots, m$. Note that $m \leq n$ where n is the length of the pattern u. To check $\bigcap_{1 \leq j \leq m} L_{I_j, F_j} \neq \emptyset$, we proceed as follows.

1. Start with the m-tuple $(T_1, \ldots, T_m)$ where $T_j := I_j$ for $j = 1, \ldots, m$.
2. Check whether $(T_1, \ldots, T_m) = (F_1, \ldots, F_m)$. If this is the case, then terminate successfully, i.e., with the result that the intersection $\bigcap_{1 \leq j \leq m} L_{I_j, F_j}$ is non-empty. Otherwise, continue with 3.

3. Guess a letter $a \in \Sigma$, and replace $(T_1, \ldots, T_m)$ by the corresponding tuple of successor states in $\mathcal{B}$, i.e., make the assignment $T_j := \{q \mid \exists p \in T_j \colon p \to_{\mathcal{A}}^{a} q\}$. Continue with 2.

Obviously, if this procedure terminates successfully, then $\bigcap_{1 \le j \le m} L_{I_j, F_j}$ is indeed non-empty. In addition, if the intersection is non-empty, then there is a way of guessing letters such that the procedure terminates successfully. However, as described until now, the procedure does not terminate if the intersection is empty. It is, however, easy to see that it is enough to guess a sequence of letters of length at most $2^{k \cdot m}$, which is the number of different tuples $(T_1, \ldots, T_m)$ to be encountered during a run of the procedure.[4] In fact, if a tuple is reached more than once in a run of our procedure, then we can cut out this cycle to get a shorter run that achieves the same. Consequently, one can stop each run after $2^{k \cdot m}$ iterations. This can be realized using a binary counter that requires $k \cdot m$ bits, i.e., space polynomial in the size of the input. □

The following theorem is an immediate consequence of the two lemmas that we have just shown.

Theorem 5. *Both the general and the linear adherence problem are PSpace-complete if the policy is given by a non-deterministic finite automaton or a regular expression.*

6 Policies Defined by LTL Formulae

In many applications, linear temporal logic (LTL) [18] is used to specify the (wanted or unwanted) behaviour of a system [15]. LTL is usually interpreted in temporal structures with infinitely many time points, but variants where temporal structures are finite sequences of time points have also been considered in the literature [6,9]. Here, we consider the setting employed in [6], where an LTL formula φ defines a regular language of finite words, which we denote by L_φ in the following. It is well-known that not all regular languages can be defined by LTL formulae: the class of languages definable by LTL formulae is precisely the class of star-free languages, which is a strict subclass of the class of all regular languages [6]. Given an LTL formula φ, one can construct a finite non-deterministic automaton $\mathcal{A}_\varphi$ that accepts L_φ (by adapting the Vardi-Wolper construction [23] to the finite case). Although the size of this automaton is exponential in the size of φ, it satisfies properties similar to the ones mentioned for the automaton $\mathcal{B}$ in the proof of Lemma 2: every single state of $\mathcal{A}_\varphi$ can be represented using linear space, deciding whether a given state of $\mathcal{A}_\varphi$ is an initial state or a final state requires only polynomial space, and the same is true for guessing, for a given state p of $\mathcal{A}_\varphi$ and $a \in \Sigma$, a state q such that $p \to_{\mathcal{A}_\varphi}^{a} q$. We will also use that, just as in the infinite case, the satisfiability problem for LTL over finite temporal structures (i.e., for a given LTL formula φ, decide whether L_φ is empty or not)

[4] Recall that k is the number of states of $\mathcal{A}$, and $m \le n$ where n is the length of the input pattern u.

is PSpace-complete (this can be shown by a proof identical to the one in [22] for the infinite case).

In this section, we consider the complexity of the violation (adherence) problem for the case where the policy L_φ is given by an LTL formula φ. First, note that the adherence and the violation problem can be reduced to each other in linear time since LTL allows for negation:

$$u \lesssim L_\varphi \text{ iff } u \not\models L_{\neg\varphi} \quad \text{and} \quad u \models L_\varphi \text{ iff } u \not\lesssim L_{\neg\varphi}.$$

Theorem 6. *Both the general and the linear violation (adherence) problem are PSpace-complete if the policy is given by an LTL formula.*

Proof. It is sufficient to show PSpace-completeness for the violation problem.

PSpace-hardness of the violation problem can be shown by a reduction of the satisfiability problem: the LTL formula φ is satisfiable iff $X \lesssim L_\varphi$.

Membership in PSpace can be shown just as in the proof of Lemma 2. We apply the PSpace decision procedure described in the proof of Theorem 2 to the automaton $\mathcal{A}_\varphi$, but without constructing this automaton explicitly. Instead, its relevant parts are constructed on the fly. Again, the main fact to be shown is that the induced intersection emptiness problems for the variable can be decided within PSpace. This can be done just as in the proof of Lemma 2. The only difference is that the automaton $\mathcal{A}_\varphi$ is non-deterministic whereas the automaton $\mathcal{B}$ considered in the proof of Lemma 2 was deterministic. However, this just means that, instead of constructing the unique tuple of successors states in Step 3, we guess such a successor tuple. □

Let us now consider the case where the policy (i.e., the LTL formula) is assumed to be fixed. This means that the size of the automaton $\mathcal{A}_\varphi$ is a constant. For the case of linear patterns, we can then decide the violation problem (and thus also the adherence problem) in linear time (see the proof of the second part of Theorem 1).

Theorem 7. *Assume that the policy is fixed and given by an LTL formula. Then, the linear violation problem and the linear adherence problem can be solved in time linear in the length of the input trace pattern.*

For non-linear trace patterns, the violation problem for fixed LTL policies has the same complexity as in the case of fixed regular policies. The results for the adherence problem then follow from the above reductions between the violation problem and the (complement of the) adherence problem. The proof of the following theorem is identical to the one of Theorem 3.

Theorem 8. *Assume that the policy is fixed and given by an LTL formula. Then the violation problem is in NP and the adherence problem is in coNP.*

In order to show NP-hardness of the violation problem for a fixed policy φ (which then implies coNP-hardness of the adherence problem for the fixed policy $\neg\varphi$), it is enough to show that the fixed policy $(T\wedge)^*T$ used in the proof of Theorem 4

is star-free [6]. Since the star-free languages are closed under complement, it is enough to show that the complement of this language is star-free. By definition, finite languages (and thus also T) are star-free. In addition, star-free languages are closed under all Boolean operations and concatenation. It is also well-known (and easy to see) that Σ^* as well as $(\Sigma \setminus \{a\})^*$ for any $a \in \Sigma$ are star-free [16]. The language $\Sigma^* \setminus (T\wedge)^*T$ is the union of the following star-free languages:

- all words not containing $\wedge$ and not belonging to T: $(\Sigma^* \setminus T) \cap (\Sigma \setminus \{\wedge\})^*$.
- all words not starting with an element of T before the first $\wedge$: $((\Sigma^* \setminus T) \cap (\Sigma \setminus \{\wedge\})^*)\wedge\Sigma^*$.
- all words not having a word of T between two consecutive occurrences of $\wedge$: $\Sigma^*\wedge((\Sigma^* \setminus T) \cap (\Sigma \setminus \{\wedge\})^*)\wedge\Sigma^*$.
- all words not ending with an element of T after the last $\wedge$: $\Sigma^*\wedge((\Sigma^* \setminus T) \cap (\Sigma \setminus \{\wedge\})^*)$.

This shows that $(T\wedge)^*T$ is star-free and thus can be expressed by an LTL formula φ. Thus, the proof of Theorem 4 yields the following results.

Theorem 9. *There exists a fixed policy given by an LTL formula such that the violation problem (adherence problem) for this policy is NP-hard (coNP-hard).*

7 Future Work

One of the main tasks to be addressed in our future work is to investigate which kinds of unwanted behaviour in online trading systems one can formally describe using regular languages and LTL formulae. We also intend to consider the problem of debugging policies as another application for our approach of matching trace patterns with regular policies. The violation and the adherence problem can be viewed as instances of the *policy querying* problem, which has been introduced in [12] as a tool for the analysis of access control policies. The main idea is that, while it may be quite hard for inexperienced users to correctly define a policy using a regular expression or an LTL formula, it should be easier to describe, using trace patterns, types of traces they want to allow or disallow. Checking for violation/adherence can then be used to find potential errors in the definition of the policy.

References

1. Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., Singerman, E., Tiemeyer, A., Vardi, M.Y., Zbar, Y.: The forSpec temporal logic: A new temporal property-specification language. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, p. 296. Springer, Heidelberg (2002)
2. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 260–272. Springer, Heidelberg (2006)

3. Bauer, A., Leucker, M., Streit, J.: SALT—structured assertion language for temporal logic. In: Liu, Z., He, J. (eds.) ICFEM 2006. LNCS, vol. 4260, pp. 757–775. Springer, Heidelberg (2006)
4. Ben-David, S., Fisman, D., Ruah, S.: Embedding finite automata within regular expressions. Theoretical Computer Science 404, 202–218 (2008)
5. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
6. Cohen, J., Perrin, D., Pin, J.-E.: On the expressive power of temporal logic. J. Comput. System Sci. 46, 271–294 (1993)
7. Colin, S., Mariani, L.: Run-time verification. In: Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.) Model-Based Testing of Reactive Systems. LNCS, vol. 3472, pp. 525–555. Springer, Heidelberg (2005)
8. Eisner, C., Fisman, D.: A Practical Introduction to PSL. Series on Integrated Circuits and Systems. Springer, Heidelberg (2006)
9. Eisner, C., Fisman, D., Havlicek, J., Lustig, Y., McIsaac, A., van Campenhout, D.: Reasoning with temporal logic on truncated paths. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 27–39. Springer, Heidelberg (2003)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability — A guide to NP-completeness. W. H. Freeman and Company, San Francisco (1979)
11. Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, p. 342. Springer, Heidelberg (2002)
12. Kirchner, C., Kirchner, H., Santana de Oliveira, A.: Analysis of rewrite-based access control policies. In: Proc. 3rd International Workshop on Security and Rewriting Techniques (2008)
13. Kozen, D.: Lower bounds for natural proof systems. In: Proc. FOCS 1977. IEEE Computer Society, Los Alamitos (1977)
14. Krukow, K., Nielsen, M., Sassone, V.: A framework for concrete reputation-systems with applications to history-based access control. In: Proc. ACM Conference on Computer and Communications Security. ACM, New York (2005)
15. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, Heidelberg (1992)
16. Perrin, D.: Finite Automata. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B. Elsevier, Amsterdam (1990)
17. Plandowski, W.: Satisfiability of word equations with constants is in PSPACE. In: Proc. FOCS 1999. IEEE Computer Society, Los Alamitos (1999)
18. Pnueli, A.: The temporal logic of programs. In: Proc. FOCS 1977. IEEE Computer Society, Los Alamitos (1977)
19. Savitch, W.J.: Relationship between nondeterministic and deterministic tape complexities. J. of Computer and System Sciences 4, 177–192 (1970)
20. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. 3(1), 30–50 (2000)
21. Schulz, K.U.: Makanin's algorithm for word equations - two improvements and a generalization. In: Schulz, K.U. (ed.) IWWERT 1990. LNCS, vol. 572. Springer, Heidelberg (1992)
22. Prasad Sistla, A., Clarke, E.C.: The complexity of propositional linear temporal logic. J. of the ACM 32(3), 733–749 (1985)
23. Vardi, M.Y., Wolper, P.: Automata-theoretic techniques for modal logics of programs. In: Proc. STOC 1984 (1984)

Absolute Convergence of Rational Series Is Semi-decidable

Raphaël Bailly and François Denis*

Laboratoire d'Informatique Fondamentale de Marseille
CNRS, Aix-Marseille Université
{raphael.bailly,francois.denis}@lif.univ-mrs.fr

Abstract. We study *real-valued absolutely convergent rational series*, i.e. functions $r : \Sigma^* \to \mathbb{R}$, defined over a free monoid Σ^*, that can be computed by a multiplicity automaton A and such that $\sum_{w\in\Sigma^*} |r(w)| < \infty$. We prove that any absolutely convergent rational series r can be computed by a multiplicity automaton A which has the property that $r_{|A|}$ is simply convergent, where $r_{|A|}$ is the series computed by the automaton $|A|$ derived from A by taking the absolute values of all its parameters. Then, we prove that the set $\mathcal{A}^{rat}(\Sigma)$ composed of all absolutely convergent rational series is semi-decidable and we show that the sum $\sum_{w\in\Sigma^*} |r(w)|$ can be estimated to any accuracy rate for any $r \in \mathcal{A}^{rat}(\Sigma)$. We also introduce a spectral radius-like parameter $\rho_{|r|}$ which satisfies the following property: r is absolutely convergent iff $\rho_{|r|} < 1$.

1 Introduction

Given a finite alphabet Σ, we consider real formal power series defined over the free monoid Σ^*, i.e. functions which map Σ^* into $\mathbb{R}$. More precisely, we consider *rational series*, which admit several characterizations, one of which being that they can be computed by multiplicity automata [1,2]. Given a rational series $r : \Sigma^* \to \mathbb{R}$, we study whether r is absolutely convergent, i.e. $\sum_{w\in\Sigma^*} |r(w)| < \infty$. It is polynomially decidable whether a rational series r is simply convergent, i.e. whether the sum $\sum_{n\geq 0} \sum_{w\in\Sigma^n} r(w)$ converges to a limit (see [3] for example). Since the Hadamard product r of two rational series s and t, defined by $r(w) = s(w)t(w)$, is rational, it is polynomially decidable whether a rational series r converges in quadratic norm, i.e. $\sum_{w\in\Sigma^*} r^2(w) < \infty$. However, to our knowledge, it is still unknown whether it can be decided if a rational series is absolutely convergent.

The motivation for the present work comes from a problem we have studied in grammatical inference. A *stochastic language* over Σ^* is a series p which takes only non negative values and s.t. $\sum_{w\in\Sigma^*} p(w) = 1$[1]. A classical problem in grammatical inference consists in inferring an estimate of a target stochastic language p from a finite sample of words $\{w_1, \ldots, w_n\}$ independently drawn according to p. In [4], we proposed an algorithm DEES which takes a sample

* This research was partially supported by the ANR project *Marmota*.

[1] This definition differs from the one given in [2].

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 117–128, 2009.

as input and outputs a rational series r which simply converges to 1 but can take negative values, and which satisfies the following property: with probability one, there exists a sample size level from which the sum $\sum_{w\in\Sigma^*} |p(w) - r(w)|$ is arbitrarily small, (which implies that r is absolutely convergent); moreover, a stochastic language p_r can be computed from r, that satisfies

$$\sum_{w\in\Sigma^*} |p_r(w) - r(w)| \leq \sum_{w\in\Sigma^*} |r|(w) - 1.$$

In other words, we know that from some sample size, we will have a solution of our problem. But we need to decide whether the series r output by DEES from the working sample is absolutely convergent to ensure that r provides a solution and we need to compute an estimate of $\sum_{w\in\Sigma^*} |r|(w)$ to bound the accuracy of this solution.

A *multiplicity automaton (MA)* is a tuple $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$, where Q is a finite set of states, φ (resp. ι, τ) is a transition (resp. initialization, termination) function, which can be used to compute a rational series r_A. Any rational series r can be computed by a MA. Given a MA A, we obtain a new MA $|A|$ by taking the absolute values of the functions φ, ι and τ. It is straightforward that r_A is absolutely convergent if $r_{|A|}$ is simply convergent. We prove that any absolutely convergent rational series r can be computed by a multiplicity automaton A which has the property that $r_{|A|}$ is simply convergent. Then, we provide an algorithm which takes a multiplicity automaton B as input and halts if and only if r_B is absolutely convergent: when the algorithm halts, it outputs a MA A equivalent to B, i.e. the series computed by A and B are equal, and such that $r_{|A|}$ is convergent. So, we have proved that the set $\mathcal{A}^{rat}(\Sigma)$ composed of all absolutely convergent rational series is semi-decidable.

The sum $\sum_{w\in\Sigma^*} |r(w)|$ can be estimated from below by computing the sum $\sum_{w\in\Sigma^{\leq n}} |r(w)|$ for increasing integers n. We prove that our algorithm can be used to provide convergent upper bounds. So, the sum $\sum_{w\in\Sigma^*} |r(w)|$ can be estimated to any accuracy rate. As a consequence, if the $L1$-distance $||r - s||_1 = \sum_{w\in\Sigma^*} |r(w) - s(w)|$ is finite, it can be estimated to any accuracy rate. It has been proved in [5] that computing the $L1$-distance between two hidden Markov models is NP-hard which implies that computing the $L1$-distance between two MA is NP-hard too. L_p distances of two probabilistic automata have been studied in [6,7], where efficient algorithm have been provided when p *is even*. Our algorithm can be used to estimate L_p distances between two rational series for any odd values of p.

Finally, for any rational series r, we introduce a spectral radius-like parameter, $\rho_{|r|}$ defined by $\rho_{|r|} = limsup_n(|r|(\Sigma^n))^{1/n}$ and we show that r is absolutely convergent iff $\rho_{|r|} < 1$.

We recall some properties on rational series and multiplicity automata in Section 2. We study absolutely convergent series in Section 3; in particular, we prove that any absolutely convergent rational series can be represented by a MA A such that $r_{|A|}$ is convergent. We prove the semi-decidability of the class $\mathcal{A}^{rat}(\Sigma)$ in Section 4. We show how the sum $\sum_{w\in\Sigma^*} |r(w)|$ can be estimated in

Section 5. To conclude, we provide some comments and describe some conjectures and future works in Section 6.

2 Preliminaries

2.1 Rational Series

Let Σ^* be the set of words on the finite alphabet Σ. The empty word is denoted by ε, and the length of a word u is denoted by $|u|$. For any integer k, we denote by Σ^k the set $\{u \in \Sigma^* \mid |u| = k\}$ and by $\Sigma^{\leq k}$ the set $\{u \in \Sigma^* \mid |u| \leq k\}$. A subset S of Σ^* is *prefix-closed* if for any $u, v \in \Sigma^*$ $uv \in S \Rightarrow u \in S$.

The general context is, for an alphabet Σ the set $\mathbb{R}\langle\langle\Sigma\rangle\rangle$ of all the mappings from Σ^* into $\mathbb{R}$. An element of this set is called a *formal power series*. This set is an $\mathbb{R}$-vector space. For any series r and any word $u \in \Sigma^*$, we denote by $\dot{u}r$ the series defined by $\dot{u}r(w) = r(uw)$. Let $LH(r)$ denote the *linear hull* of r, i.e. the vector subspace of $\mathbb{R}\langle\langle\Sigma\rangle\rangle$ spanned by $\{\dot{u}r | u \in \Sigma^*\}$.

A *multiplicity automaton (MA)* is a tuple $\langle\Sigma, Q, \varphi, \iota, \tau\rangle$ where Q is a finite set of states, $\varphi : Q \times \Sigma \times Q \to \mathbb{R}$ is the *transition function*, $\iota : Q \to \mathbb{R}$ is the *initialization function* and $\tau : Q \to \mathbb{R}$ is the *termination function*. Let $Q_I = \{q \in Q | \iota(q) \neq 0\}$ be the set of *initial states* and $Q_T = \{q \in Q | \tau(q) \neq 0\}$ be the set of *terminal states*. We extend the transition function φ to $Q \times \Sigma^* \times Q$ by $\varphi(q, wx, q') = \sum_{q'' \in Q} \varphi(q, w, q'')\varphi(q'', x, q')$ and $\varphi(q, \varepsilon, q') = 1$ if $q = q'$ and 0 otherwise, for any $q, q' \in Q$, $x \in \Sigma$ and $w \in \Sigma^*$. For any finite subset $L \subset \Sigma^*$ and any $Q' \subseteq Q$, define $\varphi(q, L, Q') = \sum_{w \in L, q' \in Q'} \varphi(q, w, q')$. For any MA $A = \langle\Sigma, Q, \varphi, \iota, \tau\rangle$, we define the series r_A by $r_A(w) = \sum_{q,q' \in Q} \iota(q)\varphi(q, w, q')\tau(q')$. For any $q \in Q$, we define the series $r_{A,q}$ by $r_{A,q}(w) = \sum_{q' \in Q} \varphi(q, w, q')\tau(q')$. The *support* of a MA $\langle\Sigma, Q, \varphi, \iota, \tau\rangle$ is a non deterministic finite automaton (NFA) $\langle\Sigma, Q, \delta, Q_I, Q_F\rangle$ where the transition function is defined by $\delta(q, x) = \{q' \in Q | \varphi(q, x, q') \neq 0\}$.

We will say that a series r is *rational* if it satisfies one of the two following equivalent conditions:

1. the dimension of $LH(r)$ is finite;
2. r can be computed by a multiplicity automaton.

The family of all rational series is denoted by $\mathbb{R}^{rat}\langle\langle\Sigma\rangle\rangle$.

2.2 Prefixial Multiplicity Automata

Representation of rational series based on prefix sets has been introduced in [1].

Definition 1. *Let $A = \langle\Sigma, Q, \varphi, \iota, \tau\rangle$ be a MA. We say that A is* prefixial *if:*

- *Q is non-empty prefix-closed finite subset of Σ^**
- *$\forall u \in Q, \iota(u) \neq 0$ iff $u = \varepsilon$*
- *$\forall x \in \Sigma, \forall u, v \in Q$ s.t. $ux \in Q$, $\varphi(u, x, v) \neq 0$ iff $v = ux$.*

A transition (u, x, v) *is called an* inner transition *if* $v = ux$ *and a* border transition *otherwise. The set* $F = \{ux | u \in Q, x \in \Sigma, ux \notin Q\}$ *is called the* frontier set *of* A.

An NFA $A = \langle \Sigma, Q, \delta, I, F \rangle$ *is* prefixial *if* A *is the support of a prefixial MA.*

A prefix-closed subset Q of Σ^* can be used as the set of states of a prefixial automaton that computes a rational series r if and only if the set $\{\dot{u}r | u \in Q\}$ spans $LH(r)$. Let us make this statement precise.

Let r be a rational series and let Q be a prefix-closed subset such that $\{\dot{u}r | u \in Q\}$ spans $LH(r)$. Let $F = \{ux | u \in Q, x \in \Sigma, ux \notin Q\}$. Let $f : Q \to \mathbb{R}$ and $g : F \times Q \to \mathbb{R}$ be two mappings that satisfy:

1. $f(u) \neq 0$ for every state u,
2. $\dot{\overline{ux}}r = \sum_{v \in Q} g(ux, v) \frac{\overline{f}(u)}{\overline{f}(v)} \dot{v}r$ for every $ux \in F$ where $\overline{f}$ is defined by $\overline{f}(\varepsilon) = f(\varepsilon)$, where $\overline{f}(wx) = \overline{f}(w) f(wx)$ for any $w \in \Sigma^*$ and $x \in \Sigma$ and where the top bar notation is meant to express that the "dot" applies to the element under the bar. The function g expresses linear dependencies.

We define the prefixial automaton $A(\Sigma, Q, f, g, r) = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ by

- $\iota(\varepsilon) = f(\varepsilon)$,
- $\forall u \in Q, x \in \Sigma$ s.t. $ux \in Q$, $\varphi(u, x, ux) = f(ux)$,
- $\forall u, v \in Q, x \in \Sigma$ s.t. $ux \in F$, $\varphi(u, x, v) = g(ux, v)$,
- $\forall u \in Q, \tau(u) = \frac{r(u)}{\overline{f}(u)}$.

Proposition 1. *The automaton* $A(\Sigma, Q, f, g, r)$ *computes* r.

Proof. Let $A(\Sigma, Q, f, g, r) = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$. Let us show, by induction on $|w|$, that for any $u \in Q$ and any $w \in \Sigma^*$, $\dot{u}r(w) = \overline{f}(u) r_{A,u}(w)$.

- Let $u \in Q$: $\dot{u}r(\varepsilon) = r(u)$ and $\overline{f}(u) r_{A,u}(\varepsilon) = \overline{f}(u)\tau(u) = r(u)$.
- Let $u \in Q$, $w \in \Sigma^*$ and $x \in \Sigma$.

If $ux \in Q$,
$$\begin{aligned} \overline{f}(u) r_{A,u}(xw) &= \overline{f}(u) \varphi(u, x, ux) r_{A,ux}(w) \\ &= \overline{f}(ux) r_{A,ux}(w) \text{ by definition of } \overline{f} \\ &= \overline{f}(ux) \frac{\dot{\overline{ux}}r(w)}{\overline{f}(ux)} \text{ by induction hypothesis} \\ &= \dot{\overline{ux}}r(w). \end{aligned}$$

If $ux \notin Q$, $\overline{f}(u) r_{A,u}(xw) = \overline{f}(u) \sum_{v \in Q} g(ux, v) r_{A,v}(w)$ by definition of A
$$\begin{aligned} &= \overline{f}(u) \sum_{v \in Q} g(ux, v) \frac{\dot{v}r(w)}{\overline{f}(v)} \text{ by induction hypothesis} \\ &= \dot{\overline{ux}}r(w) \text{ by definition of } g. \end{aligned}$$

Therefore, for any $u \in Q$, $\dot{u}r = \overline{f}(u) r_{A,u}$ and in particular, $r = f(\varepsilon) r_{A,\varepsilon} = r_A$. □

3 On Representation of Absolutely Convergent Rational Series

3.1 Absolutely Convergent Rational Series

Let r be a series and let Γ be a non-empty subset of Σ^*. We say that r is *convergent* on Γ if the sum $\sum_{n\geq 0}\sum_{w\in\Gamma\cap\Sigma^n} r(w)$ is convergent; if so, we denote the sum by $r(\Gamma)$. Let $|r|$ be the series defined by $|r|(w) = |r(w)|$. We say that r is *absolutely convergent* if $|r|$ is convergent. Note that when a series r is absolutely convergent, it is convergent over any $\Gamma \subseteq \Sigma^*$.

We denote by $\mathcal{A}(\Sigma)$ (resp. by $\mathcal{A}^{rat}(\Sigma)$) the subspace of $\mathbb{R}\langle\langle\Sigma\rangle\rangle$ (resp. of $\mathbb{R}^{rat}\langle\langle\Sigma\rangle\rangle$) composed of the series that are absolutely convergent.

Let r be a series, we denote by $res(r)$ the following subset of Σ^*: $res(r) = \{u \in \Sigma^*/\exists w \in \Sigma^*, r(uw) \neq 0\}$. For any absolutely convergent series r and any word $u \in res(r)$, we denote by $u^{-1}r$ the series defined by $u^{-1}r(w) = \frac{|r|(\Sigma^*)}{|r|(u\Sigma^*)}\dot{u}r$ and we call it the *residual* of r associated with u. The set of all the residuals of r is denoted by $Res(r)$. The vector subspace spanned by $Res(r)$ is equal to $LH(r)$. Note that for any $u \in res(r), |u^{-1}r|(\Sigma^*) = |r|(\Sigma^*)$. The mapping $r \to |r|(\Sigma^*)$ defines a norm $||\cdot||_1$ on $\mathcal{A}(\Sigma)$. Let us denote by $CH(r)$ the convex hull of $Res(r)$: $CH(r) = \{\sum_{i=1}^n \alpha_i u_i^{-1}r | n \in \mathbb{N}, \alpha_i \geq 0, \sum_{i=1}^n \alpha_i = 1, u_i \in res(r)\}$. Let us denote by $CCH(r)$, the closed convex hull of $Res(r)$, i.e. the closure of $CH(r)$. Note that when $r \in \mathcal{A}^{rat}(\Sigma)$, $||\cdot||_1$ is constant on $CCH(r)$. In particular, $CCH(r)$ is a compact convex set.

Lemma 1. *Let $r \in \mathcal{A}^{rat}(\Sigma)$. Then $\forall\epsilon > 0, \exists k \in \mathbb{N}$ such that $\forall s \in CCH(r)$, $|s|(\Sigma^{>k}) < \epsilon$. In particular, $\forall\epsilon > 0, \exists k \in \mathbb{N}$ such that $\forall u \in res(r)$, $|r|(u\Sigma^{>k}) < \epsilon|r|(u\Sigma^*)$.*

Proof. For any integer k, let $f_k : CCH(r) \to \mathbb{R}$ be defined by $f_k(s) = |s|(\Sigma^{>k})$. For any $s, t \in CCH(r)$, we have $|f_k(s) - f_k(t)| \leq ||s-t||_1$. Hence, f_k is continuous for any k. Moreover, $lim_{k\to\infty} f_k(s) = 0$ for any $s \in CCH(r)$. Since $CCH(r)$ is compact, (f_k) converges uniformy to 0: for any $\epsilon > 0$, there exists $K \geq 0$ s.t. for any $k \geq K$ and any $s \in CCH(r)$, $|f_k(s)| < \epsilon$. Apply the result to $s = u^{-1}r$ and $\epsilon/|r|(\Sigma^*)$ to obtain the second result. □

Lemma 2. *Let $r \in \mathcal{A}^{rat}(\Sigma)$ and let $d = dim(LH(r))$. For all $\epsilon > 0$, there exists an integer N such that for any $u \in res(r)$, there exists $v_1, \ldots, v_d \in res(r) \cap \Sigma^{\leq N}$ and $\alpha_1, \ldots, \alpha_d > -\epsilon$ such that $\sum_{1\leq i\leq d} \alpha_i < 1 + \epsilon$ and $u^{-1}r = \sum_{1\leq i\leq d} \alpha_i v_i^{-1}r$.*

Proof. Suppose that there exists $\epsilon > 0$ such that for any integer n, there exists $u_n \in res(r)$ such that for any $v_1, \ldots, v_d \in res(r) \cap \Sigma^{\leq n}$, $u_n^{-1}r = \sum_{i=1}^d \alpha_i v_i^{-1}r$ implies that there exists an index i s.t. $\alpha_i \leq -\epsilon$, or $\sum_{i=1}^d \alpha_i \geq 1 + \epsilon$. Let n_k be a subsequence such that $u_{n_k}^{-1}r$ converges to an absolutely convergent series $s \in CCH(r)$.

Now, let $v_1, \ldots, v_{d-1}$ be such that $v_1^{-1}r, \ldots, v_{d-1}^{-1}r, s$ form a basis of $LH(r)$. For any integer k, let $u_{n_k}^{-1}r = \alpha_{1,k}v_1^{-1}r + \ldots + \alpha_{d-1,k}v_{d-1}^{-1}r + \alpha_{d,k}s$. Since $u_{n_k}^{-1}r$ converges to s, $\alpha_{i,k}$ converges to 0 for $i = 1, \ldots, d-1$ and $\alpha_{d,k}$ converges to 1

when k tends to infinity. Therefore, there should exist an integer K such that for any $k \geq K$, each coefficient of this combination is strictly greater that $-\epsilon$, and the sum of all its coefficient is strictly lower than $1+\epsilon$, which is contradictory. □

3.2 A Particular Representation of Absolutely Convergent Rational Series

Let $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ be an MA. Let us denote by $|A|$ the MA defined by $|A| = \langle \Sigma, Q, |\varphi|, |\iota|, |\tau| \rangle$.

Lemma 3. *$|r_A| \leq r_{|A|}$. Hence, if $r_{|A|}$ is convergent, then r_A is absolutely convergent.*

Proof. Indeed,

$$|r_A(w)| = |\sum_{q,q' \in Q} \iota(q)\varphi(q,w,q')\tau(q')| \leq \sum_{q,q' \in Q} |\iota(q)\varphi(q,w,q')\tau(q')| = r_{|A|}(w). \qquad \square$$

Lemma 4. *Let $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ be a MA. Suppose that φ takes only non negative values and that there exists an integer k such that for any state q, $\varphi(q, \Sigma^k, Q) < 1$. Then, the series $r_{|A|}$ is convergent.*

Proof. Let $R = \sup\{\varphi(q, \Sigma^h, Q) | q \in Q, h < k\}$ and $\rho = \sup\{\varphi(q, \Sigma^k, Q) | q \in Q\}$. From the hypothesis, $\rho < 1$.

Since for any state q and any integers $n > 0$ and $0 \leq h < k$, $\varphi(q, \Sigma^{nk+h}, Q) = \sum_{q' \in Q} \varphi(q, \Sigma^{(n-1)k+k}, q')\varphi(q', \Sigma^h, Q)$, it can easily be shown by induction on n that $\varphi(q, \Sigma^{nk+h}, Q) \leq R\rho^n$.

Let $I = Sup\{|\iota(q)| \text{ for } q \in Q\}$ and $T = Sup\{|\tau(q)| \text{ for } q \in Q\}$. We have

$$|r_A|(\Sigma^{nk+h}) \leq \sum_{q,q' \in Q} |\iota(q)|\varphi(q, \Sigma^{nk+h}, q')|\tau(q')| \leq IT \sum_{q \in Q} \varphi(q, \Sigma^{nk+h}, Q) \leq ITR|Q|\rho^n$$

$$\text{Therefore, } |r_A|(\Sigma^*) = \sum_{n \geq 0} \sum_{h=0}^{k-1} |r_A|(\Sigma^{nk+h}) \leq ITR|Q|k \sum_{h \geq 0} \rho^n = \frac{ITR|Q|k}{1-\rho}.$$

Note that if A is prefixial, there exists a unique state q such that $\iota(q) \neq 0$. So we can take $R = \sup\{\varphi(\varepsilon, \Sigma^h, Q) | q \in Q, h < k\}$ and we have $|r_A|(\Sigma^{nk+h}) \leq ITR\rho^n$. and $|r_A|(\Sigma^*) \leq \frac{ITRk}{1-\rho}$. □

Theorem 1. *Let $r \in \mathcal{A}^{rat}(\Sigma)$. Let $\rho < 1$. There exists an integer n, and a prefixial MA $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ that computes r and such that $\forall u \in Q$, $|\varphi|(u, \Sigma^n, Q) < \rho$. Hence, the series $r_{|A|}$ is convergent.*

Proof. Let $r \in \mathcal{A}^{rat}(\Sigma)$ and let $d = dim(LH(r))$. Let $\rho < 1$ and let $\varepsilon = \rho/16$. From Lemma 1, let k be an integer such that $\forall u \in res(r)$, $\frac{|r|(u\Sigma^{\geq k})}{|r|(u\Sigma^*)} < \epsilon$. Let $\epsilon' = (d|\Sigma|)^{-1}$. From Lemma 2, let $N > k$ be an integer such that for all $u \in$

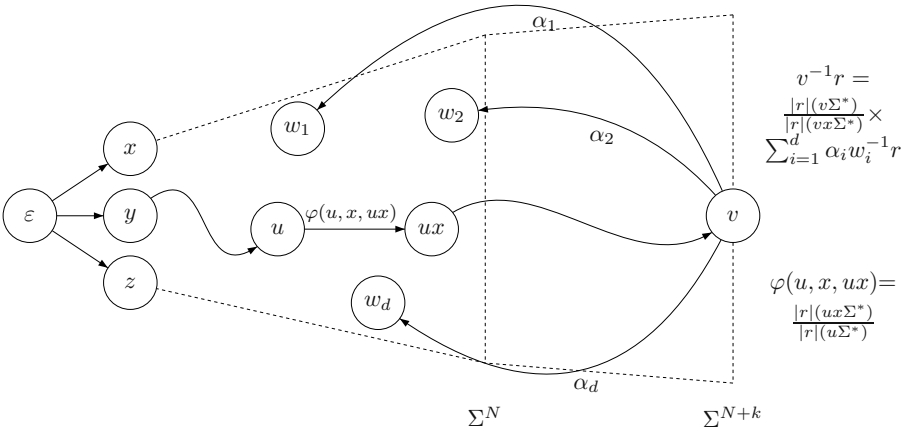


Fig. 1. Automaton built in the proof of Theorem 1

$res(r)$, there exists $v_{u,1}, \ldots, v_{u,d} \in res(r) \cap \Sigma^{\leq N}$ and $\alpha_{u,1}, \ldots, \alpha_{u,d} > -\epsilon'$ such that $\sum_{i=1}^{d} \alpha_{u,i} < 1 + \epsilon' \leq 2$ and $u^{-1}r = \sum_{i=1}^{d} \alpha_{u,i} v_{u,i}^{-1} r$.

Let $Q = \Sigma^{N+k} \cap res(r)$. Let $f : Q \to \mathbb{R}$ be defined by $f(\varepsilon) = 1$ and $\forall ux \in Q, f(ux) = \frac{|r|(ux\Sigma^*)}{|r|(u\Sigma^*)}$ where $u \in \Sigma^*$ and $x \in \Sigma$. One can easily show that $\overline{f}(u) = |r|(u\Sigma^*)/|r|(\Sigma^*)$.

Let $F = \{ux | u \in Q, x \in \Sigma, ux \notin Q\}$. For any $ux \in F \cap res(r)$, there exist coefficients $(\alpha_{ux,v})_{v \in \Sigma^{\leq N}}$ such that (i) at most d coefficients $\alpha_{ux,v}$ are not null; (ii) $(ux)^{-1}r = \sum_{v \in \Sigma^{\leq N}} \alpha_{ux,v} v^{-1} r$; (iii) $\alpha_{ux,v} > -\epsilon'$; (iv) $\sum_{v \in \Sigma^{\leq N}} \alpha_{ux,v} < 1 + \epsilon'$. From $(ux)^{-1}r = \sum_{v \in \Sigma^{\leq N}} \alpha_{ux,v} v^{-1} r$, then $\dot{\overline{ux}} r = |r|(ux\Sigma^*) \sum_{v \in \Sigma^{\leq N}} \frac{\alpha_{ux,v}}{|r|(v\Sigma^*)} \dot{v} r$.

Let $g : Q \times F \to \mathbb{R}$ be defined by $g(ux, v) = \alpha_{ux,v} \frac{|r|(ux\Sigma^*)}{|r|(u\Sigma^*)}$ if $ux \in F \cap res(r)$ and 0 otherwise. One can check that the conditions on f and g stated in Section 2.2 are satisfied. (see Fig. 1)

From the proposition 1, the MA $A(\Sigma, Q, f, g, r) = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ computes r.

Let us list below some properties of A:

1. $\forall u \in Q \cap \Sigma^{<N+k}$ and $x \in \Sigma$, $\varphi(u, x, ux) = \frac{|r|(ux\Sigma^*)}{|r|(u\Sigma^*)} \leq 1$;
2. $\forall u \in Q \cap \Sigma^{<N+k}$ and $h < N + k - |u|$, $\varphi(u, \Sigma^h, Q) = \frac{|r|(u\Sigma^{\geq h})}{|r|(u\Sigma^*)} \leq 1$;
3. $\forall u \in Q \cap \Sigma^{\leq N}$, $\varphi(u, \Sigma^k, Q) = \frac{|r|(u\Sigma^{\geq k})}{|r|(u\Sigma^*)} < \varepsilon$;
4. $\forall u \in Q \cap \Sigma^{N+k}$ and $x \in \Sigma$, $\varphi(u, x, v) = \alpha_{ux,v} \frac{|r|(ux\Sigma^*)}{|r|(u\Sigma^*)} > -\varepsilon'$;
5. $\forall u \in Q \cap \Sigma^{N+k}$, $\varphi(u, \Sigma, Q) < 1 + \varepsilon' \leq 2$;
6. $\forall u \in \Sigma^{N+k}$, $(|\varphi|(u, \Sigma, Q) - \varphi(u, \Sigma, Q)) < 2d|\Sigma|\epsilon'$.

From these properties, one can deduce that:

a) For all $u \in \Sigma^h, h \leq N - k$ one have

$$|\varphi|(u, \Sigma^{2k}, Q) = \sum_{w \in \Sigma^k} \varphi(u, w, uw)\varphi(uw, \Sigma^k, Q) < \epsilon^2 < \rho \tag{1}$$

by applying twice the property 3.

b) For $u \in \Sigma^{N-k+1}$ one have

$$|\varphi|(u, \Sigma^{2k}, Q) < \epsilon(2 + 2d|\Sigma|\epsilon') < \rho. \tag{2}$$

Indeed, $|\varphi|(u, \Sigma^{2k}, Q) =$

$$\sum_{|w|=2k-1} \varphi(u, w, uw)[\varphi(uw, \Sigma, Q) + (|\varphi|(uw, \Sigma, Q) - \varphi(uw, \Sigma, Q))]$$

$$\leq \sum_{|w|=2k-1} \varphi(u, w, uw)[2 + 2d|\Sigma|\epsilon'] \text{ from properties 5 and 6}$$

$$\leq \varepsilon(2 + 2d|\Sigma|\epsilon') \text{ from properties 2 and 3.}$$

c) For $u \in \Sigma^h$ where $N - k + 1 < h \leq N$ one have

$$|\varphi|(u, \Sigma^{2k}, Q) < \epsilon(2 + 2d|\Sigma|\epsilon') < \rho. \tag{3}$$

Indeed, $|\varphi|(u, \Sigma^{2k}, Q) =$

$$\sum_{|w|=N+k-h} \varphi(u, w, uw) \sum_{|v|\leq N} |\varphi|(uw, \Sigma, v)\varphi(v, \Sigma^{k+h-N-1}, Q)$$

$$\leq \sum_{|w|=N+k-h} \varphi(u, w, uw) \sum_{|v|\leq N} |\varphi|(uw, \Sigma, v) \text{ from property 2}$$

$$\leq \sum_{|w|=N+k-h} \varphi(u, w, uw)[\varphi(uw, \Sigma, Q) + (|\varphi|(uw, \Sigma, Q) - \varphi(uw, \Sigma, Q))]$$

$$\leq \sum_{|w|=N+k-h} \varphi(u, w, uw)[2 + 2d|\Sigma|\epsilon'] \text{ from properties 5 and 6}$$

$$\leq \varepsilon(2 + 2d|\Sigma|\epsilon') \text{ from properties 2 and 3.}$$

d) For $u \in \Sigma^h, N < h \leq N + k$ one have

$$|\varphi|(u, \Sigma^{2k}, Q) < \varepsilon(2 + 2d|\Sigma|\epsilon')^2 < \rho. \tag{4}$$

Indeed,

$$|\varphi|(u, \Sigma^{2k}, Q) < \sum_{w\in\Sigma^{N+k-h}} \varphi(u, w, uw) \sum_{v\in\Sigma^{\leq N}} |\varphi|(uw, \Sigma, v)|\varphi|(v, \Sigma^{h+k-N-1}, Q).$$

If $|v| \leq 2N - h + 1$, $|v| + h + k - N - 1 \leq N + k$, and from Property 3,

$$|\varphi|(v, \Sigma^{h+k-N-1}, Q) \leq \epsilon$$

If $|v| > 2N - h + 1$, $|\varphi|(v, \Sigma^{h+k-N-1}, Q)$

$$\leq \sum_{|w|=N+k-|v|} \varphi(v, w, vw) \sum_{v'\in\Sigma^{\leq N}} |\varphi|(vw, \Sigma, v')\varphi(v', \Sigma^{|v|+h-2N-2}, Q)$$

$$\leq \sum_{|w|=N+k-|v|} \varphi(v, w, vw)|\varphi|(vw, \Sigma, Q)$$

$$\leq (2 + 2d|\Sigma|\epsilon')\varphi(v, \Sigma^{N+k-|v|}, Q)$$

$$\leq \varepsilon(2 + 2d|\Sigma|\epsilon').$$

That is, in all cases, $|\varphi|(v, \Sigma^{h+k-N-1}, Q) \leq \varepsilon(2 + 2d|\Sigma|\epsilon')$ and

$$|\varphi|(u, \Sigma^{2k}, Q) < \varepsilon(2 + 2d|\Sigma|\epsilon') \sum_{w \in \Sigma^{N+k-h}} \varphi(u, w, uw) \sum_{v \in \Sigma^{\leq N}} |\varphi|(uw, \Sigma, v)$$
$$\leq \varepsilon(2 + 2d|\Sigma|\epsilon')^2$$

We have proved that for any $u \in Q$, $|\varphi|(u, \Sigma^{2k}, Q) < \rho$. Hence, from lemma 4, the series $r_{|A|}$ is convergent. □

Spectral radius of a matrix, joint or generalized spectral radius of a set of matrices are tools used to study asymptotic properties of powers or products of matrices. A good introduction on spectral radii can be found in the first chapters of [8].

Definition 2. *Let $r \in \mathbb{R}^{rat}\langle\langle\Sigma\rangle\rangle$. We define the* absolute spectral radius

$$\rho_{|r|} = \limsup_n \left(|r|(\Sigma^n)\right)^{1/n}$$

Proposition 2. *Let $r \in \mathbb{R}^{rat}\langle\langle\Sigma\rangle\rangle$. $r \in \mathcal{A}^{rat}(\Sigma)$ if and only if $\rho_{|r|} < 1$.*

Proof. Let $r \in \mathbb{R}^{rat}\langle\langle\Sigma\rangle\rangle$. Suppose that $\rho_{|r|} < 1$, then there exists ρ s.t. $\rho_{|r|} < \rho < 1$ and $n \in \mathbb{N}$ such that $|r|(\Sigma^{\geq n}) < \sum_{i=n}^{\infty} \rho^i$. Thus r is absolutely convergent. Suppose now that $r \in \mathcal{A}^{rat}(\Sigma)$. By the theorem 1, there exists a prefixial automaton $A = \langle \Sigma, Q, \varphi, \iota, \tau\rangle$ that computes r, $\rho < 1$ and an integer n such that for every state q, $|\varphi_A|(q, \Sigma^n, Q) < \rho$. Hence, from Lemma 4, $|r|(\Sigma^m) = O(\rho^{m/n})$ and therefore, $\rho_{|r_A|} \leq \rho^{1/n} < 1$. □

4 Decidability

We prove in this section that there exists an algorithm that takes a multiplicity automaton A as input and halts iff r_A is absolutely convergent. In other words, the class $\mathcal{A}(\Sigma)$ is semi-decidable.

Theorem 2. *The class $\mathcal{A}^{rat}(\Sigma)$ is semi-decidable.*

Proof. Let r be a rational series and let A be a MA that computes r. Queries such as: What is the dimension d of $LH(r)$? Does the word u belongs to $res(r)$? Given $u_1, \ldots, u_d \in res(r)$, is $\{\dot{u_1}r, \ldots, \dot{u_d}r\}$ a basis of $LH(r)$? can be answered by using A.

We consider the countable class $\mathcal{S}_r$ composed of all the prefixial NFA $\langle \Sigma, Q, \delta, I, T\rangle$ that satisfy the following properties:

- there exists two integers N and k such that $Q = \Sigma^{\leq N+k} \cap res(u)$;
- for any $u \in Q$ and $x \in \Sigma$ such that $ux \in res(u)\backslash Q$, the set $\delta(u, x)$ is included in $\Sigma^{\leq N}$ and the set $\{\dot{v}r | v \in \delta(u, x)\}$ forms a basis of $LH(r)$. In particular, $\delta(u, x)$ contains exactly d elements.

Let $(A_m)_{m\in\mathbb{N}}$ be an enumeration of $\mathcal{S}_r$.

Now let $(f_n)_{n\in\mathbb{N}}$ be a family of functions defined as follows: for any integer n, $f_n : res(r) \to \mathbb{R}$, $f_n(\varepsilon) = 1$ and for any $x \in \Sigma$ and $ux \in res(r)$, $f_n(ux) = \frac{|r|(ux\Sigma^{\leq n})}{|r|(u\Sigma^{\leq n+1})}$ if $|r|(u\Sigma^{\leq n}) \neq 0$ and $f_n(ux) = 1$ otherwise. Note that $\lim_{n\to\infty} f_n(ux) = \frac{|r|(ux\Sigma^*)}{|r|(u\Sigma^*)}$.

Let $m \in \mathbb{N}$. For any integer n, we define a MA $A_{m,n}$ whose support is equal to $A_m = \langle \Sigma, Q_m, \delta_m, \{\varepsilon\}, T_m\rangle$. Let $f : Q_m \to \mathbb{R}$ be defined by $f(u) = f_n(u)$. Let $F_m = \{ux | u \in Q_m, x \in \Sigma, ux \notin Q_m\}$ and let $g : F_m \times Q_m \to \mathbb{R}$ be defined by $g(ux, v) = 0$ if $v \notin \delta_m(u, x)$ and $\overline{ux}r = \sum_{v\in Q} g(ux, v)\frac{\overline{f}(u)}{\overline{f}(v)}\dot{vr}$: note that g is completely determined by A_m, f_n and r since for any $ux \in F_m$, $\{\dot{vr} | v \in \delta(u, x)\}$ forms a basis of $LH(r)$. We let $A_{m,n} = A(\Sigma, Q_m, f, g, r)$.

Now, consider the following algorithm:

- enumerate $(m, n, k) \in \mathbb{N}$
- for each tuple (m, n, k), build the MA $A_{m,n} = \langle \Sigma, Q_m, \varphi_{m,n}, \iota_{m,n}, \tau_{m,n}\rangle$
- if $Max_{q\in Q_m}\{|\varphi_{m,n}|(q, \Sigma^k, Q_m)\} < 1$, halts.

If r is absolutely convergent, from Theorem 1, there exists a prefixial MA $A = \langle \Sigma, Q_m, \varphi, \iota, \tau\rangle$ that computes r, whose support is A_m for some integer m and such that $Max_{q\in Q_m}\{\varphi(q, \Sigma^k, Q_m)\} < \rho$ for some integer k and some $\rho < 1$. Since $Max_{q\in Q_m}\{\varphi(q, \Sigma^k, Q_m)\}$ is a continuous function in the parameters of A, and since $\lim_{n\to\infty} \varphi_{m,n}(q, x, q') = \varphi(q, x, q')$ for any states q, q' and any letter x, for any $\rho < \rho' < 1$, there exists an integer N such that $Max_{q\in Q_m}\{\varphi_{m,n}(q, \Sigma^k, Q_m)\} < \rho'$ for any integer $n \geq N$. Hence, the algorithm halts on input r.

Clearly, from Lemma 3, since any MA $A_{m,n}$ computes r, the algorithm does not halt if r is not absolutely convergent. □

5 Approximation and L_1-Distance

To our knowledge, the sum $|r|(\Sigma^*)$ cannot be exactly computed. Nevertheless, it is possible to estimate it by a lower bound since for any integer n, $|r|(\Sigma^{\leq n}) \leq |r|(\Sigma^*)$ and $\lim_{n\to\infty} |r|(\Sigma^{\leq n}) = |r|(\Sigma^*)$. We will now define upper bounds of $|r|(\Sigma^*)$ which tend to $|r|(\Sigma^*)$. Hence, it is possible to bound the error made by the approximation to any accuracy rate.

We first give a variation of Theorem 1.

Lemma 5. *Let $r \in \mathcal{A}^{rat}(\Sigma)$. For any $u \in res(r)$ and any $x \in \Sigma$, let $f_n(ux) = \frac{|r|(ux\Sigma^{\leq n})}{|r|(u\Sigma^{\leq n+1})}$ if $|r|(u\Sigma^{\leq n+1}) \neq 0$ and $f_n(ux) = 1$ otherwise: $\lim_{n\to\infty} f_n(ux) = f(ux) = \frac{|r|(ux\Sigma^*)}{|r|(u\Sigma^*)}$. Let $T > |r|(\Sigma^*)$.*

For any integers N and k, let $D : \Sigma^{N+k+1} \to 2^{\Sigma^{\leq N}}$ be such that for any $ux \in \Sigma^{N+k+1}$, $\{\dot{vr} | v \in D(ux)\}$ forms a basis of $LH(r)$. In particular, $|D(ux)| = d = dim(LH(r))$. Given N, k, D and f_n, there exists a unique function $g : \Sigma^{N+k+1} \times \Sigma^{\leq N} \to \mathbb{R}$ such that $g(ux, v) = 0$ if $v \notin D(ux)$ and such that $A(\Sigma, Q, f_n, g, r)$ computes r (see Proposition 1). Let us denote it by $A(N, k, D, f_n)$.

Now given $\rho < 1$, there exists k_0, N_0 such that $\forall N > N_0, \forall k > k_0$ there exists n and D such that $A = A(N, k, D, f_n)$ satisfies:

- $\varphi_{|A|}(u, \Sigma^{2k}, Q) < \rho$
- $\forall u \in \Sigma^{N+k}, |\tau(u)| < T$

Proof. By considering the proof of Theorem 1, it can be proved that there exists integers N_0 and k_0 such that $\forall N > N_0, \forall k > k_0$ there exists $D : \Sigma^{N+k+1} \to 2^{\Sigma^{\leq N}}$ such that $A = A(N, k, D, f)$ satisfies $\varphi_{|A|}(u, \Sigma^{2k}, Q) < \rho/2$ and $\forall u \in \Sigma^{N+k}, |\tau(u)| \leq |r|(\Sigma^*)$. Since $\varphi_{|A(N,k,D,f_n)|}(u, \Sigma^{2k}, Q)$ and $|\tau_{|A(N,k,D,f_n)|}(u)|$ are continuous relatively to the transition coefficients, there exists an integer n such that the conclusion holds. □

Proposition 3. *Let $r \in \mathcal{A}^{rat}(\Sigma)$. Let $\mathbb{A} = \{A(N, k, D, f_n)/N, k, n \in \mathbb{N}, D \in \mathcal{P}(\Sigma^N)\}$. Then $\inf(r_{|A|}(\Sigma^*))_{A\in\mathbb{A}} = |r|(\Sigma^*)$.*

Proof. Let N_0 and k_0 be such as in Lemma 5. We define a sequence $(A_z)_{z\in\mathbb{N}}$ of automata:

- $N_z = N_0 + z$, $k_z = k_0$
- $\forall z$, n_z and D_z are such as in Lemma 5, $A_z = A(N_z, k_z, D_z, f_{n_z})$.

As N_z grows, $r_{|A_z|}$ converges pointwisely to $|r|$. We have $\|\tau_z\|_\infty < T$ by construction. We have $R = sup_{h<2k_z}(|\varphi_z|(\varepsilon, \Sigma^{2k_z}, Q)) < 1$, as $|\varphi_z|(u, \Sigma, Q)) < 1$ for $u \in \Sigma^{<N_z+k_z}$. Applying Lemma 4, we have $r_{|A_z|}(\Sigma^{>n}) < T\rho^{n/2k_0-1}$ and therefore $|r|(\Sigma^{>n}) < T\rho^{n/2k_0-1}$. Now, $|r_{|A_z|}(\Sigma^*) - |r|(\Sigma^*)| \leq \sum_{w\in\Sigma^*} |r_{|A_z|}(w) - |r|(w)| \leq \sum_{w\in\Sigma^{\leq z_1}} |r_{|A_z|}(w) - |r|(w)| + \sum_{w\in\Sigma^{>z_1}} r_{|A_z|}(w) + \sum_{w\in\Sigma^{>z_1}} |r|(w)$. For $\epsilon > 0$, one can find Z_1 such that both $\sum_{w\in\Sigma^{>z_1}} r_{|A_z|}(w) < \epsilon/3$ and $\sum_{w\in\Sigma^{>z_1}} |r|(w) < \epsilon/3$. Finally, as $r_{|A_z|}$ converges pointwisely to $|r|(w)$, one can find Z_2 such that $\sum_{w\in\Sigma^{\leq z_1}} |r_{|A_z|}(w) - |r|(w)| < \epsilon/3$, and we can conclude. □

Proposition 4. *Let A_Z be an enumeration of the set $\{A(N, k, D, n)/N, k, n \in \mathbb{N}, D \in \mathcal{P}(\Sigma^N)\}$. Let $G_0 = r_{|A(N_0,k_0,D_0,f_{n_0})|}(\Sigma^*)$. Let $G_z = r_{|A_z|}(\Sigma^*)$ if it exists, and if $G_z < G_{z-1}$, $G_z = G_{z-1}$ otherwise. Then $\lim_{z\to\infty} G_z = |r|(\Sigma^*)$.*

Proof. Straightforward from Proposition 3. □

Theorem 3. *Let $r \in \mathcal{A}^{rat}(\Sigma)$. For any $\varepsilon > 0$, it is possible to compute an estimate $\widehat{|r|(\Sigma^*)}$ of $|r|(\Sigma^*)$ such that $\left|\widehat{|r|(\Sigma^*)} - |r|(\Sigma^*)\right| < \varepsilon$. If the L_1-distance d of two rational series r and s is finite, for any $\varepsilon > 0$, it is possible to compute an estimate $\hat{d}$ of d such that $|\hat{d} - d| < \varepsilon$.*

Proof. Using Proposition 4, find G_z and n such that $G_z - |r|(\Sigma^{\leq n}) < 2\varepsilon$. For the second part, apply the result to $r - s$. □

6 Conclusion

In this paper, we have proved that it is semi-decidable whether a rational series is absolutely convergent. Then, given an absolutely convergent rational series

r, we have provided an algorithmic way to estimate the sum $|r|(\Sigma^*)$ to any accuracy rate. We do not know whether $\mathcal{A}^{rat}(\Sigma)$ is decidable. We conjecture that it should be decided that a rational series r is not absolutely convergent when $\rho_{|r|} > 1$ but the case $\rho_{|r|} = 1$ is likely to be difficult. We intend to study the links between the spectral radius we have defined and the joint or generalized spectral radii. Finally, we are currently looking for more efficient algorithms and heuristics than those we have described to approximate the L1 distance of two rational series, even if there is no hope to find efficient algorithms in the worst case.

References

1. Berstel, J., Reutenauer, C.: Noncommutative Rational Series With Applications. Cambridge University Press, Cambridge (2008)
2. Salomaa, A., Soittola, M.: Automata: Theoretic Aspects of Formal Power Series. Springer, Heidelberg (1978)
3. Denis, F., Esposito, Y.: On rational stochastic languages. Fundamenta Informaticae 86(1-2), 41–77 (2008)
4. Denis, F., Esposito, Y., Habrard, A.: Learning rational stochastic languages. In: Lugosi, G., Simon, H.U. (eds.) COLT 2006. LNCS, vol. 4005, pp. 274–288. Springer, Heidelberg (2006)
5. Lyngsø, R.B., Pedersen, C.N.S.: The consensus string problem and the complexity of comparing hidden markov models. J. Comput. Syst. Sci. 65(3), 545–569 (2002)
6. Cortes, C., Mohri, M., Rastogi, A.: On the computation of some standard distances between probabilistic automata. In: H. Ibarra, O., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 137–149. Springer, Heidelberg (2006)
7. Cortes, C., Mohri, M., Rastogi, A.: Lp distance and equivalence of probabilistic automata. Int. J. Found. Comput. Sci. 18(4), 761–779 (2007)
8. Theys, J.: Joint Spectral Radius: theory and approximations. PhD thesis, UCL - Université Catholique de Louvain, Louvain-la-Neuve, Belgium (2005)

Non-solvable Groups Are Not in FO+MOD+$\widehat{\text{MAJ}}_2$[REG]

Christoph Behle, Andreas Krebs, and Stephanie Reifferscheid

WSI - University of Tübingen, Sand 13, 72076 Tübingen, Germany
{behlec,krebs,reiffers}@informatik.uni-tuebingen.de

Abstract. Motivated by the open question whether $\mathrm{TC}^0 = \mathrm{NC}^1$ we consider the case of linear size TC^0. We use the connections between circuits, logic, and algebra, in particular the characterization of TC^0 in terms of finitely typed monoids. Applying algebraic methods we show that the word problem for finite non-solvable groups cannot be described by a FO+MOD+MAJ[REG] formula using only two variables. This implies a separation result of FO[REG]-uniform linear TC^0 from linear NC^1.

1 Introduction

An outstanding problem in circuit complexity is that of understanding the relation of the complexity classes ACC^0, TC^0 and NC^1. While the containment $\mathrm{ACC}^0 \subseteq \mathrm{TC}^0 \subseteq \mathrm{NC}^1$ is long known, attempts for separation results have had only limited success. One strategy for attacking these problems is to use the strong connection between circuit classes, logic, and algebra exhibited in [1] and [2]. Since in the general case an answer to these questions seems to be out of reach, the research has concentrated on more restricted settings, which also correspond to meaningful restrictions in logic and algebra. For example restricting the uniformity of circuit classes to a set of predicates yields logic classes with the same set of predicates [1,3], or restricting to linear size circuits corresponds to logic with only two variables [4]. On the algebraic side two-variable logic using only the order predicate corresponds to weakly iterated block products [5,6,7] of suitable monoids. Analogous results in the case of linear size TC^0 , denoted by LTC^0, are shown in [8].

Extending the algebraic characterization of LTC^0 we obtain a class powerful enough to contain FO+MOD+MAJ_2[REG] and FO[REG]-uniform LTC^0, as well as FO[REG]-uniform linear size ACC^0. The main result of this paper is that the word problem of any finite non-solvable group cannot be recognized by this class and is hence not in FO[REG]-uniform LTC^0, while it can be recognized by linear FO[REG]-uniform NC^1. The motivation for such a result is given by the result of Barrington [9] that the word problem of any finite non-solvable group is complete for NC^1, thus a proof that the word problem of a non-solvable group is not in TC^0 would immediately yield a separation result of TC^0 from NC^1.

Limitating TC^0 circuits to linear size is in this case not as severe as it might seem. By the result of Allender and Koucký [10] any superlinear lower bound for the word problem of the S_5 (a non-solvable group), would separate TC^0 from

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 129–140, 2009.

NC^1. Although our motivation is the separation of circuit classes, we choose an algebraic approach, since groups are algebraic objects. This allows us to exploit concepts like commutators and commutativity in our proofs.

2 Preliminaries

As usual $\mathbb{N} = \{1, 2, 3, \ldots\}$ denotes the natural numbers whereas $\mathbb{Z}$ stands for all integers. Further we denote by $\mathbb{Z}^+$, $\mathbb{Z}_0^-$ the partition of $\mathbb{Z}$ into the positive and non positive integers respectively. For a Cartesian product $M = \prod_{i \in I} M_i$ the projection onto the i-th component is denoted by $\pi_i(M)$. Throughout this paper we use Σ for a finite alphabet and a, b for elements of Σ. Moreover, the length of a word w in Σ^* is denoted by $|w|$.

Logic. We give a short introduction following the notation of Straubing [11]. The reader familiar with this topic can skip this part and proceed to the definition of the extended majority quantifier.

First we define the syntactic structure of a formula. A numerical predicate of arity c consists of a symbol P_i^c and a subset of $\mathbb{N}^{c+1}$, the interpretation. To ease notation we will identify a numerical predicate with its symbol and use P_i^c, or just P_i if the arity is clear, when referring to a predicate. Let $\mathcal{V}$ be a set of variables and $x, x_1, \ldots, x_c \in \mathcal{V}$. The atomic formulas consist either of $P_i^c(x_1, \ldots, x_c)$ for a numerical predicate P_i^c or $Q_a(x)$ for $a \in \Sigma$, $x \in \mathcal{V}$. The set of all first order formulas, denoted by FO, is defined recursively as follows: Every atomic formula is a formula and if ϕ and ψ are formulas then $\phi \wedge \psi$, $\phi \vee \psi$, and $\neg\phi$ are formulas. Furthermore for a formula ϕ and a variable x also $\exists x\ \phi$ and $\forall x\ \phi$ are formulas. For the class FO+MOD we admit the modulo quantifier $\mathrm{Mod}_p^r x\ \phi$, $0 \leq r < p$. The notions of bound and free variables are defined as usual and we will write $\phi(x)$ to indicate that x is a free variable occurring in ϕ.

We define the semantics of logic formulas using $\mathcal{V}$-structures. Let $\mathcal{V}$ be a finite set of variables, a $\mathcal{V}$-structure is a string $w = (w_1, \mathcal{V}_1)(w_2, \mathcal{V}_2) \ldots (w_n, \mathcal{V}_n) \in (\Sigma \times 2^{\mathcal{V}})^*$, where the $\mathcal{V}_i, 1 \leq i \leq n$ are pairwise disjoint and $\bigcup_{i=1}^n \mathcal{V}_i = \mathcal{V}$. One can imagine a $\mathcal{V}$-structure as a word with variables pointing to fixed positions. For an alphabet Σ and a set of free variables $\mathcal{V}$ we denote the set of all $\mathcal{V}$-structures by $\Sigma^* \otimes \mathcal{V}$. We use $w_{x=i}$ to denote a $\mathcal{V}$-structure such that $x \in \mathcal{V}_i$.

Let ϕ be a formula and w be a $\mathcal{V}$-structure where $\mathcal{V}$ contains all free variables of ϕ. We define now when w models ϕ, written $w \models \phi$. $w \models Q_a(x)$ iff there exists a letter $(a, \mathcal{V}_j)$ in w with $x \in \mathcal{V}_j$. For $n = |w|$ we set $w \models P_i^c(x_1, \ldots, x_c)$ iff $(n, x_1, \ldots, x_c)$ is contained in the interpretation of P_i^c. Further $w \models \phi \wedge \psi$ ($w \models \phi \vee \psi$) iff $w \models \phi$ and $w \models \psi$ ($w \models \phi$ or $w \models \psi$) and $w \models \neg\phi$ iff w is not a model for ϕ. Finally we define when $w \models \exists x\ \phi$ ($w \models \forall x\ \phi$). If x is not contained in one of the $\mathcal{V}_j$ then $w \models \exists x\ \phi$ ($w \models \forall x\ \phi$) iff there exists some i, $1 \leq i \leq |w|$ (iff for all i, $1 \leq i \leq |w|$) $(w_1, \mathcal{V}_1)(w_2, \mathcal{V}_2) \ldots (w_i, \mathcal{V}_i \cup \{x\}) \ldots (w_n, \mathcal{V}_n) \models \phi$. Else if x is contained in on of the $\mathcal{V}_j$ we remove x from $\mathcal{V}_j$ and proceed as in the case if it is free. Likewise $w \models \mathrm{Mod}_p^r x\ \phi$ iff the number of positions that $w_{x=i} \models \phi$ is congruent to r modulo p.

The Extended Majority Quantifier. In addition to these well known quantifiers we consider the extended majority quantifier $\widehat{\text{Maj}}$ introduced in [8]. The syntax for the extended majority quantifier is as follows: let $\phi_1, \ldots, \phi_c$ be formulas, then $\widehat{\text{Maj}}\ x\langle\phi_1, \ldots, \phi_c\rangle$ is a formula. The semantic meaning of this formula is given by $w \models \widehat{\text{Maj}}\ x\ \langle\phi_1, \ldots, \phi_c\rangle \Leftrightarrow 0 < \sum_{i=1}^{n} \sum_{j=1}^{c} \begin{cases} 1 & \text{if } w_{x=i} \models \phi_j \\ -1 & \text{otherwise} \end{cases}$.

In this paper we use the common numerical predicates: $x < y$ is true if x points to a position before y, the successor predicate $x = y + 1$, also referred as *succ*, is defined by $\{(n, i, i+1) \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \mid i < n\}$. The modulo predicate $mod_p(x)$ is defined by the set $\{(n, i) \subseteq \mathbb{N} \times \mathbb{N} \mid i \leq n \text{ and } i \equiv 0 \mod p\}$ and $last(x)$ by $\{(n, n) \mid n \in \mathbb{N}\}$. A c-ary numerical predicate P is called regular iff the language $L_P = \{w \in \{a\}^* \otimes \{x_1, \ldots x_c\} \mid w \models P(x_1, \ldots x_c)\}$ is regular. We denote by REG the set of regular predicates.

A numerical predicate P is said to be first order constructible from a set of numerical predicates $\mathfrak{P}$ if there exists a FO$[<, \mathfrak{P}]$ formula ϕ such that for all w, $w \models P$ iff $w \models \phi$. Furthermore for two c-ary numerical predicates P, P' we call P a shifting predicate of P' if there exist integers $v_1, \ldots, v_{c+1}$ such that $P = \{(i_1 + v_1, \ldots, i_{c+1} + v_{c+1}) \mid (i_1, \ldots, i_{c+1}) \in P'\}$.

We use FO+MOD+$\widehat{\text{MAJ}}_2$ to denote the set of formulas which can be written with two variables (which can be used more than once) using first order, modulo, and extended majority quantifiers. For a set of numerical predicates $\mathfrak{P}$ we use FO+MOD+$\widehat{\text{MAJ}}_2[\mathfrak{P}]$ to denote all formulas in FO+MOD+$\widehat{\text{MAJ}}_2$ using only numerical predicates in $\mathfrak{P}$. We will use this notations analogously for FO and FO+MOD. As usual, for a logic class C we denote by $\mathcal{L}(C)$ the class of languages describable in C.

In [8] the connections between logic, algebra, and circuits in the case of two variables with special regards to the extended majority quantifier have been studied. To apply these results for our setting we need the following lemma.

Lemma 1. *$\mathcal{L}(FO+MOD+\widehat{MAJ}_2[REG]) = \mathcal{L}(FO+MOD+\widehat{MAJ}_2[<, succ])$.*

It is also true that $\text{FO}_2[\text{REG}] = \text{FO}_2[<, succ, mod]$ and $\text{FO+MOD}_2[\text{REG}] = \text{FO+MOD}_2[<, succ]$.

Monoids. The reader is assumed to be familiar with the basic algebraic concepts as they are presented for instance in the first two chapters of Pin [12] or in [11], Chapter 5. For an overview about the connections between two-variable logic and algebra we refer the reader to [6]. In particular, we denote for a monoid M by e the neutral element and we write e_M to avoid ambiguity if necessary. An important monoid is the monoid U_1 with the elements $\{0, 1\}$ and multiplication given by $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$ and $1 \cdot 1 = 1$.

Definition 1 (Word problem). *For a finite group G the word problem $L_G \subseteq G^*$ is the set of all words $g_1 \ldots g_n$ such that $g_1 \cdot \ldots \cdot g_n = e$ where e is the neutral element in G.*

Given a language $L \subseteq \Sigma^*$ we call $e \in \Sigma$ a neutral letter (for L) if for all words $u, v \in \Sigma^*$, $uv \in L$ iff $uev \in L$. The word problem always has a neutral letter, namely the letter corresponding to the neutral element e_G.

Our separation results depends on showing that groups with a certain property cannot be recognized by certain monoids. For this we need the concept of a commutator. For a group G and $g, h \in G$ we denote by $[g,h] := g^{-1}h^{-1}gh$ the commutator of g and h. The group generated by all commutators of G is called the commutator subgroup G' of G. A non-trivial group G is called perfect if $G = G'$.

It is a well known fact that a group is non-solvable iff it possesses a perfect subgroup. Recall that A_5, the alternating group on five elements, is a perfect group. As usual we denote by $\overline{G}_{solv}$ the class of all finite monoids M such that every group contained in M is solvable.

3 Semilinear Sets

Semilinear sets play a significant role in our work. We recall here the definition and refer the reader to [13] for more details.

Definition 2 (Semilinear Sets). *Let $d \in \mathbb{N}$. A set $S \subseteq \mathbb{Z}^d$ is linear iff there are $x, y_1, \ldots, y_c \in \mathbb{Z}^d$, such that $S = \{x + \sum_{i=1}^{c} k_i y_i \mid k_1, \ldots, k_c \in \mathbb{N} \cup \{0\}\}$. In the following we denote by $\langle x, y_1, \ldots, y_c \rangle$ the set S. A set is semilinear iff it is a finite union of linear sets.*

The following two lemmas are simple observations and can be proved using combinatorics.

Lemma 2. *Let $s_1, t_1, u_1 \in \mathbb{N}$ and $N = \operatorname{lcm}\{t_1, u_1\}$. If there are $l_1, l_2 \in \mathbb{N}$ such that $s_1 + l_1 t_1 + l_2 u_1 \equiv 0 \mod N$, then there are unique minimal $k_1, k_2 \in \mathbb{N}$ such that $s_1 + k_1 t_1 + k_2 u_1 \equiv 0 \mod N$, and $(N/t_1)|(l_1 - k_1)$ and $(N/u_1)|(l_2 - k_2)$.*

Lemma 3. *Let $S = \langle s, t, u \rangle \subseteq \mathbb{N} \times \mathbb{N}$ be a semilinear set, then there are numbers $N, \gamma \in \mathbb{N}$ and $(p_1, q_1), (p_2, q_2) \in [0,1]_{\mathbb{Q}} \times \mathbb{Z}$, where $[0,1]_{\mathbb{Q}}$ is the rational unit interval, such that for all $n_1 \equiv n_2 \equiv 0 \mod N$, and all $i_1 \in \{1, \ldots, n_1\}$ and $i_2 \in \{1, \ldots, n_2\}$: If $\{j \mid p_j \cdot n_1 + q_j \leq i_1\} = \{j \mid p_j \cdot n_2 + q_j \leq i_2\}$ and $i_1 \equiv i_2 \mod \gamma$ then $(n_1, i_1) \in S \Leftrightarrow (n_2, i_2) \in S$.*

The following proposition shows that unary semilinear sets can be split in a finite partition such that every partition entry behaves like a mod predicate.

Proposition 1 (Semilinear Proposition). *Let $S \subseteq \mathbb{N} \times \mathbb{N}$ be a semilinear set. Then there are numbers $N, \gamma \in \mathbb{N}$, and $(p_1, q_1), \ldots, (p_c, q_c) \in [0,1]_{\mathbb{Q}} \times \mathbb{Z}$, such that for all $n_1 \equiv n_2 \equiv 0 \mod N$, and all $i_1 \in \{1, \ldots, n_1\}$ and $i_2 \in \{1, \ldots, n_2\}$ holds: If $\{j \mid p_j \cdot n_1 + q_j \leq i_1\} = \{j \mid p_j \cdot n_2 + q_j \leq i_2\}$ and $i_1 \equiv i_2 \mod \gamma$ then $(n_1, i_1) \in S \Leftrightarrow (n_2, i_2) \in S$.*

Semilinear sets are in our setting of interest since they are connected to numerical predicates. We call a numerical predicate semilinear if its associated subset is semilinear. It has been shown in [14] that MAJ[<] can only express semilinear predicates and for unary predicates we can now complement this result for FO+MOD+$\widehat{\text{MAJ}}_2$[REG]. We summarize these results in the following proposition:

Proposition 2. *The unary predicates that can be expressed by FO+MOD+* $\widehat{MAJ}_2[REG]$ *are exactly the unary semilinear predicates. Moreover all unary semilinear predicates can be expressed by a FO+MOD+*$\widehat{MAJ}_2[<, last]$ *formula of quantifier depth 1.*

4 Finitely Typed Monoids

In this section we introduce the algebraic counterpart of the class FO+MOD+ $\widehat{MAJ}_2[REG]$, namely the class **W** consisting of certain so called finitely typed monoids introduced in [2]. To obtain the characterization for the two variable case we need to restrict the recognizing morphisms as in [8] (see below for the exact definition). At the end of this section we prove the so called Commutator Proposition stating that in our setting we may assume that the recognizing morphism has a special form; here the non-solvability of the group is exploited.

Before giving the formal definition of a finitely typed monoid, we consider the following example. Let $\Sigma = \{a, b\}$ and $L = \{w \mid |w|_a > |w|_b\}$ the set of words where the majority of letters are a's. Define a morphism into the additive group of integers $h : \Sigma^* \to \mathbb{Z}$ by $h(a) = 1$ and $h(b) = -1$. Then obviously $w \in L$ iff $h(w) > 0$. Thus when recognizing L via the given morphism, the exact value of $h(w)$ is irrelevant, the important information however is whether $h(w)$ is positive or not. So, if we equip $\mathbb{Z}$ with the "types" $\mathbb{Z}^+$ and $\mathbb{Z}_0^-$, then L is recognized by $(\mathbb{Z}, \{\mathbb{Z}^+, \mathbb{Z}_0^-\})$ via h in the sense that $L = h^{-1}(\mathbb{Z}^+)$. A generalization of this idea leads to the definition of finitely typed monoids and recognizability of languages by these.

We call a monoid T *finitely typed with type set* $\mathfrak{T} = \{\mathcal{T}_i \mid i \in I\}$ iff $T = \dot{\bigcup}_{i\in I}\mathcal{T}_i$ for a finite set I. The pairwise disjoint sets $\mathcal{T}_i$, $i \in I$, are called the *types* of T. We call the elements of $\mathcal{B}(\mathfrak{T})$, (the Boolean algebra generated by $\mathfrak{T}$), *extended types* of T. If the type set $\mathfrak{T}$ of T is understood we often simply write T instead of $(T, \mathfrak{T})$. Note that a finite monoid T can be regarded as a finitely typed monoid equipped with the discrete typeset $\{\{t\} \mid t \in T\}$. The *direct product* $(S, \mathfrak{S}) \times (T, \mathfrak{T})$ of two finitely typed monoids $(S, \mathfrak{S})$ and $(T, \mathfrak{T})$ is the usual Cartesian product equipped with the type set $\mathfrak{S} \times \mathfrak{T} = \{\mathcal{S} \times \mathcal{T} \mid \mathcal{S} \in \mathfrak{S},\ \mathcal{T} \in \mathfrak{T}\}$.

An important concept in the structure theory of finite monoids is the block product introduced in [15]. Since the usual concept for finite monoids is too powerful for our purposes we work with a restricted version of the block product. Let $(S, \mathfrak{S})$, $(T, \mathfrak{T})$ be finitely typed monoids. We call a function $f : (T, \mathfrak{T}) \times (T, \mathfrak{T}) \to S$ *strongly type respecting* if there are elements u, v such that the value of the function at (t_1, t_2) depends only on the type of $(T, \mathfrak{T}) \times (T, \mathfrak{T})$ to which (t_1u, vt_2) belongs. We call f *type dependent* if there is a constant c such that the value of the function at (t_1, t_2) depends only on the type the element t_1ct_2 belongs to. Finally, we call f *type respecting* (with respect to T and S) if it can be written as finite product of strongly type respecting and type dependent functions (as usual the product of two functions is the product of their values). Now we are able to define the block product:

Definition 3 (Block Product). *Let* $(S, \mathfrak{S})$, $(T, \mathfrak{T})$ *be finitely typed monoids and let* V *be the set of all type respecting functions (with respect to* T *and* S*). The* finitely typed block product $(X, \mathfrak{X}) = (S, \mathfrak{S}) \boxdot (T, \mathfrak{T})$ *of* $(S, \mathfrak{S})$ *with* $(T, \mathfrak{T})$ *is defined as the bilateral semidirect product* $V * * T$ *of* V *with* T *where the right (respectively left) action of* T *on* V *is given by* $(f \cdot t)(t_1, t_2) = f(t_1, tt_2)$ *(respectively* $(t \cdot f)(t_1, t_2) = f(t_1 t, t_2)$*),* $t,\ t_1,\ t_2 \in T,\ f \in V$*. The type set* $\mathfrak{X}$ *of* X *consists of all types* $\mathcal{X}_{\mathcal{S}} = \{(f, t) \in X \mid f(e_T, e_T) \in \mathcal{S}\}$*, where* $\mathcal{S} \in \mathfrak{S}$*.*

We say that a finitely typed monoid $(T, \mathfrak{T})$ recognizes the language $L \subseteq \Sigma^*$ if there is a morphism $h : \Sigma^* \to T$ and an extended type $\mathcal{T}$ of T such that $L = h^{-1}(\mathcal{T})$. Note that the only difference to the usual notion of language recognition by a monoid is the limitation of the allowed accepting sets.

The algebraic class considered in this paper is defined by applying the direct product and the block product to a set of special finitely typed semigroups.

Definition 4 ($\mathfrak{SL}$). *Let* $\mathfrak{SL}$ *be the set of all finite partitions of* $\mathbb{Z}$ *such that all elements of the partition are semilinear sets. Let* $(\mathbb{Z}, \mathfrak{T})$ *be a finitely typed monoid. A type set* $\mathfrak{T}$ *is called semilinear iff* $\mathfrak{T} \in \mathfrak{SL}$*. In the following we will write* $(\mathbb{Z}, \mathfrak{Sl})$ *to denote a finitely typed monoid equipped with a semilinear typeset.*

In [8], for a binary predicate P a so called predicate monoid was introduced, that is a finitely typed monoid $(T, \mathfrak{T})$ with distinguished element t, the incremental element, such that there is a morphism $h : (\{a\} \times 2^{\{x,y\}})^* \to T,\ h(a, \emptyset) = t$ such that $a^n_{x=i,y=j} \models P(x, y)$ iff $h(a^n_{x=i,y=j}) \in \mathcal{T}$, and moreover, every predicate being recognized in this sense by such morphism can be built by P via FO[<]-constructions and shifting.

Lemma 4. *Set* $(T_<, \mathcal{T}_<) = ((U_1, \{0, 1\}) \boxdot (\mathbb{Z}, \{\mathbb{Z}^+, \mathbb{Z}_0^-\}))$ *and* $t_< = (f_<, 0)$*, where* $f_< : \mathbb{Z} \times \mathbb{Z} \to U_1,\ f_<(z_1, z_2) = 1_{U_1}$ *if* $z_1 \in \mathbb{Z}_0^-$ *and* $f_<(z_1, z_2) = 0_{U_1}$ *else. Then* $((T_<, \mathfrak{T}_<), t_<)$ *is a predicate monoid for* $<$ *and a predicate monoid for successor.*

In the following we use $T_<$ for $((T_<, \mathfrak{T}_<), t_<)$ in Lemma 4.

We now proceed as in [8], and construct a class of finitely typed monoids that corresponds to our logic. As in this paper we need to restrict the choice of the recognizing morphisms in order to distinguish between numerical predicates and quantifiers in the algebra.

Definition 5 (W). *Let* $\mathbf{W} = \bigcup_{d \geq 0} \mathbf{W}_d$*, where*

$\mathbf{W}_0 = \{(\bigtimes_{j=1}^{i} (\mathbb{Z}, \mathfrak{Sl}_j)) \boxdot T_<^k \mid i \in \mathbb{N}, \forall j \leq i : \mathfrak{Sl}_j \in \mathfrak{SL}, k \in \mathbb{N}\}$ *and*

$\mathbf{W}_{d+1} = \{T' \boxdot ((\bigtimes_{j=1}^{i} (\mathbb{Z}, \mathfrak{Sl}_j)) \boxdot T_<^k) \mid T' \in \mathbf{W}_d, i \in \mathbb{N}, \forall j \leq i : \mathfrak{Sl}_j \in \mathfrak{SL}, k \in \mathbb{N}\}$.

As in [8] the construction of the class of monoids is not enough to exactly describe a set of languages, we also need to define which are the restricted elements in the monoids of our class. We do this by induction over the structure of monoids in $\mathbf{W}$.

Definition 6 (Restricted Element, Restricted Morphism). *We define inductively the set of restricted elements of monoids in* $\mathbf{W}$*:*

1. *All elements of* $(\mathbb{Z}, \mathfrak{Sl})$ *are restricted, with* $\mathfrak{Sl} \in \mathfrak{SL}$.
2. *For* $((T_<, \mathfrak{T}_<), t_<)$ *only the incremental element* $t_<$ *is restricted.*
3. *An element* $x \in A \times B$ *is restricted iff* $\pi_1(x)$ *and* $\pi_2(x)$ *are restricted.*
4. *An element* $x \in A \boxdot B$ *is restricted iff all elements in the image of* $\pi_1(x)$ *are restricted and* $\pi_2(x)$ *is restricted.*

A morphism $h : \Sigma^* \to T \in \mathbf{W}$ *is restricted iff* $h(s)$ *is restricted for all* $s \in \Sigma$.

Note that if $h : \Sigma^* \to T \in \mathbf{W}$ is a restricted morphism then for any morphism $\alpha : \Sigma'^* \to \Sigma^*$, $h \circ \alpha$ is also a restricted morphism. This is due to the fact that the product of two restricted elements is restricted again, which follows since $t_<$ is idempotent.

Definition 7 ($\mathcal{H}_R^{-1}(\mathbf{W})$)**.** *A language* $L \subseteq \Sigma^*$ *is in* $\mathcal{H}_R^{-1}(\mathbf{W})$ *iff there is a monoid* $(T, \mathfrak{T}) \in \mathbf{W}$ *and a restricted morphism* $h : \Sigma^* \to (T, \mathfrak{T})$ *such that* $L = h^{-1}(\mathcal{T})$ *for some extended type* $\mathcal{T}$ *of* T. *In this case we say* L *is restrictively recognized by* T.

The following lemma shows that the recognizing power of finite direct products of monoids in $\mathbf{W}$ is absorbed by a monoid in $\mathbf{W}$ of higher block depth. Thus the reader familiar with varieties may notice that the languages being recognized by $\mathbf{W}$ are exactly the languages being recognized by the variety generated by $\mathbf{W}$.

Lemma 5. *Let* $h : \Sigma^* \to T = (A \boxdot B) \times (C \boxdot D)$ *be a (restricted) morphism. Then there is a (restricted) morphism* $h' : \Sigma^* \to T' = (A \times C) \boxdot (B \times D)$, *such that for every type* $\mathcal{T}$ *of* T *there is a type* $\mathcal{T}'$ *of* T' *with* $h^{-1}(\mathcal{T}) = h'^{-1}(\mathcal{T}')$.

With the lemma above and Lemma 1 we can now use similar techniques as in [8] to prove the following proposition.

Proposition 3. $\mathcal{L}(FO{+}MOD{+}\widehat{MA}J_2[REG]) = \mathcal{H}_R^{-1}(\mathbf{W})$.

We will exploit the algebraic characterization to prove our results.

Lemma 6 (Commutator Lemma). *Let* G *be a finite group and let* L_G *be recognized by a finitely typed monoid* T *and a restricted morphism* h. *Further let* K *be a commutative monoid and* $k : h(G^*) \to K$ *be a monoid morphism. Then* $L_{G'}$ *can be recognized by* $h' : (G')^* \to T$ *such that* $|k(h'(G'))| = 1$, *where* G' *denotes the commutator subgroup of* G.

Applying Lemma 6 to our situation gives

Proposition 4 (Commutator Proposition). *If* L_G *is recognized by a (restricted) morphism* $h : G^* \to T = T' \boxdot (\bigtimes_{j=1}^{i}(\mathbb{Z}, \mathfrak{Sl}_j) \boxdot T_<^k) \in \mathbf{W}$ *for a finite group* G. *Then there is a (restricted) morphism* $h' : (G')^* \to T$, *recognizing the language* $L_{G'}$, *such that* $|\pi_2(h'(G'))| = 1$.

By Proposition 4 holds in particular, that for perfect groups G, thus $L_G = L_{G'}$, we may assume that there is a fixed element $m \in (\bigtimes_{j=1}^{i}(\mathbb{Z}, \mathfrak{Sl}_j) \boxdot T_<^k)$ such that for all $s \in \Sigma$ we have $\pi_2(h(s)) = m$.

5 Prefix/Suffix Mappings

In circuit theory there is the notion of circuits that recognize languages with advice. In this section we define what it means that a language is recognized by a morphism and given advice depending only on the length of the word. The advice is realized via so called prefix/suffix functions. The main step of the proof of Theorem 1 is that under certain circumstances we can reduce the block depth of the recognizing finitely typed monoid by extending the advice in the morphism; this is proved at the end of this section.

Definition 8. *A function $\nu : \mathbb{N} \to \Sigma^*$ is a semilinear prefix/suffix function iff $\forall n : |\nu(n)| = n$ and for each $s \in \Sigma$, the set $\chi_s = \{(n,i) \in \mathbb{Z}^2 \mid 1 \leq i \leq n \text{ and } \nu(n)_i = s\}$ is semilinear. The pair $(\nu, h) : \mathbb{N} \to T$, where ν is a semilinear prefix/suffix function, T is a monoid and $h : \Sigma^* \to T$ is a monoid morphism, is called a semilinear prefix/suffix mapping. Two semilinear prefix/suffix mappings (ν_1, h_1) and (ν_2, h_2) are equivalent iff $h_1(\nu_1(n)) = h_2(\nu_2(n))$ for all $n \in \mathbb{N}$.*

It is not hard to see that prefix/suffix mappings are stable under the following operations.

Lemma 7 (Concatenation). *Let (ν, h), (ν_1, h_1) and (ν_2, h_2) be semilinear prefix/suffix mappings into the finitely typed monoid T and let $c \in \mathbb{N}$ and $r, l \in T$.*

(a) There is a prefix/suffix mapping ν', h' such that $\forall n : h(\nu(cn)) = h'(\nu'(n))$.
(b) There is a semilinear prefix/suffix mapping (ν, h) such that $\forall n : h(\nu(n)) = h_1(\nu_1(n)) \cdot h_2(\nu_2(n))$.
(c) There is a semilinear prefix/suffix mapping (ν', h') such that $\forall n : h'(\nu'(n)) = r \cdot h(\nu(n)) \cdot l$.

We will see in Lemma 9 that prefix/suffix functions in some sense also behave well with respect to the block product.

Now we define the extended notion of accepting languages.

Definition 9. *Let Σ be a finite alphabet, T be a finitely typed monoid, $\mathcal{T}$ be an extended type of T, and $h : \Sigma^* \to T$ be a morphism. Further let (λ, h_λ), (ρ, h_ρ) be semilinear prefix/suffix mappings, with $\lambda : \mathbb{N} \to \Sigma_\lambda^*$, $\rho : \mathbb{N} \to \Sigma_\rho^*$, where $\Sigma_\lambda, \Sigma_\rho$ are possibly different alphabets than Σ, and let $h_\lambda : \Sigma_\lambda^* \to T$, $h_\rho : \Sigma_\rho^* \to T$ be morphisms. A language $L \subseteq \Sigma^*$ is recognized by $((h, T, \mathcal{T}), (\lambda, h_\lambda), (\rho, h_\rho))$ iff for all $w \in \Sigma^*$ holds: $w \in L \Leftrightarrow h_\lambda(\lambda(|w|))h(w)h_\rho(\rho(|w|)) \in \mathcal{T}$. We say L is restrictively recognized by $((h, T, \mathcal{T}), (\lambda, h_\lambda), (\rho, h_\rho))$ iff the morphism h is a restricted morphism.*

Note that if a language $L \subseteq \Sigma^*$ can be recognized by a morphism $h : \Sigma^* \to T$, then it can be recognized by $((h, T, \mathcal{T}), (\lambda, h_\lambda), (\rho, h_\rho))$ for suitable prefix/suffix mappings. Note further that Proposition 4 can be applied in the presence of prefix/suffix functions: With notations as in the proof of Lemma 6, let $c = |w_g|$, for $g \in G'$ then the assertion follows by Lemma 7 and Proposition 4.

The following lemma together with Proposition 2 shows that the class **W** can exactly recognize the unary semilinear predicates.

Lemma 8. *Let $L \subseteq \{a\}^* \otimes \{x\}$ be a language recognized by the triple $((h,T,\mathcal{T}),(\lambda,h_\lambda),(\rho,h_\rho))$, where $T \in \mathbf{W}$, then for all n the set $\{(n,i) \mid a^n_{x=i} \in L\}$ is a semilinear set.*

In an analogous manner, we can prove that prefix/suffix mappings behave nicely with respect to the block product, which is needed in the following proposition.

Lemma 9. *Let $(\nu,h) : \mathbb{N} \to T = T' \boxdot ((\bigtimes^i_{j=1}(\mathbb{Z},\mathfrak{Sl}_j)) \boxdot T^k_<) \in \mathbf{W}$ and $(\nu_c,h_c),(\nu_d,h_d) : \mathbb{N} \to (\bigtimes^i_{j=1}(\mathbb{Z},\mathfrak{Sl}_j)) \boxdot T^k_<$ be prefix/suffix mappings. Then there is a prefix/suffix mapping $(\nu',h') : \mathbb{N} \to T'$ such that $\forall n : \pi_1(h(\nu(n)))(h_c(\nu_c(n),h_d(\nu_d(n)))) = h'(\nu'(n)))$.*

Now we can prove that we can reduce the block depth given that the last component of the morphism is equal for each generator. Note that in case $L = L_G$ for a perfect group G this assumption is given by Proposition 6.

Proposition 5 (Reduction Proposition). *Let $L \subseteq \Sigma^*$ be restrictively recognized by $((h,T,\mathcal{T}),(\lambda,h_\lambda),(\rho,h_\rho))$, where $T = T' \boxdot ((\bigtimes^i_{j=1}(\mathbb{Z},\mathfrak{Sl}_j)) \boxdot T^k_<) \in \mathbf{W}$ for some $i \in \mathbb{N}$. Assume further the existence of a neutral letter $e \in \Sigma$ (for L) and that $|\pi_2(h(\Sigma))| = 1$. Then there is $((h',T',\mathcal{T}'),(\lambda',h_{\lambda'}),(\rho',h_{\rho'}))$, that recognizes L restrictively with T' as above.*

Proof. The proof of Proposition 5 is done in three steps. To state them we use the following notation: We set $M = (\bigtimes^i_{j=1}(\mathbb{Z},\mathfrak{Sl}_j)) \boxdot T^k_<$ and $h(s) = (f_s,m)$ for all $s \in \Sigma$, where $m = ((g_1,b),\ldots,(g_i,b)) \in M$ with $b = (t_<,\ldots,t_<)$. Further set $W_s = Im(f_s) \subseteq T'$ for $s \in \Sigma$ and $h_\lambda(\lambda(n)) = (f_{\lambda(n)},m_{\lambda(n)})$, $h_\rho(\rho(n)) = (f_{\rho(n)},m_{\rho(n)})$.

Step I: *We show that we can find a natural number γ, such that for any word $w = w_1 \ldots w_n$ of length n with $n \equiv 0 \mod \gamma$ there is an interval such that at position i in that interval the corresponding value of f_{w_i} depends only on the position modulo γ and the letter w_i but not on the letters in the prefix and suffix; that is: we have an interval $[l(n),r(n)]$ such that for $l(n) \le i,j \le r(n)$ holds: if $i \equiv j \mod \gamma$ and $w_i = w_j$, then $f_{w_i}(m_{\lambda(n)}m^{i-1},m^{n-i}m_{\rho(n)}) = f_{w_j}(m_{\lambda(n)}m^{j-1},m^{n-j}m_{\rho(n)})$.*

Let $s \in \Sigma$ and $t \in W_s$. Since $|\pi_2(h(\Sigma))| = 1$ by Lemma 8 the set $S_{s,t} = \{(n,i) \in \mathbb{N} \times \mathbb{N} \mid i \le n\},\ f_s(m_{\lambda(n)}m^{i-1},m^{n-i}m_{\rho(n)}) = t\}$ is semilinear. Thus Proposition 1 gives a finite partition of $\{(n,i) \in \mathbb{N} \times \mathbb{N} \mid i \le n\}$, given by lines $(p_{s,t,1},q_{s,t,1}),\ldots,(p_{s,t,c},q_{s,t,c})$, a cycle length $\gamma_{s,t}$ and a natural number $N_{s,t}$ such that for suitable numbers n_1,n_2 holds: If two tuples (n_1,i_1) and (n_2,i_2) are in the same partition and the i's have the same value modulo $\gamma_{s,t}$ then both tuples are in $S_{s,t}$ or none. Σ is finite and so is W_s for every s, thus we can choose a refinement of the partitions given as above. We pick a linear growing element of this partition and call the corresponding bounding lines (p_1,q_1), (p_2,q_2). We assume without loss in generality that $q_1 = q_2 = 0$ (otherwise we reduce this partition element in a suitable way). Further we set $\gamma = \text{lcm}(\gamma_{s,t},N_{s,t} \mid s \in \Sigma,\ t \in W_s)$. Thus we have the assertion with $l(n) = np_1$ and $r(n) = np_2$. In the following assume that $np_1,np_2 \in \mathbb{N}$ (otherwise pick a suitable multiple of γ).

Step II: *We show, that we can find semilinear prefix/suffix mappings* $(\tilde{\lambda}, h_{\tilde{\lambda}})$, $(\tilde{\rho}, h_{\tilde{\rho}})$ *and a restricted morphism* $\tilde{h} : \Sigma^* \to T$ *such that the triple* $((\tilde{h}, T, \mathcal{T}), (\tilde{\lambda}, h_{\tilde{\lambda}}), (\tilde{\rho}, h_{\tilde{\rho}}))$ *restrictively recognizes* L *and for all* $s \in \Sigma$ *the function* $\pi_1(\tilde{h}(s)) : M \times M \to T'$ *is constant.*

The idea is to extend each word $w \in \Sigma^*$ to a word $\tilde{w} \in \Sigma^*$ by adding suitable powers of the neutral word e (thus the new word is in L iff the old one is). To do this we need the notation of Step I; further we set $p_2 - p_1 = \frac{c_1}{c_2}$ with natural numbers $c_1, c_2, 0 < c_1 < c_2$ (thus we assume without loss in generality that $p_2 - p_1 \in]0, 1[_{\mathbb{Q}}$). First we extend $w = w_1 \dots w_n$ to $\overline{w} = w_1 e^{c_1\gamma - 1} \dots w_n e^{c_1\gamma - 1}$, which ensures that every letter of $\Sigma \setminus \{e\}$ has the same position modulo γ. Since we want the whole word $\overline{w}$ to be located in the above described interval of an adequate bigger word , we add suitable powers of the neutral word to both sides of $\overline{w}$. For this we set $\tilde{n} = n\gamma c_2$ and $\tilde{w} = e^{l(\tilde{n})}\overline{w}e^{\tilde{n} - r(\tilde{n})}$. Note that $|\tilde{w}| = \tilde{n} \equiv 0 \mod \gamma$ and denote the letters of $\tilde{w}$ by $\tilde{w}_i$, that is $\tilde{w} = \tilde{w}_1, \dots, \tilde{w}_{\tilde{n}}$ with $\tilde{w}_i \in \Sigma$. Since e is a neutral letter we have $w \in L$ iff $\tilde{w} \in L$.

We define the morphism $\overline{h} : \Sigma^* \to T$ by $\overline{h}(s) = h(se^{\gamma c_1 - 1})$. Since by Lemma 7 (a),(b) there are semilinear prefix/suffix mappings $(\tilde{\lambda}, h_{\tilde{\lambda}})$, $(\tilde{\rho}, h_{\tilde{\rho}})$ such that $h_\lambda(\lambda(\tilde{n}))h(e)^{l(\tilde{n})} = h_{\tilde{\lambda}}(\tilde{\lambda}(n))$ and $h(e)^{\tilde{n} - r(\tilde{n})h_\rho(\rho(\tilde{n}))} = h_{\tilde{\rho}}(\tilde{\rho}(n))$ we have

$$w \in L \Leftrightarrow h_\lambda(\lambda(|\tilde{w}|))h(\tilde{w})h_\rho(\rho(|\tilde{w}|)) \in \mathcal{T} \Leftrightarrow h_{\tilde{\lambda}}(\tilde{\lambda}(|w|))\overline{h}(w)h_{\tilde{\rho}}(\tilde{\rho}(|w|)) \in \mathcal{T}.$$

Step I gives for $l(\tilde{n}) \le i, j \le r(\tilde{n})$: If $i, j \equiv k \mod \gamma$ and $\tilde{w}_i = \tilde{w}_j = s$ then $f_{\tilde{w}_i}(m_{\lambda(\tilde{n})}m^{i-1}, m^{\tilde{n}-i}m_{\rho(\tilde{n})}) = f_{\tilde{w}_j}(m_{\lambda(\tilde{n})}m^{j-1}, m^{\tilde{n}-j}m_{\rho(\tilde{n})})$. We denote this value by $\tilde{t}_s^{(k)})$) where $0 \le k \le \gamma - 1$. Thus for $\tilde{h} : \Sigma^* \to T$ defined by $\tilde{h}(s) = (\tilde{f}_s, m^{\gamma c_1})$ with $\tilde{f}_s : M \times M \to T', (m_1, m_2) \mapsto t_s := \tilde{t}_s^{(0)}\tilde{t}_e^{(1)} \dots \tilde{t}_e^{(\gamma c_1 - 1)}$ for all $(m_1, m_2) \in M \times M$ we have $w \in L$ iff $h_{\tilde{\lambda}}(\tilde{\lambda}(|w|))\tilde{h}(w)h_{\tilde{\rho}}(\tilde{\rho}(|w|)) \in \mathcal{T}$ and hence the assertion.

Step III: *The definition of* $((h', T', \mathcal{T}'), (\lambda', h_{\lambda'}), (\rho', h_{\rho'}))$.
We define $h' : \Sigma^* \to T'$ by setting $h'(s) = t_s$ for $s \in \Sigma$ (notation as in Step II). To ease notation set $\tilde{\lambda}_w = h_{\tilde{\lambda}}(\tilde{\lambda}(|w|))$ and $\tilde{\rho}_w = h_{\tilde{\rho}}(\tilde{\rho}(|w|))$. Then by Step II $w \in L \Leftrightarrow \pi_1(\tilde{\lambda}_w\tilde{h}(w)\tilde{\rho}_w)(e, e) \in \mathcal{T}'$. By the definition of the block product and h' this is equivalent to $\pi_1(\tilde{\lambda}_w)(e_M, m^{n\gamma c_1}\pi_2(\tilde{\rho}_w))h'(w)\pi_1(\tilde{\rho}_w)(\pi_2(\tilde{\lambda}_w))m^{n\gamma c_1}, e_M) \in \mathcal{T}'$ and by Lemma 9 there exist prefix/suffix mappings $(\lambda', h_{\lambda'})$ and $(\rho', h_{\rho'})$ such that the latter holds iff $h_{\lambda'}(\lambda'(|w|)))h'(w)h_{\rho'}(\rho'(|w|)) \in \mathcal{T}'$. Hence the proof is completed.

6 Results

We obtain now our separation results for logic and circuits by proving the theorem stating no finite non-solvable monoid can be recognized by $\mathbf{W}$. The main idea for the latter is the following: Every finite non-solvable monoid contains a perfect group. For this we know by Proposition 4 (Commutator Proposition) that the recognizing morphism is of a special form, thus we can use Proposition 5 (Reduction Proposition) to reduce the block depth of the recognizing monoid.

The implication in terms of logic follows from Proposition 3, and yields by [8] also separation results for circuit classes.

Theorem 1. *Every regular language* $L \in \mathcal{H}_R^{-1}(\mathbf{W})$ *has a syntactic monoid in* $\overline{G}_{solv}$.

Since by Proposition 3 the regular languages in $\mathcal{H}_R^{-1}(\mathbf{W})$ are exactly the regular languages in FO+MOD+$\widehat{\text{MAJ}}_2$[REG] we obtain:

Theorem 2. *Every regular language in* $\mathcal{L}(FO{+}MOD{+}\widehat{MA}J_2[REG])$ *has a syntactic monoid in* $\overline{G}_{solv}$.

Applying the main theorems of [8] we obtain the statement in terms of circuit theory which is a separation of the following circuit classes from uniform linear NC^1, since any regular language can be recognized by a linear circuit in NC^1.

Theorem 3. *Every regular language in* $\mathcal{L}(FO[REG] - \text{uniform } LTC^0)$ *has a syntactic monoid in* $\overline{G}_{solv}$.

Since the arguments in [8] can be transferred for the case of first order and modulo logic we obtain a known statement for linear ACC^0 (shortly LCC^0):

Theorem 4. *Every regular language in* $\mathcal{L}(FO[REG] - \text{uniform } LCC^0)$ *has a syntactic monoid in* $\overline{G}_{solv}$.

7 Discussion

We have shown that the only groups being recognized by $\mathbf{W}$ are solvable. Although our motivation is the separation of circuit classes, we choose the algebraic approach, since by Barrington's result [9] any finite non-solvable group is a natural candidate to show a separation, and groups are algebraic objects. Further, concepts like commutators and commutativity are basic concepts in algebra.

Besides our main result we show that all binary semilinear predicates can be expressed in FO + MOD + $\widehat{\text{MAJ}}_2$[REG] yielding the bound opposite to the one in [14]. It is also interesting that similar to Lemma 1 we have that all regular numerical predicates can be expressed by $\text{FO}_2[<, succ, mod]$ or $\text{FO+MOD}_2[<, succ]$, thus the difference in predicates between two variables and unbounded variables is exactly the successor predicate.

As mentioned before prefix/suffix functions correspond to advice in circuit theory. Likewise we could have defined prefix/suffix functions of polynomial size and used the same proofs but this would not have led to a more general result.

From the logical point of view there are two options for further research, first extending the set of predicates, second extending the number of variables. The second option seems to be more compelling towards a separation of TC^0 from NC^1, but since the algebra would change drastically new approaches would be needed.

Interestingly the latter extension can be also achieved by extending the set of predicates. Assume there exists a FO[REG]-uniform polynomial size TC^0 circuit recognizing L_{A_5}, then we can define a language $\tilde{L}$ by padding L_{A_5} with the neutral letter, corresponding to polynomial suffix mappings. This allows that $\tilde{L}$ can be recognized by a circuit of linear size, but this circuit is no longer FO[REG]-uniform. Still it would be $FO[<, +, *]$-uniform, actually this would be

also true if we started with a $FO[<,+,*]$-uniform TC^0 circuit. Hence if we could extend the uniformity to $FO[<,+,*]$ with the method presented in this paper, we would get a separation result of uniform TC^0 from NC^1.

Also in case there is a non-uniform TC^0 circuit recognizing L_{A_5}, we could find a polynomial advice encoding the connection predicate of the circuit, and hence apply our method if we would allow arbitrary suffix mappings. This means, by lifting the set of predicates, we could get around the limitation of two variables, and by allowing arbitrary suffix mappings one could even, at least hope to separate non-uniform TC^0 from NC^1.

Acknowledgments. We thank Klaus-Jörn Lange for valuable discussions and for many helpful comments. We also thank the anonymous referees for their helpful comments to improve the quality of the paper.

References

1. Barrington, D.A.M., Immerman, N., Straubing, H.: On uniformity within NC^1. J. Comput. Syst. Sci. 41, 274–306 (1990)
2. Krebs, A., Lange, K.J., Reifferscheid, S.: Characterizing TC^0 in terms of infinite groups. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 496–507. Springer, Heidelberg (2005)
3. Behle, C., Lange, K.J.: FO[<]-uniformity. In: IEEE Conference on Computational Complexity, pp. 183–189 (2006)
4. Koucký, M., Lautemann, C., Poloczek, S., Thérien, D.: Circuit lower bounds via Ehrenfeucht-Fraisse games. In: IEEE Conference on Computational Complexity, pp. 190–201 (2006)
5. Thérien, D., Wilke, T.: Over words, two variables are as powerful as one quantifier alternation. In: STOC, pp. 234–240 (1998)
6. Straubing, H., Thérien, D.: Weakly iterated block products of finite monoids. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 91–104. Springer, Heidelberg (2002)
7. Straubing, H., Thérien, D.: Regular languages defined by generalized first-order formulas with a bounded number of bound variables. Theory Comput. Syst. 36(1), 29–69 (2003)
8. Behle, C., Krebs, A., Mercer, M.: Linear circuits, two-variable logic and weakly blocked monoids. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 147–158. Springer, Heidelberg (2007)
9. Barrington, D.A.M.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. J. Comput. Syst. Sci. 38, 150–164 (1989)
10. Allender, E., Koucký, M.: Amplifying lower bounds by means of self-reducibility. In: IEEE Conference on Computational Complexity, pp. 31–40 (2008)
11. Straubing, H.: Finite Automata, Formal Logic, and Circuit Complexity. Birkhäuser, Basel (1994)
12. Pin, J.E.: Varieties of formal languages. Plenum, London (1986)
13. Ginsburg, S., Spanier, E.H.: Semigroups, presburger formulas, and languages. Pacific journal of Mathematics 16, 285–296 (1966)
14. Lautemann, C., McKenzie, P., Schwentick, T., Vollmer, H.: The descriptive complexity approach to LOGCFL. J. Comput. Syst. Sci. 62, 629–652 (2001)
15. Rhodes, J.L., Tilson, B.: The kernel of monoid morphisms. J. Pure Applied Alg. 62, 27–268 (1989)

Reoptimization of Traveling Salesperson Problems: Changing Single Edge-Weights

Tobias Berg and Harald Hempel

Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität Jena, 07740 Jena, Germany
tberg@minet.uni-jena.de, hempel@informatik.uni-jena.de

Abstract. We consider the following optimization problem: Given an instance of an optimization problem and some optimum solution for this instance, we want to find a good solution for a slightly modified instance. Additionally, the scenario is addressed where the solution for the original instance is not an arbitrary optimum solution, but is chosen amongst all optimum solutions in a most helpful way. In this context, we examine reoptimization of the travelling salesperson problem, in particular MinTSP and MaxTSP as well as their corresponding metric versions. We study the case where the weight of a single edge is modified. Our main results are the following: existence of a 4/3-approximation for the metric MinTSP-problem, a 5/4-approximation for MaxTSP, and a PTAS for the metric version of MaxTSP.

1 Introduction

The travelling salesperson problem is one of the best known and best studied NP-optimization problems. A lot is known about its approximability. For instance, the minimization version of TSP (MinTSP) is known to be not efficiently approximable, whereas its metric version allows an 1.5-approximation [1]. In contrast, the maximization version of TSP (MaxTSP) has constant factor approximations in both cases, namely a 4/3-approximation in the general case [2] and an 8/7 approximation in the metric case [3].

In this paper, we study *reoptimization* aspects of TSP. Here, reoptimization is the problem, given an optimum solution to an original instance, of finding a good solution for a slightly modified instance. The problem of reoptimizing MinTSP and MaxTSP with respect to insertion or deletion of vertices has been considered in [4]. We complement these results by considering another elementary modification, namely the change of a single edge-weight. This reoptimization problem has already been examined in [5]. There it is shown, that reoptimization allows a 7/5-approximation for the metric version of MinTSP. We improve on this result by giving a 4/3-approximation. Furthermore, we are able to show several other positive results, for example, a 5/4-approximation when increasing the weight of an edge in a MaxTSP-instance and an PTAS for increasing or decreasing edge-weights in the metric case of MaxTSP. Furthermore, we introduce for the first

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 141–151, 2009.

time in the context of reoptimization a stronger notion of hardness of a problem. That is, usually it is sufficient to show that there *exist* optimum solutions of the original instance, that are not helpful for finding a good solution. In contrast, adopting ideas from [6], we prove in this paper that most reoptimization problems are hard, for *all* choices of optimum solutions.

The paper is organized as follows. In the next section, we provide basic notations and definitions. In Section 3, we present our results for the problem MinTSP. The maximization version of TSP is examined in Section 4.

2 Preliminaries

For a graph G, let $V(G)$ and $E(G)$ denote the set of vertices and the set of edges of G, respectively. The degree of a vertex u in a graph G, short $deg_G(u)$, is the number of edges that are incident to u. A weighted graph is a pair that consists of a graph (V, E) and a weight function $w : E \to \mathbb{N}$. A weighted graph (G, w) is said to satisfy the triangle inequality if and only if for all vertices $s, u, v \in V(G)$ it holds that $w(\{s, u\}) + w(\{u, v\}) \geq w(\{s, v\})$. For a graph G and a subset S of its vertices, we denote the subgraph induced by S by $G_{[S]}$.

We now formally state one of our main problems, namely, finding a minimum solution in a TSP instance in which the weight of an edge is increased (decreased).

Problem: *inc*-MinTSP (*dec*-MinTSP)

Instance: A complete graph G, two edge cost functions $w^o, w^m : E(G) \to \mathbb{N}$ that coincide for all but one edge e, where $w^o(e) \leq w^m(e)$ (resp., $w^o(e) \geq w^m(e)$), and a minimum-cost Hamiltonian cycle in G w.r.t. w^o.

Solution: A Hamiltonian cycle T in G.

Measure: The length of the tour T w.r.t. w^m, i.e., $cost(G, T) = \sum_{e \in T} w^m(e)$.

If a weighted graph G^o is modified to a graph G^m by increasing the weight of an edge e by the amount of i we write $G^m = inc(G, (e, i))$. The cost of a minimum weight Hamiltonian cycle in G^m will be denoted by $opt(G^m)$.

Optimization problems *inc*-MaxTSP and *dec*-MaxTSP can be defined in the same way. In this paper, we will also consider several reoptimization problems c-A, where either c is a restricted form of *inc* or *dec*, or A is an optimization problem whose inputs satisfy an additional condition, e.g., satisfy the triangle inequality. We desist from giving formal definitions for all these problems c-A and instead appeal to the reader's intuition in defining these problems.

3 Minimum Travelling Salesperson (MinTSP)

In this section we study the problem of reoptimizing MinTSP with respect to *inc* (increasing the weight of an edge) and *dec* (decreasing the weight of an edge).

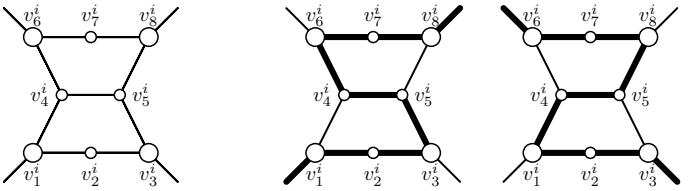


Fig. 1. The graph gadget H_i and its two traversals

Theorem 1 ([7]). *Let p be a polynomial. Unless* P = NP*, dec-MinTSP and inc-MinTSP are not $p(|V|)$-approximable.*

The proof is based on showing that there are original instances G^o such that, even if a selected optimum solution T^o_{opt} of G^o is given, a good approximation of a solution in a slightly modified graph G^m makes the NP-complete problem of finding a Hamiltonian cycle solvable in polynomial time. A close look at the proof of Theorem 1 from [7] reveals that in fact a stronger statement can be made. We are able to show that the nonapproximability of modified instances of TSP is independent of the given optimum solution for the original instance. Aside from that our modified construction yields a better nonapproximability bound.

Theorem 2. *Unless* P = NP*, the problems inc-MinTSP and dec-MinTSP are not $2^{|V|}$-approximable, for any choice of given optimum solution.*

Proof. (Sketch) We first consider the case *inc*: Before diving right into the proof, we give a rough outline. We show that if *inc*-MinTSP is $2^{|V|}$-approximable, then the NP-complete problem of finding a Hamiltonian cycle (HC) is in P. Given a graph G, we construct from G a MinTSP instance G^o with a known optimum solution T^o_{opt}, which is modified to G^m by increasing the weight of an edge. In addition, the instance G^o shall contain exactly one optimum solution, so that this is the only choice for an optimum solution. Finally, we show that a $2^{|V|}$-approximation for G^m can be used to decide if $G \in$ HC.

In detail, the graph G^o is constructed as follows. Let $G = (\{v_1, ..., v_n\}, E)$ be a graph. For each vertex v_i of G we take into G^o a copy of the graph gadget H_i as depicted in Figure 1. This gadget has the useful property, that it only has two Hamiltonian traversals, one that enters/leaves at v_1^i and v_8^i, and another one that enters/leaves at v_3^i and v_6^i. We connect the gadgets among each other via the edges $\{v_8^i, v_1^{i+1}\}$, $1 \leq i \leq n-1$. Furthermore, we add the edge $\{v_8^n, v_1^1\}$. In consequence, the resulting graph contains the Hamiltonian cycle

$$C = (v_1^1, v_2^1, v_3^1, v_5^1, v_4^1, v_6^1, v_7^1, v_8^1, v_1^2, ..., v_8^{n-1}, v_1^n, v_2^n, v_3^n, v_5^n, v_4^n, v_6^n, v_7^n, v_8^n, v_1^1).$$

Also, for every edge $\{v_i, v_j\} \in E$ we add the edges $\{v_3^i, v_6^j\}$ and $\{v_3^j, v_6^i\}$. Let G' denote the graph constructed so far. It is not hard to verify that (a) G' has a Hamiltonian cycle different from C if and only if $G \in$ HC and (b) every Hamiltonian cycle in G' besides C uses the edge $\{v_1^1, v_4^1\}$ but not the edge $\{v_3^1, v_5^1\}$. To get

a MinTSP-instance G^o we assign the weight 1 to all edges in G', except the edge $\{v_1^1, v_4^1\}$ which gets weight 2. In order to make G^o a complete graph we insert into G' all edges not mentioned so far and assign weight $2^{8n}(8n+1)$ to them. This completes the construction of the graph G^o. The graph G^m is obtained from G^o by increasing the weight of the edge $\{v_3^1, v_5^1\}$ from 1 to $2^{8n}(8n+1)$.

Note that G^o has exactly one optimum solution (minimum weight TSP-tour), namely C with cost $|V(G')| = 8n$. Also, the graph G^m has a TSP-tour of cost $8n+1$ only if $G \in$ HC. Otherwise, the best TSP-tour in G^m has weight at least $2^{8n}(8n+1)+1$. Thus, a $2^{|V|}$-approximation for *inc*-MinTSP gives a solution of size at most $2^{8n}(8n+1)$ if and only if $G \in$ HC. This yields a polynomial-time algorithm to decide HC.

The proof for *dec*-MinTSP is similar. We use the same auxiliary graph G' as above, but assign the following weights to the edges of G^o:

$$w(e) := \begin{cases} 1, & \text{if } e \in E(G') \setminus \{\{v_1^1, v_4^1\}, \{v_3^1, v_5^1\}\}, \\ 2^{8n}(8n+1), & \text{if } e = \{v_3^1, v_5^1\}, \\ 2^{8n}(8n+2), & \text{otherwise.} \end{cases}$$

Note that any Hamiltonian tour through G' has to use one of the edges $\{v_1^1, v_4^1\}$ or $\{v_3^1, v_5^1\}$. Consequently, C' is the sole optimum tour in G^o.

We modify G^o to G^m by decreasing the weight of the edge $\{v_1^1, v_4^1\}$ from $2^{8n}(8n+2)$ to 1. Thus, if $G \in$ HC then G^m has a tour of size $8n$, otherwise a tour in G^m has cost at least $2^{8n}(8n+1)$. □

We have seen that the concept of reoptimization is not really beneficial for the problem MinTSP. But this is not the case for all optimization problems. For instance, we are able to improve on the best known upper approximation bound of 1.5 for MinTSP$_\Delta$ (see [1]), a subset of MinTSP that contains all instances that satisfy the triangle inequality. Since changing the weight of a single edge may lead to violation of the triangle inequality, we consider the following modifications inc_Δ and dec_Δ for the problem MinTSP$_\Delta$:

$$inc_\Delta(G,(e,i)) := \begin{cases} inc(G,(e,i)), & \text{if } G \text{ and } inc(G,(e,i)) \text{ satisfy} \\ & \text{the triangle inequalilty,} \\ ((\emptyset,\emptyset),\emptyset), & \text{otherwise.} \end{cases}$$

The modification dec_Δ is defined in the same way. It was shown in [5] that dec_Δ-MinTSP$_\Delta$ and inc_Δ-MinTSP$_\Delta$ are 7/5-approximable. In the same paper the authors established the following lemma.

Lemma 1 ([5]). *Let G^o be a weighted graph, $e \in E(G^o)$ be an edge of G^o, $i \in \mathbb{N}$, $c \in \{dec_\Delta, inc_\Delta\}$, and $G^m := c(G^o,(e,i))$. If G^o and G^m satisfy the triangle inequality, then every edge incident to e has cost at least $i/2$.*

Using a different and refined analysis while applying Lemma 1 and drawing on constructions already contained in [4] we improve the approximation factor of 7/5 for dec_Δ-MinTSP$_\Delta$ and inc_Δ-MinTSP$_\Delta$ given in [5].

Theorem 3. *The problem dec_Δ-MinTSP$_\Delta$ is 4/3-approximable.*

Proof. Given an original graph G^o, a modified graph $G^m := dec_\Delta(G^o, (e, i))$, and an optimum tour T^o_{opt} for G^o, an algorithm A that approximates a tour for G^m with a factor 4/3 works as follows. The case $G^m = ((\emptyset, \emptyset), \emptyset)$ is trivial. Otherwise, A computes a solution T^m_{Chr} for G^m using Christofides' algorithm [1]. We will now argue that the better of the both tours T^o_{opt} and T^m_{Chr} yields a 4/3-approximation.

Without going into detail, we state that in general the cost of T^m_{Chr} in G^m, $cost(G^m, T^m_{Chr})$, can be bounded by the size of a minimum spanning tree of G^m (short, $MST(G^m)$) added to the size of a minimum perfect matching M between the vertices of odd degree in $MST(G_m)$. Also, the size of M is bounded by $\frac{1}{2}opt(G^m)$. For details see [1] or [8]. By Lemma 1, an optimum tour T^m_{opt} in G^m uses at least one edge of weight at least $\frac{i}{2}$. Since T^m_{opt} without that $i/2$-weight edge is a spanning tree, we have that $MST(G^m) \leq opt(G^m) - \frac{i}{2}$. Thus, $cost(G^m, T^m_{Chr}) \leq \frac{3}{2}opt(G^m) - \frac{i}{2}$.

For the tour T^o_{opt} we have that $cost(G^m, T^o_{opt}) \leq opt(G^m) + i$, since otherwise we would have a better tour than T^o_{opt} for G^o by taking a tour having length $opt(G^m)$ in G^m as a tour in G^o. In case $i \leq \frac{1}{3}opt(G^m)$ the tour T^o_{opt} yields a 4/3-approximation. In case $i > \frac{1}{3}opt(G^m)$ the tour T^m_{Chr} is a 4/3-approximation. □

This proof translates mutatis mutandis to inc_Δ, resulting in a 4/3-approximation for inc_Δ-MinTSP$_\Delta$.

Theorem 4. *The problem inc_Δ-MinTSP$_\Delta$ is 4/3-approximable.*

Proof. Let $G^m := inc_\Delta(G^o, (e, i))$. Obviously, $opt(G^o) \leq opt(G^m)$. Also,

$$cost(G^m, T^o_{opt}) = \begin{cases} opt(G^o) + i, \text{ if } e \text{ is part of } T^o_{opt}, \\ opt(G^o), \quad \text{otherwise,} \end{cases}$$

and therefore $cost(G^m, T^o_{opt}) \leq opt(G^o) + i$. By combining the two inequalities we get that $cost(G^m, T^o_{opt}) \leq opt(G^m) + i$. The remaining part of the proof proceeds in analogy to the proof of Theorem 3. □

We can generalize the above idea to the case in which the weights of k edges are decreased (increased).

Theorem 5. *The problems dec^k_Δ-MinTSP$_\Delta$ and inc^k_Δ-MinTSP$_\Delta$ are approximable with ratio $\frac{3k+1}{2k+1}$, for any $k \in \mathbb{N}$.*

Proof. (Sketch) As in the proof of Theorem 3 we output the better one of the old solution and the solution obtained by Christofides' algorithm. Let $i_1, ..., i_k$ be the amounts by which the edge weights are decreased. The analysis relies on the facts that

$$cost(G^m, T^o_{opt}) \leq opt(G^m) + k \cdot \max_{1 \leq j \leq k} i_j \ , \text{ and}$$
$$cost(T^m_{Chr}) \leq \frac{3}{2}opt(G^m) - \max_{1 \leq j \leq k} \frac{i_j}{2}.$$

If $\max_{1\le j\le k} i_j \le opt(G^m)/(2k+1)$ then T^o is a $\frac{3k+1}{2k+1}$-approximation, otherwise T^m_{Chr} yields such a bound. □

Besides these positive results, we can show the following lower bound.

Theorem 6. *Unless* P = NP*, there exists no* FPTAS *for* dec_Δ*-MinTSP*$_\Delta$ *and* inc_Δ*-MinTSP*$_\Delta$*, for any choice of the given optimum solution.*

Proof. First, we show that dec_Δ-MinTSP$_\Delta$ has no FPTAS. Assume to the contrary that there is an FPTAS for dec_Δ-MinTSP$_\Delta$. Let A denote such an algorithm that, given $\varepsilon > 0$ and a weighted graph G, outputs a tour T^A with $cost(G, T^A) \le (1+\varepsilon)\cdot opt(G)$ in time $p(1/\varepsilon, |V(G)|)$, where p is a polynomial. We show that under this assumption HC ∈ P.

Let $G = (V, E)$ be a graph, $|V| = n$. We construct the graph G' from G in the same way as in the proof of Theorem 2. From G' we construct an MinTSP$_\Delta$-instance G^o by assigning cost 3 to the edge $\{v_1^1, v_4^1\}$, cost 2 to all other edges of G', and cost 3 to all edges in the complete graph G^o that are not in G'. Note that any graph with weights 2 and 3 satisfies the triangle inequality, and even so if the weight of a single edge is reduced to 1.

Observe that $|V(G^o)| = 8n$ and note that the tour

$$T^o_{opt} = (v_1^1, v_2^1, v_3^1, v_5^1, v_4^1, v_6^1, v_7^1, v_8^1, v_1^2, ..., v_8^{n-1}, v_1^n, v_2^n, v_3^n, v_5^n, v_4^n, v_6^n, v_7^n, v_8^n, v_1^1),$$

is the sole optimum tour in G^o and has cost $16n$. Now we modify the graph G^o by decrementing the cost of the edge $\{v_1^1, v_4^1\}$ to 1. The resulting graph is called G^m. Note that T^o_{opt} still has cost $16n$ in G^m, therefore $opt(G^m) \le 16n$. In addition, we claim that G^m has a tour with cost $16n-1$ if and only if G has a Hamiltonian cycle. For a proof of this fact, note that a tour with cost $16n-1$ has to use the edge $\{v_1^1, v_4^1\}$ and has to avoid all edges with cost 3. Consequently, it traverses the gadget H_1, and also all other gadgets H_i, via $(v_3^i, v_2^i, v_1^i, v_4^i, v_5^i, v_8^i, v_7^i, v_6^i)$, $1 \le i \le n$. This is possible if and only if G itself is Hamiltonian.

Let $\varepsilon = 1/(17n)$. For every tour T with $cost(G^m, T) = opt(G^m) + 1$ we have

$$\begin{aligned} cost(G^m, T) &= \left(1 + \frac{1}{opt(G^m)}\right) opt(G^m) \\ &\ge \left(1 + \frac{1}{16n}\right) opt(G^m) \\ &> (1+\varepsilon) opt(G^m). \end{aligned}$$

Consequently, the output T^A of the algorithm $A(G^m, \varepsilon)$ is an optimum solution of G^m. Now, $cost(T^A) = 16n - 1$ if and only if $G \in$ HC. The assertion follows from the fact that the running time of $A(G^m, \varepsilon)$ is bounded by $p(1/\varepsilon, 8n) = p(17n, 8n)$.

The proof for inc_Δ-MinTSP$_\Delta$ is essentially the same, but starting with the edge $\{v_1^1, v_8^n\}$ having weight 1 and increasing the weight of this edge to 3. □

4 Maximum Travelling Salesperson (MaxTSP)

In this section we study the maximization version of the travelling salesperson problem, also known as taxicab-ripoff problem. Without reoptimization, the best known approximation result for MaxTSP is a 4/3-approximation [2]. In contrast to MinTSP, the general MaxTSP problem benefits from reoptimization. In particular we will show that *inc*-MaxTSP is 5/4-approximable. Before we prove this result, we introduce some additional notation.

Definition 1 ([9]). *Let G be a graph and $f : V(G) \to \mathbb{N}$. An f-factor of G is a subgraph H of G such that $deg_H(v) = f(v)$ for all $v \in V(G)$.*

When f is a constant function, i.e., $f(v) = k$ for all $v \in V(G)$ and some fixed $k \in \mathbb{N}$,we get the notion of a k-factor. Note that a 1-factor of G is a perfect matching of G. A 2-factor of G is a partition of G into node disjoint cycles.

Lemma 2. *Let G be a weighted graph and $P = (p_1, ..., p_m)$, $m \geq 2$, be a path in G. There is a polynomial time algorithm that finds a maximum weight 2-factor for G that contains the path P.*

Proof. A maximum weight 2-factor for G that respects a given path $(p_1, ..., p_m)$ is induced by a maximum weight f-factor for G where

$$f(x) = \begin{cases} 0, \text{ if } x \in \{p_2, ..., p_{m-1}\}, \\ 1, \text{ if } x \in \{p_1, p_m\}, \\ 2, \text{ if } x \in V(G) \setminus \{p_1, ..., p_m\}. \end{cases}$$

Chapter 10.1. of [9] contains a reduction function, call it g, such that for any graph G, G has an f-factor if and only if $g(G)$ has a perfect matching. We can alter this reduction to yield a similar statement for weighted graphs. Utilizing an algorithm from [10] for finding a maximum weight perfect matching, we obtain an $O(n^3)$ algorithm for finding a maximum weight f-factor of G. □

We are now prepared to prove our main result of this section. The proof is in spirit similar to the proof of Theorem 5 in [4].

Theorem 7. *The problem inc-MaxTSP is 5/4-approximable.*

Proof. Let G^o denote the original instance and let T^o_{opt} be a maximum tour for G^o. Let $G^m := inc(G^o, (e, i))$ denote the modified graph, $e = \{u, v\}$, and let T^m_{opt} be a maximum tour for G^m. Note that $cost(G^o, T^o_{opt}) \geq cost(G^m, T^m_{opt}) - i$, since otherwise T^o_{opt} was not an optimum tour for G^o. Also, we assume that $e \notin T^o_{opt}$ and $e \in T^m_{opt}$, otherwise T^o_{opt} is an optimum tour in G^m and is chosen as output when compared to other solutions that are obtained in the coming.

1. **Case: $|V(G^o)|$ is even:** From T^o_{opt} we can obtain a perfect matching M with $cost(G^o, M) \geq cost(G^o, T^o_{opt})/2 \geq (cost(G^m, T^m_{opt}) - i)/2$. By adding the edge e to M we obtain a set M' that contains a path of length 3 and with $cost(G^m, M') \geq (cost(G^m, T^m_{opt}) + i)/2$.

Now, consider a 2-factor $F = (C_1, ..., C_\ell)$ of G^m such that (a) e is contained in C_1, (b) $|C_1| \geq 5$, and (c) is of maximum weight among all 2-factors of G^m that satisfy (a) and (b). Such a 2-factor can be found in polynomial-time by constructing a maximum weighted 2-factor for G^m that contains the path (r, s, t, u, v) (see Lemma 2), for all possibilities of expanding e to a path of length four, and selecting the costliest of these 2-factors. Since $e \in T^m_{opt}$ we have that $cost(G^m, F) \geq cost(G^m, T^m_{opt})$.

Applying the method of Serdyukov [2] (see also [11]) we can iteratively, for $p = 1, ..., \ell$, delete an edge from C_p and add this edge to M' such that the modified set M' is still a union of paths. For $C_1 = (r, s, t, u, v, ...)$ this is possible since r, s, t are endpoints of a path in M' (only u and v are no endpoints) but only two of them can be endpoints of the same path. Hence, one of $\{s, r\}$ or $\{s, t\}$ can be added to M'. For all other cycles C_p this is possible since only vertices from already processed cycles C_j, $1 \leq j < p$, can have degree 2 in M'. Thus, all (of at least 3) vertices of C_p are endpoints of some path in M' but only two of them can be endpoints of the same path.

By this procedure, the 2-factor F and the set of edges M' are transformed into two sets of paths P_1 and P_2 satisfying

$$\begin{aligned} cost(G^m, P_1) + cost(G^m, P_2) &= cost(G^m, M') + cost(G^m, F) \\ &\geq (cost(G^m, T^m_{opt}) + i)/2 + cost(G^m, T^m_{opt}). \end{aligned}$$

By taking the costlier of P_1 and P_2 we get a set of paths with cost larger than $\frac{3}{4}(cost(G^m, T^m_{opt}) + \frac{i}{4}$. Let T denote the completion of this partial tour to a cycle in G^m, since G^m is a complete graph T always exists.

Now for $i \leq \frac{1}{5} cost(G^m, T^m_{opt})$ the solution T^o_{opt} yields a 5/4-approximation, for $i > \frac{1}{5} cost(G^m, T^m_{opt})$ the tour T yields such a bound.

2. Case: $|V(G^o)|$ is odd: It is easy to obtain from T^o_{opt} a set of paths M with at most one path of length 2 such that (a) $cost(G^o, M) \geq cost(G^o, T^o_{opt})/2$, (b) inserting e into M prolongs the longest path in M, and (c) u and v are no endpoints in $M \cup \{e\}$. Thus, $M' := M \cup \{e\}$ only consists of paths of length 1, with the exception of a single path of length at most 4 containing u and v, and u, v being no endpoints of the path.

Now, consider a maximum cost 2-factor $F = (C_1, ..., C_\ell)$ of G^m that contains e as an edge in C_1 and $|C_1| \geq 8$. Since M' contains at most 3 vertices that are not an endpoint of a path in M', the cycle C_1 contains 3 consecutive vertices that are endpoints in M'. The rest of the proof translates mutatis mutandis from the case $|V(G^o)|$ is even. □

A similar result for *dec* has not yet been found. But it can be shown that both, *dec*-MaxTSP and *inc*-MaxTSP, probably have no FPTAS.

Theorem 8. *Unless* P = NP, *there is no* FPTAS *for inc-MaxTSP and dec-MaxTSP, for any choice of a given optimum solution.*

The Theorem follows immediately from the upcoming Theorem 11 in which it is shown that already some restricted versions of *dec*-MaxTSP and *inc*-MaxTSP have no FPTAS.

In analogy to MinTSP, we consider as special case of MaxTSP the problem MaxTSP_Δ, where the input instances are required to satisfy the triangle inequality. The problem MaxTSP_Δ is approximable with ratio $\frac{8}{7}$ when no old solution is given [3]. We can prove that there exists a PTAS for dec_Δ-MaxTSP_Δ by showing approximability of alternative solutions for MaxTSP_Δ — a result interesting in its own right.

Theorem 9. *Let G be a $MaxTSP_\Delta$-instance, T_{opt} be a maximum tour in G, and e be an edge of T_{opt}. We can find a tour T' such that e does not belong to T' and $cost(G,T') \geq \big(1 - \frac{2}{|V(G)|-2}\big) cost(G,T_{opt})$ in polynomial-time.*

Proof. Let $T_{opt} = (v_1, ..., v_n)$ be a maximum tour of the complete graph G on n vertices with weight function w. We assume without loss of generality that $e = \{v_n, v_1\}$. There exist three consecutive vertices v_{i-1}, v_i, v_{i+1} on T^o_{opt}, $2 \leq i \leq n-1$, such that $w(\{v_{i-1}, v_i\}) + w(\{v_i, v_{i+1}\}) \leq \frac{2}{n-2} cost(G, T_{opt})$. To see this, assume to the contrary that $w(\{v_{i-1}, v_i\}) + w(\{v_i, v_{i+1}\}) > \frac{2}{n-2} cost(G, T_{opt})$, for all $2 \leq i \leq n-1$. But then, by summarizing the weights of the paths $(v_{2i-1}, v_{2i}, v_{2i+1})$, $1 \leq i \leq \lfloor (n-1)/2 \rfloor$ in T_{opt} we get that $cost(G, T_{opt}) > (\frac{n-1}{2} - \frac{1}{2}) \cdot \frac{2}{n-2} \cdot cost(G, T_{opt})$, a contradiction.

By deleting the edges $\{v_n, v_1\}$, $\{v_{i-1}, v_i\}$, and $\{v_i, v_{i+1}\}$ from T_{opt} and inserting the edges $\{v_n, v_i\}$, $\{v_i, v_1\}$, and $\{v_{i-1}, v_{i+1}\}$ we obtain a tour T' which does not contain the edge e. Due to the triangle inequality the cost of the tour T_{opt} increases when taking the path (v_n, v_i, v_1) instead of the shortcut (v_n, v_1). Thus, T' is shortened by at most $w(\{v_{i-1}, v_i\}) + w(\{v_i, v_{i+1}\})$ compared to T_{opt}. □

Corollary 1. *The problem dec_Δ-$MaxTSP_\Delta$ has a* PTAS.

Proof. The only interesting case is that the modified edge e does belong to a maximum tour T^o_{opt} in the original graph G^o. Let $\varepsilon > 0$. In case that $\varepsilon \geq \frac{2}{|V(G)|-2}$ Theorem 9 yields a solution T' with $cost(G^o, T') \geq (1-\varepsilon) cost(G^o, T^o_{opt})$ that does not contain e. The assertion follows from the facts that $cost(G^o, T^o_{opt}) \geq cost(G^m, T^m_{opt})$ and that $cost(G^o, T') = cost(G^m, T')$. In case that $\varepsilon < \frac{2}{|V(G)|-2}$ we perform a brute force search for a maximum tour in G^m. □

The existence of a PTAS can also be shown when the modification is inc_Δ.

Theorem 10. *The problem inc_Δ-$MaxTSP_\Delta$ has a* PTAS.

Proof. Let $\varepsilon \geq 0$ and G be a MaxTSP_Δ-instance. If $\varepsilon < \frac{2}{|V(G)|-2}$ we perform a brute force search for an optimum solution in G^m. In the other case, $\varepsilon \geq \frac{2}{|V(G)|-2}$, we output the old solution T^o_{opt}. This suffices since $cost(G^m, T^o_{opt}) \geq \big(1-\varepsilon\big) cost(G^m, T^m_{opt})$.

To see this, take an optimum tour T^m_{opt} of G^m. If e is contained in T^m_{opt} we use Theorem 9 to get a tour T' that does not contain e and for which

$$cost(G^m, T') \geq \left(1 - \frac{2}{|V(G)|-2}\right) cost(G^m, T^m_{opt}) \geq \big(1-\varepsilon\big) cost(G^m, T^m_{opt}).$$

If e is not contained in T^m_{opt} we set $T' := T^m_{opt}$. Since the modified edge e is not contained in T' and since T^o_{opt} is a maximum in G^o we have

$$cost(G^m, T') = cost(G^o, T') \leq cost(G^o, T^o_{opt}) \leq cost(G^m, T^o_{opt}).$$

The assertion follows immediately. □

Regarding hardness, we are able to prove the following.

Theorem 11. *Unless* P = NP*, there is no* FPTAS *for* inc_Δ*-MaxTSP*$_\Delta$ *and* dec_Δ*-MaxTSP*$_\Delta$*, for any choice of optimum solution.*

Proof. The proof is similar to the proof of Theorem 6, except that we assign the weights 2, 3, and 4 to the edges, in order to satisfy the triangle inequality.

In more detail, we use the auxiliary graph G' from the proof of Theorem 2. For the case that the modification is inc_Δ we assign the weight 2 to the edge $\{v^1_1, v^1_4\}$ of G', the cost 3 to all other edges of G', and cost 2 to all edges missing in G'. The resulting graph is G^o, which clearly has a single maximum solution of size $3n'$. We modify G^m by increasing the weight of the edge $\{v^1_2, v^1_4\}$ from 2 to 4. The resulting graph G^m has a solution of size $3n'+1$ if and only if $G \in$ HC the rest of the proof is similar to the proof of Theorem 6.

In case that the modification is dec_Δ we assign the weight 4 to the edge $\{v^1_3, v^1_5\}$, the weight 3 to all other edges of G', and the weight 2 to all edges missing in G'. We modify this graph by decreasing the weight of the edge $\{v^1_3, v^1_5\}$ from 4 to 2. The rest of the proof is as usual. □

Acknowledgements. We thank the anonymous referees for their valuable suggestions.

References

1. Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem, Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh (1976)
2. Serdyukov, A.I.: An algorithm with an estimate for the traveling salesman problem of the maximum. Upravlyaemye Sistemy 25, 80–86 (1984)
3. Chen, Z.Z., Nagoya, T.: Improved approximation algorithms for metric MaxTSP. J. Comb. Optim. 13(4), 321–336 (2007)
4. Ausiello, G., Escoffier, B., Monnot, J., Paschos, V.T.: Reoptimization of minimum and maximum traveling salesman's tours. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 196–207. Springer, Heidelberg (2006)
5. Böckenhauer, H.J., Hromkovič, J., Mömke, T., Widmayer, P.: On the hardness of reoptimization. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 50–65. Springer, Heidelberg (2008)
6. Liberatore, P.: The complexity of modified instances. arXiv.org cs/0402053 (2004)
7. Böckenhauer, H.J., Forlizzi, L., Hromkovič, J., Kneis, J., Kupke, J., Proietti, G., Widmayer, P.: On the approximability of TSP on local modifications of optimally solved instances. Algorithmic Oper. Res. 2(2), 83–93 (2007)

8. Papadimitriou, C.H., Steiglitz, K.: Combinatorial optimization: algorithms and complexity. Dover Publications Inc., Mineola (1998); corrected reprint of the 1982 original
9. Lovász, L., Plummer, M.: Matching Theory. Annals of Discrete Mathematics, vol. 29. North-Holland, Amsterdam (1986)
10. Gabow, H.: Implementation of algorithms for maximum matching on nonbipartite graphs, Ph.D. Thesis, Stanford University (1974)
11. Barvinok, A., Gimadi, E.K., Serdyukow, A.I.: The Maximum Traveling Salesman Problem. In: Gutin, G., Punnen, A. (eds.) The Traveling Salesman Problem and Its Variations, pp. 585–608. Kluwer Academic Publishers, Dordrecht (2002)

Refinement and Consistency of Timed Modal Specifications*

Nathalie Bertrand[1], Sophie Pinchinat[2], and Jean-Baptiste Raclet[3]

[1] INRIA Rennes, France
[2] IRISA & Université Rennes 1, France
[3] INRIA Rhône-Alpes, France

Abstract. In the application domain of component-based system design, developing theories which support compositional reasoning is notoriously challenging. We define *timed modal specifications*, an automata-based formalism combining modal and timed aspects. As a stepping stone to compositional approaches of timed systems, we define the notions of refinement and consistency, and establish their decidability.

1 Introduction

The increasing complexity of computer systems has led to methodologies almost universally based on component assembling. Because in the system development process, some pieces may not be completed or are not yet available, analysis methodologies must rely on an abstract description of the components behaviour.

Logic-based formalisms, such as modal and temporal logics, are robust formal tools to express statements about the behaviours of computer systems. Unfortunately, logics do not relate well in general to compositional approaches; the description of a system as a collection of interacting components cannot be exploited. However, by conceding a loss of expressiveness, like confining attention to safety properties, satisfactory frameworks can be developed.

A convincing proposal is the *modal specification* approach of [1][1], inspired by [3]. Modal specifications are deterministic automata equipped with two types of transitions: *may*-transitions, that are optional, as opposed to *must*-transitions, that are obligatory. Arbitrary safety properties can be expressed, as well as some elementary liveness ones by using must-transitions. Moreover, the formalism subsumes interface automata of [4] as shown in [5]. The algebraic setting developed by [1] has nice features that lead to effective methods, which we recall here.

The *consistency* of a modal specification, i.e. whether it has a model, is decidable, and the finite model property holds. Inclusion of the sets of models can also be decided since it coincides with the *refinement preorder* on modal specifications. Also, the *greatest lower bound* of two modal specifications, referred to as *shared refinement* in [6], can be effectively computed. From a logic perspective, satisfiability, implication, and conjunction correspond to consistency, refinement, and greatest lower bound, respectively.

* This research was supported by the European COMBEST project.

[1] See [2] for a complete exposition.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 152–163, 2009.

Moreover, modal specifications behave well with regard to compositional reasoning: Raclet in [1] has introduced a *product* combinator between modal specifications which reflects the parallel product of models. Furthermore, the dual *quotient* combinator is extremely relevant for the incremental design of component-based systems. Modal specification-based approaches seem thus very promising to develop formal tools in this application domain. In particular, they should be amenable to apply to the challenging domain of embedded systems, provided the framework can take real-time aspects into account.

Towards this end, the present contribution extends the algebraic framework of [1] to a timed setting. *Timed modal specifications* provide a logical formalism which combines modal and timed statements. They generalize both modal specifications and timed automata, just as timed automata generalize ordinary automata, and modal specifications generalize ordinary automata, respectively. Decision methods for refinement and consistency are achieved by bridging timed modal specifications and (untimed) modal specifications, via a region-based construction. These results are stepping stones to compositional reasoning on timed systems, by validating a timed extension of the algebraic theory of Raclet [1].

The paper is organized as follows: we define timed modal specifications in Sect. 2, and present their semantics in Sect. 3. Section 4 is dedicated to the refinement preorder, and is followed by consistency issues in Sect. 5.

2 Timed Automata and Timed Modal Specifications

As a modal specification is a simple automaton with a distinction between may and must transitions, a timed modal specification can be seen as a timed automaton equipped with may and must edges. In the following, we recall basics on timed automata [7], and then introduce timed modal specifications.

2.1 Timed Automata

Let $\mathcal{X}$ be a finite set of *clocks*. A *clock valuation* over $\mathcal{X}$ is a mapping $\nu : \mathcal{X} \to \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of nonnegative reals. By $\mathcal{V}$, we represent the set of clock valuations over $\mathcal{X}$, and define $\overline{0} \in \mathcal{V}$ by $\overline{0}(x) = 0$ for all $x \in \mathcal{X}$. A *guard* over $\mathcal{X}$ is a finite conjunction of expressions of the form $x \sim c$ where $x \in \mathcal{X}$, $c \in \mathbb{N}$ is a *constant*, and $\sim \in \{<, \leq, =, \geq, >\}$. We denote by $\xi[\mathcal{X}]$ the set of guards over $\mathcal{X}$. For some fixed $N \in \mathbb{N}$, $\xi_N[\mathcal{X}]$ represents the set of guards involving expressions where the constants are smaller than or equal to N. The satisfaction relation $\models \subseteq \mathcal{V} \times \xi[\mathcal{X}]$ between clock valuations and guards is defined in a natural way: we write $\nu \models g$ whenever ν satisfies g. In the following, we will often abuse notation and write g to denote the set of valuations which satisfy the guard g.

We consider timed automata with possibly infinitely many states (also called locations), and with a slight abuse of terminology we still call them timed automata.

Definition 1. *Given $\mathcal{X}$ a set of clocks, Σ an alphabet and $N \in \mathbb{N}$, a* timed automaton *(*TA*) is a structure $\mathcal{C} = (C, c^0, \mathcal{X}, \Sigma, \delta)$ where C is a (possibly infinite)*

set of states, c^0 is the initial state, and $\delta \subseteq C \times \xi_N[\mathcal{X}] \times \Sigma \times 2^{\mathcal{X}} \times C$ is a transition relation. We call $(\Sigma, \mathcal{X}, N)$ the signature *of $\mathcal{C}$.*

From now on, we fix a signature $(\Sigma, \mathcal{X}, N)$. A *region*, denoted θ, is a set of clock-valuations which satisfy exactly the same guards of $\xi_N[\mathcal{X}]$. Given a region θ, we write $\mathsf{Succ}(\theta)$ for the union of all regions that can be obtained from θ by letting time elapse. We let Θ be the set of all regions.

Definition 2. *Given a timed automaton $\mathcal{C} = (C, c^0, \mathcal{X}, \Sigma, \delta)$, we can build its associated* region automaton *$R(\mathcal{C}) = (C \times \Theta, (c^0, \overline{0}), \Delta)$ over the alphabet $\Theta \times \Sigma \times 2^{\mathcal{X}}$. The transitions in $R(\mathcal{C})$ stem from those in $\mathcal{C}$ in the following way: for all $(c, g, a, r, c') \in \delta$, for each region θ such that (c, θ) is reachable from $(c^0, \overline{0})$, for each region $\theta'' \in \mathsf{Succ}(\theta) \cap g$, there exists $((c, \theta), \theta'', a, r, (c'\theta')) \in \Delta$, where θ' is the region obtained from θ'' by resetting clocks in r, which we write $\theta''[r := 0]$.*

Without loss of generality, we assume that any region automaton $R(\mathcal{C})$ is pruned, i.e. all its states are reachable. As an example, a TA $\mathcal{C}$ and its region automaton $R(\mathcal{C})$ are represented in the left part of Fig. 2, on page 159.

Note that any automaton over the alphabet $\Theta \times \Sigma \times 2^{\mathcal{X}}$ can be seen as a timed automaton over the signature $(\Sigma, \mathcal{X}, N)$. We introduce the operator T which transforms the former into the latter in a straightforward manner, in order to distinguish the two distinct interpretations of the same syntactic object.

Definition 3. *A* TA *$\mathcal{C}$ is* in normal form *if it is isomorphic to (the reachable part) of $(T \circ R)(\mathcal{C})$.*

A direct consequence of Definition 3 is the ability to associate a unique region to any state of a TA in normal form. One can easily be convinced that given a timed automaton $\mathcal{C}$, $(T \circ R)(\mathcal{C})$ is isomorphic to $(T \circ R)^2(\mathcal{C})$. As a consequence, any TA $\mathcal{C}$ can be *normalized* by letting its *normal form* be $\mathcal{C}\!\downarrow := (T \circ R)(\mathcal{C})$. Notice that normalizing a TA is safe with respect to its semantics, as stated by the following proposition:

Proposition 1. *Let $\mathcal{C}$ be a* TA*. The timed languages of $\mathcal{C}$ and $\mathcal{C}\!\downarrow$ coincide.*

Proof. We prove that the configuration graphs of $\mathcal{C}$ and $\mathcal{C}\!\downarrow$ are bisimilar, the above proposition is then a direct consequence. We recall that the configuration graph of a TA $\mathcal{C}$, defined in [7] is as follows. Vertices are configurations, and there are two kinds of edges. For all $d \in \mathbb{R}$ there is a delay-edge labeled by d between configurations (c, ν) and (c', ν') whenever $c = c'$ and $\nu' = \nu + d$ (defined as $\forall x \in \mathcal{X},\ (\nu + d)(x) = \nu(x) + d$). There is an action edge labeled with a between (c, ν) and (c', ν') if there exists $c \xrightarrow{g,a,r} c' \in \delta_{\mathcal{C}}$ such that ν satisfies the guard g and $\nu' = \nu[r := 0]$.

Given a valuation ν, we denote $[\nu]$ the region it belongs to. We define the following binary relation between configurations of $\mathcal{C}$ and of $(T \circ R)(\mathcal{C})$:

$$\sim := \{((c, \nu), (c, [\nu_1]), \nu) \mid \nu \in \mathsf{Succ}([\nu_1])\}$$

and show that $\sim$ is a bisimulation. By definition, $(c^0, \overline{0}) \sim ((c^0, \overline{0}), \overline{0})$. Assume now $(c, \nu) \sim ((c, [\nu_1]), \nu)$.

Any edge $(c,\nu) \xrightarrow{d} (c,\nu+d)$ in the configuration graph of $\mathcal{C}$ can be simulated by $((c,[\nu_1]),\nu) \xrightarrow{d} (c,[\nu_1]),\nu+d)$ in the configuration graph of $R(\mathcal{C})$, and vice versa. Moreover, we indeed have $(c,\nu+d) \sim (c,[\nu_1]),\nu+d)$, since $[\nu] \in \mathsf{Succ}([\nu'])$ entails $[\nu+d] \in \mathsf{Succ}([\nu'])$.

Assume $(c,\nu) \xrightarrow{a} (c',\nu')$, because of some transition $c \xrightarrow{g,a,r} c' \in \delta_{\mathcal{C}}$. Necessarily, $\nu \in g$ and $\nu' = \nu[r := 0]$. Because $[\nu]$ is a time successor of $[\nu_1]$, the transition $(c,[\nu_1]) \xrightarrow{[\nu],a,r} (c',[\nu'])$ exists in the region graph $R(\mathcal{C})$ and justifies the transition $((c,[\nu_1]),\nu) \xrightarrow{[\nu],a,r} ((c',[\nu']),\nu')$.

Reciprocally, assume $((c,[\nu_1]),\nu) \xrightarrow{a} ((c',\theta'),\nu')$ in the configuration graph of $R(\mathcal{C})$. Then, there exists $(c,[\nu_1]) \xrightarrow{\theta,a,r} (c',\theta')$ in $R(\mathcal{C})$, such that (1) $\nu \in \theta$, (2) $\nu' = \nu[r := 0] \in \theta'$, and (3) $\theta \subseteq \mathsf{Succ}([\nu_1])$.

Transition $(c,[\nu_1]) \xrightarrow{\theta,a,r} (c',\theta')$ in $R(\mathcal{C})$, stem from some $c \xrightarrow{g,a,r} c'$ in $\mathcal{C}$ with (4) $\theta' = \theta[r := 0]$ and (5) $\theta \subseteq g$.

By (1) and (5), $\nu \in g$. Since $c \xrightarrow{g,a,r} c'$ is a transition in $\mathcal{C}$, there must be an edge $(c,\nu) \xrightarrow{a} (c',\nu[r := 0])$ in $\mathcal{C}$'s configuration graph.

(2), $\nu[r := 0] = \nu'$, showing that $(c,\nu) \xrightarrow{a} (c',\nu')$. It remains to establish that $(c',\nu') \sim ((c',\theta'),\nu')$, that is $[\nu'] \in \mathsf{Succ}(\theta')$, which is immediate by (4). □

Even if this amounts to performing a normalization operation, we assume from now on that every TA is in normal form.

2.2 Timed Modal Specifications

Definition 4. *A* timed modal specification (TMS) *over the signature* $(\Sigma,\mathcal{X},N)$ *is a structure* $\mathcal{S} = (Q,q^0,\mathcal{X},\Sigma,\delta^m,\delta^M)$, *where*

- Q *is a finite set of states, and* $q^0 \in Q$ *is the unique initial state;*
- $\delta^m,\delta^M \subseteq Q \times \xi_N[\mathcal{X}] \times \Sigma \times 2^{\mathcal{X}} \times Q$ *are finite sets of transitions, with the requirements* $\delta^M \subseteq \delta^m$, *and* δ^m *and* δ^M *are deterministic.*
  - δ^m *is the set of* may*-transitions representing the allowed transitions. Given a may-transition* $(q,g,a,r,q') \in \delta^m$, q *is the source state,* q' *is the destination state,* $g \in \xi_N[\mathcal{X}]$ *is the guard that specifies the valuations for which the transition can be taken,* $a \in \Sigma$ *is the action labeling the transition and* $r \subseteq \mathcal{X}$ *is the set of clocks reset by the transition.*
  - δ^M *is the set of* must*-transitions representing the required transitions.*

Determinism of the transition relation means that for every state q, every action a and every region θ, there exists at most one transition $(q,g,a,r,q') \in \delta^m$. Assuming that modal specifications are deterministic is common in the untimed case since it allows to relate refinement and inclusion of sets of models. This is not the case when nondeterminism is allowed [8]. We will often write $q \xrightarrow{g,a,r} q'$ to denote $(q,g,a,r,q') \in \delta^M$ and $q \overset{g,a,r}{\dashrightarrow} q'$ to denote $(q,g,a,r,q') \in \delta^m$.

In the next section, we give the semantics of TMS in terms of a collection of timed automata-like models.

3 Timed Modal Specification Semantics

3.1 Models of Timed Modal Specifications

Models of untimed modal specifications [1] are prefix-closed languages and can be represented by (a priori infinite-state) automata. Given a TMS $\mathcal{S}$, its models are TA which relate to $\mathcal{S}$ via a simulation-like relation. Formally,

Definition 5. *Let $\mathcal{C} = (C, c^0, \mathcal{X}, \Sigma, \delta)$ be a* TA *and $\mathcal{S} = (Q, q^0, \mathcal{X}, \Sigma, \delta^m, \delta^M)$ be a* TMS. *$\mathcal{C}$ is a* model *of $\mathcal{S}$, written $\mathcal{C} \models \mathcal{S}$, if there exists a binary relation $\rho \subseteq C \times Q$ such that $(c^0, q^0) \in \rho$, and for all $(c, q) \in \rho$, the following holds:*

- *for every $q \xrightarrow{g,a,r} q' \in \delta^M$, and every region θ such that both (c, θ) and (q, θ) are reachable (in C and $\mathcal{S}$ respectively), there exist $n \in \mathbb{N}$, states $c_1 \cdots c_n \in C$, and guards $g_1, \cdots, g_n \in \xi[\mathcal{X}]$ with*
  - $\mathsf{Succ}(\theta) \cap g \subseteq \mathsf{Succ}(\theta) \cap \bigcup_{i=1}^{n} g_i$,
  - $c \xrightarrow{g_i,a,r} c_i \in \delta,\ \forall 1 \leq i \leq n$, *and*
  - $(c_i, q') \in \rho,\ \forall 1 \leq i \leq n$.
- *for all $c \xrightarrow{g,a,r} c' \in \delta$, there exist a state $q' \in Q$ and a guard $g' \in \xi[\mathcal{X}]$ with*
  - $g \subseteq g'$,
  - $q \overset{g',a,r}{\dashrightarrow} q' \in \delta^m$, *and*
  - $(c', q') \in \rho$.

Intuitively, the first condition of Definition 5 ensures that any move required by the specification (a must-transition) is reflected in the model, potentially split in several transitions; the second condition guarantees that any transition of the model is allowed in the specification (as a may-transition). In this latter condition, notice that because $\mathcal{C}$ is in normal form, the guard g in the transition $c \xrightarrow{g,a,r} c'$ is necessarily a region. Let us illustrate Definition 5 on an example.

Example 1. Consider the TMS $\mathcal{S}$ and TA $\mathcal{C}$ represented on Fig. 1, where dashed arrows denote transitions in $\delta^m \setminus \delta^M$ and plain arrows transitions in δ^M. Also, the action alphabet Σ is left implicit, and transitions are just labeled by guards and optional resets.

In order to show that $\mathcal{C}$ is a model of $\mathcal{S}$ we consider the intuitive simulation relation that associates all pairs (c_i, q_i) and show that this relation satisfies the

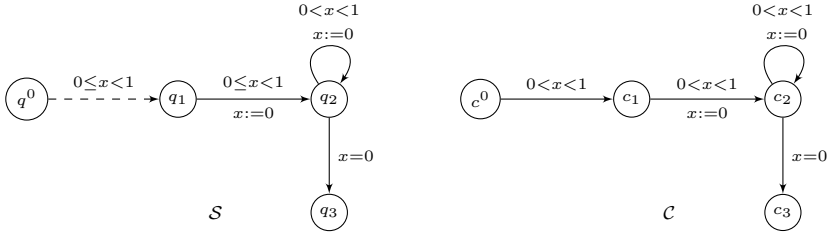


Fig. 1. A TMS $\mathcal{S}$ and a TA $\mathcal{C}$ with $\mathcal{C} \models \mathcal{S}$

conditions of Definition 5. We draw the reader's attention to two arguments: First, the transition $c^0 \xrightarrow{0<x<1} c_1$ is justified in $\mathcal{S}$ by a may-transition with a looser guard, namely $q^0 \overset{0\leq x<1}{\dashrightarrow} q_1$. Second, the must-transition $q_1 \xrightarrow{0\leq x<1, x:=0} q_2$ in $\mathcal{S}$ is correctly reflected in $\mathcal{C}$ since c_1 is only reachable in $\mathcal{C}$ within region $(0,1)$. Hence it suffices to consider the intersection of the must-transition guard $(0 \leq x < 1)$ with the time-successors of the reachable region $(0,1)$ as guard for the transition in the model $\mathcal{C}$.

Let us now define the following particular TMS $\mathcal{S}_\top$ that denotes the true formula (i.e. for every TA $\mathcal{C}$, $\mathcal{C} \models \mathcal{S}_\top$) by:

$$\mathcal{S}_\top := (\{q^0\}, q^0, \delta_\top^m, \delta_\top^M) \text{ with } \delta_\top^m = \{(q^0, \mathbf{true}, a, \emptyset, q^0) \mid a \in \Sigma\} \text{ and } \delta_\top^M = \emptyset$$

In the following, we write $\mathsf{Mod}(\mathcal{S})$ for the set of models of $\mathcal{S}$ and emphasize the fact that only TA in normal form are considered.

3.2 The Region-Based Interpretation

We show here that the semantics of TMS can be characterized by that of modal specifications via the transformation from timed automata to region automata. Modal specifications (MS), also called modal automata [3], correspond to the untimed variant of TMS. As we already dedicated significant space to TMS, we expect the reader to understand their untimed variants as objects like $\mathcal{R} = (P, p^0, Act, \Delta^m, \Delta^M)$, with an obvious interpretation of the components (states, initial state, set of actions, may-transitions, must-transitions).

An automaton $\mathcal{M} = (M, m^0, Act, \Delta)$ is a model of a MS $\mathcal{R} = (P, p^0, Act, \Delta^m, \Delta^M)$ (written $\mathcal{M} \models \mathcal{R}$) if there exists a binary relation $\rho \subseteq (M \times P)$ such that $(m^0, p^0) \in \rho$, and for all $(m, p) \in \rho$, the following holds:

- for every $p \xrightarrow{a} p' \in \Delta^M$ there is a transition $m \xrightarrow{a} m' \in \Delta$ and $(m', p') \in \rho$;
- for every $m \xrightarrow{a} m' \in \Delta$ there is a transition $p \xrightarrow{a} p' \in \Delta^m$ and $(m', p') \in \rho$.

The natural untimed object associated with a TMS $\mathcal{S}$, is its region modal automaton, which is obtained by generalizing the construction of the region automaton $R(\mathcal{C})$ for a TA $\mathcal{C}$ (Definition 2). An example of $\mathcal{S}$ and $R(\mathcal{S})$ is represented in Fig. 2 on page 159.

Notice that for every TA $\mathcal{C}$ and TMS $\mathcal{S}$ over the signature $(\Sigma, \mathcal{X}, N)$, the automaton $R(\mathcal{C})$ and the modal specification $R(\mathcal{S})$ share alphabet $\Theta \times \Sigma \times 2^{\mathcal{X}}$.

Proposition 2. *Let $\mathcal{M}$ be an automaton over the alphabet $\Theta \times \Sigma \times 2^{\mathcal{X}}$ and $\mathcal{S}$ be a* TMS. *If $\mathcal{M} \models R(\mathcal{S})$ then $T(\mathcal{M})$ is in normal form; moreover $T(\mathcal{M}) \models \mathcal{S}$.*

Proof. Assume $\mathcal{M}$ is a model of the modal specification $R(\mathcal{S})$ over the alphabet $\Theta \times \Sigma \times 2^{\mathcal{X}}$: each state m in $\mathcal{M}$ can be decorated with a unique region θ_m reflecting the valuation of the clocks when entering m. By definition of T, $T(\mathcal{M})$ and $\mathcal{M}$ have the same set of states. In order to distinguish between the set of states of $\mathcal{M}$ and the one of $T(\mathcal{M})$, we write the latter $T(M)$. From the simulation relation ρ' between $\mathcal{M}$ and $R(\mathcal{S})$, we define $\rho \subseteq T(M) \times Q$ by $(T(m), q) \in \rho$ if there exists θ such that $(m, (q, \theta)) \in \rho'$. Note that in this case, θ is uniquely determined.

Firstly, $(T(m^0), q^0) \in \rho$ since $(m^0, (q^0, \bar{0})) \in \rho'$. Let us pick $(T(m), q) \in \rho$. We prove that all requirements of the specification $\mathcal{S}$ appear in $T(\mathcal{M})$, and that all transitions in $T(\mathcal{M})$ are allowed in $\mathcal{S}$:

For any transition $q \xrightarrow{g,a,r} q'$ in $\mathcal{S}$, for any region θ such that (q, θ) is reachable in $R(\mathcal{S})$, and any region $\theta'' \in g \cap \mathsf{Succ}(\theta)$, there exists in $R(\mathcal{S})$ a must-transition $(q, \theta) \xrightarrow{\theta'',a,r} (q', \theta')$. Since $\mathcal{M}$ is a model of $R(\mathcal{S})$, there must be some transition $m \xrightarrow{\theta'',a,r} m'$ in $\mathcal{M}$ with $(m', (q', \theta')) \in \rho'$. Hence $(m', q') \in \rho$ (by definition of ρ). State m is solely reachable for a given clock region θ. Since the latter transition in $\mathcal{M}$ exists for all regions $\theta'' \in \mathsf{Succ}(\theta)$, we come with a collection of transitions in $\mathcal{M}$: $m \xrightarrow{g_i a,r} m_i$ such that $(m_i, q') \in \rho$, and $\mathsf{Succ}(\theta) \cap g = \mathsf{Succ}(\theta) \cap \bigcup_i g_i$.

Any transition $T(m) \xrightarrow{g,a,r} T(m')$ comes from some $m \xrightarrow{g,a,r} m'$. Since $\mathcal{M}$ is a model of $R(\mathcal{S})$, there exists a may-transition $(q, \theta) \overset{g,a,r}{\dashrightarrow} (q', \theta')$ in $R(\mathcal{S})$ with $(m', (q', \theta')) \in \rho'$. Note that this implies that g is a region. This transition appears in $\mathcal{S}$ under the form $q \overset{g',a,r}{\dashrightarrow} q'$, with $g \subseteq g'$ and $(m', q') \in \rho$. □

Theorem 1. *Let $\mathcal{C}$ be a* TA *(in normal form) and $\mathcal{S}$ be a* TMS. *Then*

$$\mathcal{C} \models \mathcal{S} \text{ if, and only if, } R(\mathcal{C}) \models R(\mathcal{S})$$

Proof. Let us first assume that $\mathcal{C} \models \mathcal{S}$. There exists a simulation relation ρ between $\mathcal{C}$ and $\mathcal{S}$ meeting requirements of Definition 5. We define $\rho' \subseteq (C \times \Theta) \times (Q \times \Theta)$ by: $((c, \theta), (q, \theta')) \in \rho'$ if $(c, q) \in \rho$ and $\theta = \theta'$. We show that ρ' is a simulation between $R(\mathcal{C})$ and $R(\mathcal{S})$, entailing $R(\mathcal{C}) \models R(\mathcal{S})$. First $((c^0, \bar{0}), (q^0, \bar{0})) \in \rho'$ by definition of ρ' and since $(c^0, q^0) \in \rho$. Let us pick $((c, \theta), (q, \theta)) \in \rho'$.

Any must-transition $(q, \theta) \xrightarrow{\theta'',a,r} (q', \theta')$ in $R(\mathcal{S})$ comes from some must-transition in $\mathcal{S}$: $q \xrightarrow{g,a,r} q'$ with $\theta'' \subseteq g$. Note that $\theta'' \in \mathsf{Succ}(\theta)$. Since $\mathcal{C}$ is a model of $\mathcal{S}$, and $(c, q) \in \rho$ there are states $c_1, \cdots, c_n$ and regions $\theta_1, \cdots, \theta_n$ such that $\mathsf{Succ}(\theta) \cap g \subseteq \mathsf{Succ}(\theta) \cap \bigcup_i \theta_i$, and for all i there is some transition $c \xrightarrow{\theta_i,a,r} c_i$ with $(c_i, q') \in \rho$. In particular, there is a transition in $\mathcal{C}$ of the form $c \xrightarrow{g',a,r} c'$ with $\theta'' \subseteq g'$ and $(c', q') \in \rho$. Back to $R(\mathcal{C})$, there must be a transition $(c, \theta) \xrightarrow{\theta'',a,r} (c', \theta')$ and $((c', \theta'), (q', \theta')) \in \rho'$.

Any transition $(c, \theta) \xrightarrow{\theta'',a,r} (c', \theta')$ in $R(\mathcal{C})$, comes from some $c \xrightarrow{g,a,r} c'$ in $\mathcal{C}$ with $\theta'' \subseteq g$. Since $\mathcal{C}$ is a model for $\mathcal{S}$, there must be several may-transitions in $\mathcal{S}$ of the form $q \overset{g_i,a,r}{\dashrightarrow} q_i \in \delta^m_{\mathcal{S}}$ with $g \subseteq \bigcup_i g_i$ and $(c', q_i) \in \rho$. In particular, there is a transition $q \overset{g',a,r}{\dashrightarrow} q'$ with $\theta'' \subseteq g'$ and $(c', q') \in \rho$. This transition appears in $R(\mathcal{S})$ as several transitions guarded by regions, one of which is $(q, \theta) \overset{\theta'',a,r}{\dashrightarrow} (q', \theta')$. Since $(c', q') \in \rho$, it follows that $((c', \theta'), (q', \theta')) \in \rho'$.

Assume now $R(\mathcal{C})$ is a model of the untimed modal specification $R(\mathcal{S})$ over the alphabet $\xi[\mathcal{X}] \times \Sigma \times 2^{\mathcal{X}}$. By Proposition 2, $(T \circ R)(\mathcal{C}) \models \mathcal{S}$. Since $\mathcal{C}$ is in normal form, $(T \circ R)(\mathcal{C})$ and $\mathcal{C}$ are isomorphic, hence $\mathcal{C} \models \mathcal{S}$. □

Note that Theorem 1 does not hold for arbitrary TA since its 'if'-part relies on the assumption that TA are in normal form. This assumption cannot be dispensed

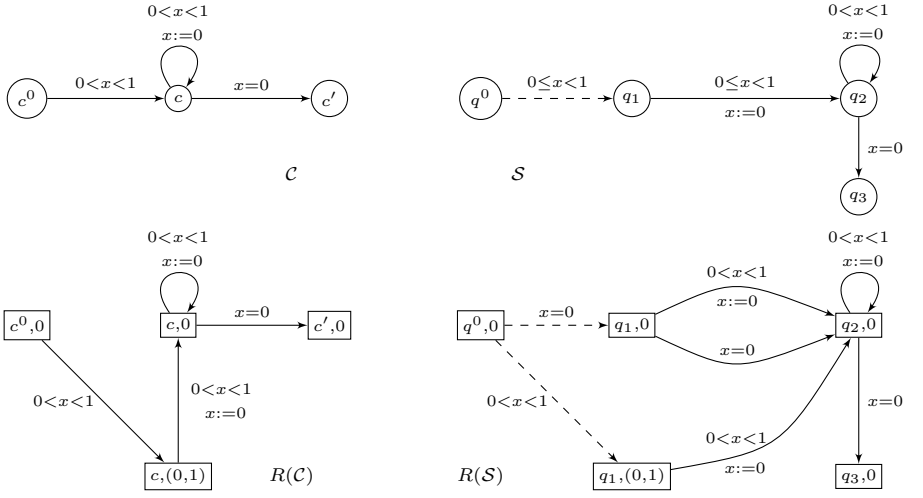


Fig. 2. A TA $\mathcal{C}$, a TMS $\mathcal{S}$ and their region automata

with altogether: the TA $\mathcal{C}$ of Fig. 2 is not in normal form, and is not a model of the TMS $\mathcal{S}$. And yet $R(\mathcal{C})$ is a model of $R(\mathcal{S})$. Indeed, $R(\mathcal{C})$ is obtained from $R(\mathcal{S})$ by cutting the may-transition $(q^0, 0) \overset{x=0}{\dashrightarrow} (q_1, 0)$, and keeping transition $(q^0, 0) \overset{x=0}{\dashrightarrow} (q_1, (0, 1))$.

Corollary 1. *It is decidable, given a* TMS $\mathcal{S}$ *and a* TA $\mathcal{C}$*, whether* $\mathcal{C} \models \mathcal{S}$.

Proof. This is an immediate consequence of Theorem 1 and the decidability result in [1] for the model relation of untimed modal specifications. □

Until now, TMS are provided with a clear semantics. We now wish to provide the framework with standard logic concepts. We successively define an implication-like relation, the *refinement*, and a conjunction-like operation, *the greatest lower bound*, that yield effective methods. Recall that we fixed a signature $(\Sigma, \mathcal{X}, N)$; this is necessary when we want to talk about refinement and consistency of timed modal specifications.

4 Refinement

Refinement is a preorder between specifications, and expresses that whenever $\mathcal{S}_1$ refines $\mathcal{S}_2$, $\mathcal{S}_1$ guarantees the properties $\mathcal{S}_2$ does, and maybe others. It happens to correspond in the untimed case to the inclusion of sets of models [1]. We now investigate the refinement preorder for TMS.

Thanks to the semantics of TMS, a good candidate for the refinement preorder is the one naturally inherited from the refinement preorder between MS, which we recall here.

Given two MS $\mathcal{R}_1 = (P_1, p_1^0, Act, \Delta_1^m, \Delta_1^M)$ and $\mathcal{R}_2 = (P_2, p_2^0, Act, \Delta_2^m, \Delta_2^M)$, $\mathcal{R}_1$ is *a refinement* of (or *refines*) $\mathcal{R}_2$, written $\mathcal{R}_1 \preceq \mathcal{R}_2$, if there exists a (simulation) relation $\rho \subseteq P_1 \times P_2$ such that $(p_1^0, p_2^0) \in \rho$, and for all $(p_1, p_2) \in \rho$, the following holds:

– for every $p_2 \xrightarrow{a} p_2' \in \Delta_2^M$ there exists $p_1 \xrightarrow{a} p_1' \in \Delta_1^M$ with $(p_1', p_2') \in \rho$;
– for every $p_1 \overset{a}{\dashrightarrow} p_1' \in \Delta_1^m$ there exists $p_2 \overset{a}{\dashrightarrow} p_2' \in \Delta_2^m$ with $(p_1', p_2') \in \rho$.

Definition 6. *Given two* TMS $\mathcal{S}_1$ *and* $\mathcal{S}_2$, $\mathcal{S}_1$ refines $\mathcal{S}_2$, *written* $\mathcal{S}_1 \preceq \mathcal{S}_2$, *whenever* $R(\mathcal{S}_1) \preceq R(\mathcal{S}_2)$. *We write* $\mathcal{S}_1 \equiv \mathcal{S}_2$ *whenever* $\mathcal{S}_1 \preceq \mathcal{S}_2$ *and* $\mathcal{S}_2 \preceq \mathcal{S}_1$.

Lemma 1. *For every* TMS, $\mathcal{S} \equiv (T \circ R)(\mathcal{S})$

Proof. Lemma 1 relies on the observation that given a TMS $\mathcal{S}$, $R(\mathcal{S})$ and $R((T \circ R)(\mathcal{S}))$ are isomorphic. Therefore, they are equivalent according to the refinement preorder on (untimed) modal specifications. □

Until now, we have required TMS to be finite-state (Definition 4). Actually, the framework smoothly extends to infinite-state TMS, so that TA, which may have infinitely many states, naturally embed into TMS as follows: given a TA $\mathcal{C} = (C, c^0, \mathcal{X}, \Sigma, \delta)$, we define the TMS $\mathcal{C}^* := (C, c^0, \mathcal{X}, \Sigma, \delta, \delta)$.

Lemma 2. *Let* $\mathcal{C}$ *be a* TA *and* $\mathcal{S}$ *a* TMS. *Then* $\mathcal{C} \models \mathcal{S}$ *if, and only if,* $\mathcal{C}^* \preceq \mathcal{S}$.

Proof. By Theorem 1, since $\mathcal{C}$ is in normal form, $\mathcal{C} \models \mathcal{S}$ is equivalent to $R(\mathcal{C}) \models R(\mathcal{S})$. Moreover, according to [1] for the untime setting, $R(\mathcal{C}) \models R(\mathcal{S})$ is equivalent to $R(\mathcal{C}) \models R(\mathcal{S})$, which by definition is equivalent to $\mathcal{C}^* \preceq \mathcal{S}$, since $R(\mathcal{C})$ $R(\mathcal{C}^*)$ are isomorphic. □

Theorem 2. *It is decidable whether, for any two* TMS $\mathcal{S}_1$ *and* $\mathcal{S}_2$, $\mathcal{S}_1 \preceq \mathcal{S}_2$. *Moreover,* $\mathcal{S}_1 \preceq \mathcal{S}_2$ *if, and only if,* $\mathsf{Mod}(\mathcal{S}_1) \subseteq \mathsf{Mod}(\mathcal{S}_2)$.

Proof. $R(\mathcal{S}_i)$ are *deterministic* modal automata (that is, their may and must transition functions are deterministic) thus the (untimed) refinement relation coincide with the inclusion of models [1]. This entails the decidability of the refinement relation for TMS.

Let us prove now that $\mathcal{S}_1 \preceq \mathcal{S}_2 \Leftrightarrow \mathsf{Mod}(\mathcal{S}_1) \subseteq \mathsf{Mod}(\mathcal{S}_2)$. The 'only if'-part is easy: Assume $\mathcal{S}_1 \preceq \mathcal{S}_2$, and consider $\mathcal{C} \models \mathcal{S}_1$. Then by Lemma 2 $\mathcal{C}^* \preceq \mathcal{S}_1$. Therefore $\mathcal{C}^* \preceq \mathcal{S}_2$, and again by Lemma 2 $\mathcal{C} \models \mathcal{S}_2$.

For the 'if'-part, assume it not the case that $\mathcal{S}_1 \preceq \mathcal{S}_2$. By definition, $R(\mathcal{S}_1) \preceq R(\mathcal{S}_2)$ does not hold either. Then there exists an automaton $\mathcal{M}$ with $\mathcal{M} \models R(\mathcal{S}_1)$ but $\mathcal{M} \not\models R(\mathcal{S}_2)$. By Proposition 2, $T(\mathcal{M})$ is in normal form and $T(\mathcal{M}) \models \mathcal{S}_1$. However $T(\mathcal{M}) \not\models \mathcal{S}_2$, otherwise, $(R \circ T)(\mathcal{M})$ would be a model of $R(\mathcal{S}_2)$. This is impossible since $(R \circ T)(\mathcal{M})$ is isomorphic to $\mathcal{M}$. Hence $T(\mathcal{M})$ is a witness for $\mathsf{Mod}(\mathcal{S}_1) \not\subseteq \mathsf{Mod}(\mathcal{S}_2)$. □

We extend the class of TMS (over a fixed signature) with an extra object, written $\mathcal{S}_\perp$, for which Definition 4 does not apply: $\mathcal{S}_\perp$ has an empty set of states, and thus empty sets of transitions, and no initial state. By convention, $\mathsf{Mod}(\mathcal{S}_\perp) = \emptyset$ and we extend the refinement preorder by letting $\mathcal{S}_\perp \preceq \mathcal{S}$ for every TMS $\mathcal{S}$. Note that all the properties established so far remain valid for this slight extension. Intuitively, with the logic point of view, $\mathcal{S}_\perp$ is meant to denote an antilogy, while dually, $\mathcal{S}_\top$ (defined in Sect. 2) denotes a tautology.

Lemma 3. $\mathcal{S}_\perp \preceq \mathcal{S} \preceq \mathcal{S}_\top$, *for any* TMS $\mathcal{S}$.

5 Consistency

Consistency of a specification is a standard property in logic which expresses the existence of a model. Notice that TMS have the finite model property: indeed given a TMS $\mathcal{S}$, we can decide whether $R(\mathcal{S})$ has a model, and if so, effectively synthesize an automaton $\mathcal{M}$ model of $R(\mathcal{S})$. $T(\mathcal{M})$ is then a TA (with finitely many states), and by Proposition 2, $T(\mathcal{M})$ is model of $\mathcal{S}$.

Consistency *between a pair of specifications* can also be considered, with the meaning that the specifications share a common model. In a pure logical setting, consistency between two specifications reduces to the consistency of a single one, by considering their conjunction. In this regard, we equip the TMS with a conjunction operator derived from the one originally proposed by [1] for the untimed case. We recall this untimed-case construction.

A universal principle when operating a conjunction is to focus on the strongest constraint in the operands. When the operands are untimed modal specifications, constraints refer to the modal status (may or must) of transitions, and the conjunction of two untimed modal specifications amounts to combine transitions in the following manner (recall that dashed arrows are may-transitions and solid arrows are must-transitions): for example, if $p_1 \xrightarrow{a} p_1'$ in the first (untimed) modal specification and $p_2 \overset{a}{\dashrightarrow} p_2'$ in the second (untimed) modal specification, then $(p_1, p_2) \xrightarrow{a} (p_1', p_2')$ is a transition of their conjunction.

Formalizing wholly this idea leads to the three following rules:

$$\frac{p_1 \xrightarrow{a} p_1' \quad p_2 \overset{a}{\dashrightarrow} p_2'}{(p_1,p_2) \xrightarrow{a} (p_1',p_2')} \qquad \frac{p_1 \overset{a}{\dashrightarrow} p_1' \quad p_2 \xrightarrow{a} p_2'}{(p_1,p_2) \xrightarrow{a} (p_1',p_2')} \qquad \frac{p_1 \overset{a}{\dashrightarrow} p_1' \quad p_2 \overset{a}{\dashrightarrow} p_2'}{(p_1,p_2) \overset{a}{\dashrightarrow} (p_1',p_2')}$$

Remark that constraints of each operand can be inconsistent, e.g. when $p_1 \xrightarrow{a} p_1'$ (a must occur) but there is no transition $p_2 \overset{a}{\dashrightarrow} p_2'$ (written $p_2 \overset{a}{\not\dashrightarrow}$ and meaning that a is forbidden in p_2). In that case, the product state (p_1, p_2) is *inconsistent*, and it is modeled by the ability from the compound state (p_1, p_2) to reach the particular state $\bot$ which precisely denotes inconsistency. Hence the two following additional rules.

$$\frac{p_1 \xrightarrow{a} p_1' \quad p_2 \overset{a}{\not\dashrightarrow}}{(p_1,p_2) \longrightarrow \bot} \qquad \frac{p_2 \xrightarrow{a} p_2' \quad p_1 \overset{a}{\not\dashrightarrow}}{(p_1,p_2) \longrightarrow \bot}$$

By the five rules above, we obtain a structure where the inconsistent state $\bot$ may be reachable. The *conjunction of the two untimed modal specifications* is the greatest sub-structure closed by must transitions and which does not contain the state $\bot$. Notice that this sub-structure may be empty. As proved by [1], the resulting delivers indeed the conjunction of the two specifications in the sense that it denotes the intersection of their models.

Definition 7. *The conjunction of the* TMS $\mathcal{S}_1$ *and* $\mathcal{S}_2$*, denoted* $\mathcal{S}_1 \wedge \mathcal{S}_2$ *is the* TMS $T(R(\mathcal{S}_1) \wedge R(\mathcal{S}_2))$*. If* $R(\mathcal{S}_1) \wedge R(\mathcal{S}_2)$ *is empty then* $\mathcal{S}_1 \wedge \mathcal{S}_2$ *is* $\mathcal{S}_\bot$*.*

Corollary 2. $\mathcal{S}_1 \wedge \mathcal{S}_2$ *is the* $\preceq$*-greatest lower bound of* $\mathcal{S}_1$ *and* $\mathcal{S}_2$*.*

Proof. Write $\mathcal{R}_i := R(\mathcal{S}_i)$. By [1], $\mathcal{R}_1 \wedge \mathcal{R}_2 \preceq \mathcal{R}_i$. Moreover, since $\mathcal{S}_1 \wedge \mathcal{S}_2 = T(\mathcal{R}_1 \wedge \mathcal{R}_2)$ and T is monotonic, we have $\mathcal{S}_1 \wedge \mathcal{S}_2 \preceq T(\mathcal{R}_i)$. Finally because

$T(\mathcal{R}_i) \equiv \mathcal{S}_i$ (Lemma 1), $\mathcal{S}_1 \wedge \mathcal{S}_2 \preceq \mathcal{S}_i$. We now show it is the greatest element under $\mathcal{S}_1$ and $\mathcal{S}_2$. Assume that there exists $\mathcal{S}$ such that $\mathcal{S} \preceq \mathcal{S}_i$. Therefore, by definition of $\preceq$, $R(\mathcal{S}) \preceq R_i$ which entails $R(\mathcal{S}) \preceq R_1 \wedge R_2$. Now, we have $\mathcal{S} \equiv T(R(\mathcal{S})) \preceq T(R_1 \wedge R_2)$ since T is monotonic; as $\mathcal{S}_1 \wedge \mathcal{S}_2 = T(R_1 \wedge R_2)$ by definition, we conclude. □

Corollary 3. $\mathsf{Mod}(\mathcal{S}_1 \wedge \mathcal{S}_2) = \mathsf{Mod}(\mathcal{S}_1) \cap \mathsf{Mod}(\mathcal{S}_2)$

Proof. From Corollary 2 we have $\mathcal{S}_1 \wedge \mathcal{S}_2 \preceq \mathcal{S}_i$. Then Theorem 2 entails, $\mathsf{Mod}(\mathcal{S}_1 \wedge \mathcal{S}_2) \subseteq \mathsf{Mod}(\mathcal{S}_i)$. Thus $\mathsf{Mod}(\mathcal{S}_1 \wedge \mathcal{S}_2) \subseteq [\mathsf{Mod}(\mathcal{S}_1) \cap \mathsf{Mod}(\mathcal{S}_2)]$. Let $\mathcal{C}$ be a TA such that $\mathcal{C} \in [\mathsf{Mod}(\mathcal{S}_1) \cap \mathsf{Mod}(\mathcal{S}_2)]$. By Lemma 2, $\mathcal{C}^* \preceq \mathcal{S}_1$ and $\mathcal{C}^* \preceq \mathcal{S}_2$. By Corollary 2, $\mathcal{C}^* \preceq \mathcal{S}_1 \wedge \mathcal{S}_2$ and by Lemma 2, $\mathcal{C} \models \mathcal{S}_1 \wedge \mathcal{S}_2$. As a result $[\mathsf{Mod}(\mathcal{S}_1) \cap \mathsf{Mod}(\mathcal{S}_2)] \subseteq \mathsf{Mod}(\mathcal{S}_1 \wedge \mathcal{S}_2)$. □

6 Conclusion

We introduced timed modal specifications as a logical formalism to combine modal and timed statements, and provided them with a decidable notion of refinement and a computable conjunction operator.

Regarding related work, *timed interfaces* have been proposed in [9] as a timed extension of interface automata from [4]. This framework explicitly distinguishes between the output actions (from the components of the system) and the input actions (from the environment). A decidable test of compatibility is developed between, on the one hand, the respective assumptions made by each individual component on its environment and, on the other hand, the guarantees that each component provides. However, no refinement preorder is considered. We also mention the work from [10], in which so-called timed modal specifications are introduced as well. Essentially, these are obtained as modal extensions of configuration graphs of timed automata, but presented instead as enriched CCS-like processes with durations and modalities. Because, the authors do not explicitly adopt a logical-based setting, their study of several types of refinement relations does not lead to addressing the corresponding lower-bound operators, thereby the work misses the crucial discussion about consistency.

In future work, we will extend product and quotient of [1] to timed modal specifications, borrowing know-hows from the untimed setting. Also, timed interfaces should be embeddable into timed modal specifications, via a construction in the line of [5] for the untimed setting. The main difficulty would be to establish that the compatibility relation of [9] is hereditary with respect to the present notion of refinement preorder; as a consequence, two compatible components could be implemented independently.

References

1. Raclet, J.B.: Residual for component specifications. In: Proceedings of FACS 2007 (2007)
2. Raclet, J.B.: Quotient de spécifications pour la réutilisation de composants. PhD thesis, Université de Rennes I (December 2007) (in French)

3. Larsen, K.G.: Modal specifications. In: Sifakis, J. (ed.) Automatic Verification Methods for Finite State Systems. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1990)
4. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Proceedings of FSE 2001, pp. 109–120 (2001)
5. Larsen, K.G., Nyman, U., Wasowski, A.: Modal I/O automata for interface and product line theories. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 64–79. Springer, Heidelberg (2007)
6. Doyen, L., Henzinger, T.A., Jobstmann, B., Petrov, T.: Interface theories with component reuse. In: de Alfaro, L., Palsberg, J. (eds.) EMSOFT 2008, pp. 79–88. ACM Press, New York (2008)
7. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
8. Larsen, K.G., Nyman, U., Wasowski, A.: On modal refinement and consistency. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 105–119. Springer, Heidelberg (2007)
9. de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Timed interfaces. In: Sangiovanni-Vincentelli, A.L., Sifakis, J. (eds.) EMSOFT 2002. LNCS, vol. 2491, pp. 108–122. Springer, Heidelberg (2002)
10. Čerāns, K., Godskesen, J.C., Larsen, K.G.: Timed modal specification - theory and tools. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 253–267. Springer, Heidelberg (1993)

Nondeterministic Instance Complexity and Proof Systems with Advice

Olaf Beyersdorff[1], Johannes Köbler[2], and Sebastian Müller[2,⋆]

[1] Institut für Theoretische Informatik, Leibniz-Universität Hannover, Germany
beyersdorff@thi.uni-hannover.de
[2] Institut für Informatik, Humboldt-Universität zu Berlin, Germany
{koebler,smueller}@informatik.hu-berlin.de

Abstract. Motivated by strong Karp-Lipton collapse results in bounded arithmetic, Cook and Krajíček [1] have recently introduced the notion of propositional proof systems with advice. In this paper we investigate the following question: *Given a language L, do there exist polynomially bounded proof systems with advice for L?* Depending on the complexity of the underlying language L and the amount and type of the advice used by the proof system, we obtain different characterizations for this problem. In particular, we show that the above question is tightly linked with the question whether L has small nondeterministic instance complexity.

1 Introduction

The classical Cook-Reckhow Theorem states that $\mathsf{NP} = \mathsf{coNP}$ if and only if the set of all tautologies TAUT has a polynomially bounded proof system, i.e., there exists a polynomial p such that every tautology φ has a proof of size $\leq p(|\varphi|)$ in the system. Consequently, showing super-polynomial lower bounds to the proof size in propositional proof systems of increasing strength provides one way to attack the P/NP problem. This approach, also known as the Cook-Reckhow program, has lead to a very fruitful research on the length of propositional proofs.

Motivated by strong Karp-Lipton collapse results in bounded arithmetic, Cook and Krajíček [1] have recently introduced the notion of propositional proof systems using advice. This model seems to be strictly more powerful than classical proof systems, as long-standing open problems, such as the existence of optimal proof systems, receive affirmative answers in this setting [1,2].

In the present paper we focus on the question whether there exist polynomially bounded proof systems with advice. We do not only consider propositional proof systems, but investigate this question for arbitrary proof systems and languages. As in the Cook-Reckhow Theorem above, we obtain a series of results which provide a complete complexity-theoretic characterization for this question.

In particular, we show a tight connection of this problem to the notion of nondeterministic instance complexity. Similarly as Kolmogorov complexity, instance complexity measures the complexity of individual instances of a language [3]. In

⋆ Supported by DFG grant KO 1053/5-2.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 164–175, 2009.

its nondeterministic version, Arvind, Köbler, Mundhenk, and Torán [4] used this complexity measure to show that, under reasonable complexity-theoretic assumptions, there are infinitely many tautologies that are hard to prove in every propositional proof system. In the light of our present contribution, this connection between nondeterministic instance complexity and proof complexity is strengthened by results of the following form: *all elements of a given language L have small instance complexity if and only if L has a proof system with advice such that every $x \in L$ has a short proof.*

To achieve these results, we start in Sect. 3 by reviewing the notion of nondeterministic instance complexity of [4]. Instance complexity is measured by two parameters: the size of the machine and its running time. The most interesting choice for these parameters seems to allow logarithmic size programs with polynomial running time. We combine all languages admitting such programs in the class $\mathsf{NIC}[\log, \mathsf{poly}]$. Using a proof idea from [3], we show the class $\mathsf{NIC}[\log, \mathsf{poly}]$ to lie properly between the nonuniform classes $\mathsf{NP}/\log$ and $\mathsf{NP}/\mathsf{poly}$.

In Sect. 4, we generalize the notion of propositional proof systems with advice of Cook and Krajíček [1] to arbitrary languages. For functional proof systems we consider three types of advice selectors: those that match the input length, those that match the output length, and arbitrary. Input advice turns out to be not really restrictive. Similarly as in [1], we show that for every language L, the class of all proof systems for L using logarithmic input advice contains an optimal proof system.

Our main results follow in Sects. 5 and 6 where we consider the plausibility of various languages having polynomially bounded proof systems with advice. In Sect. 5 we investigate this problem for arbitrary languages, whereas in Sect. 6 we focus on TAUT which presents the most interesting case for practical applications. At this point, instead of providing a comprehensive account, we will just explain a few central results from Sects. 5 and 6.

For output advice, the classical Cook-Reckhow Theorem generalizes in a straightforward manner, and thus a language L has a polynomially bounded proof system with logarithmic output advice if and only if $L \in \mathsf{NP}/\log$.

For input advice, which yields a strictly more powerful model, this question is more intricate. Here we establish the connection to nondeterministic instance complexity: a language L has a polynomially bounded proof system with logarithmic input advice if and only if $L \in \mathsf{NIC}[\log, \mathsf{poly}]$. While $\mathsf{NIC}[\log, \mathsf{poly}]$ and $\mathsf{NP}/\log$ are different classes, we prove that they do not differ on sets from coNP. Thus, for any language $L \in \mathsf{coNP}$, the notion of a polynomially bounded proof system with logarithmic advice is the same when considering advice in terms of per proof length or per instance length.

While this result also holds for a polynomial amount of advice, it appears to fail when reducing the amount of advice to constantly many bits, as unlikely collapse consequences would follow. Finally, we summarize the relative strengths of various proof systems by showing that the actual existence of polynomially bounded advice proof systems for TAUT produces different collapses of the polynomial hierarchy.

2 Preliminaries

We assume familiarity with standard complexity classes. In the following we just mention a few classes which occur in this paper. The *Boolean hierarchy* BH is the closure of NP under union, intersection, and complementation. The levels of BH are denoted BH_k, where BH_2 is also known as $\mathsf{D^P}$. The Boolean hierarchy coincides with $\mathsf{P}^{\mathsf{NP}[O(1)]}$ consisting of all languages which can be solved in polynomial time with constantly many queries to an NP oracle. If we allow $O(\log n)$ adaptive queries we get the presumably larger class $\mathsf{P}^{\mathsf{NP}[\log]}$.

Complexity classes with *advice* were first considered by Karp and Lipton [5]. For each function $h : \mathbb{N} \to \Sigma^*$ and each language L we let $L/h = \{x \mid \langle x, h(|x|)\rangle \in L\}$. If C is a complexity class and F is a class of functions, then $\mathsf{C}/F = \{L/h \mid L \in \mathsf{C}, h \in F\}$.

Cook and Reckhow [6] defined the notion of a *proof system* for an arbitrary language L quite generally as a partial polynomial-time computable function f with range L. A string w with $f(w) = x$ is called an f-proof for $x \in L$.

Proof systems are compared according to their strength by simulations as introduced in [6] and [7]. If f and g are proof systems for L, we say that g *simulates* f (denoted $f \leq g$), if there exists a polynomial p such that for all $x \in L$ and f-proofs w of x there is a g-proof w' of x with $|w'| \leq p(|w|)$. If such a proof w' can even be computed from w in polynomial time, we say that g *p-simulates* f and denote this by $f \leq_p g$. If the systems f and g mutually (p-)simulate each other they are called *(p-)equivalent*. A proof system for L is *(p-)optimal* if it (p-)simulates all proof systems for L. For a function $t : \mathbb{N} \to \mathbb{N}$, a proof system f for L is *t-bounded* if for all $x \in L$ there exists an f-proof of size at most $t(|x|)$. If t is a polynomial, then f is called *polynomially bounded*.

3 Nondeterministic Instance Complexity

While Kolmogorov complexity studies the hardness of individual strings, the notion of instance complexity was introduced by Orponen, Ko, Schöning, and Watanabe [3] to measure the hardness of individual instances of a given language. The deterministic instance complexity of [3] was later generalized to the nondeterministic setting by Arvind, Köbler, Mundhenk, and Torán [4].

As required for Kolmogorov complexity and instance complexity, we fix a universal Turing machine $U(M, x)$ which executes nondeterministic programs M on inputs x. In the sequel, we refrain from always mentioning U explicitly. Thus we simply write statements like "M is a t-time bounded Turing machine" with the precise meaning that U always spends at most $t(n)$ steps to simulate M on inputs of length n. Likewise, to "simulate a machine M on input x" always means executing $U(M, x)$.

A nondeterministic Turing machine M is *consistent* with a language L (or L-consistent), if $L(M) \subseteq L$. We can now give the definition of nondeterministic instance complexity from [4].

Definition 1 (Arvind et al. [4]). *For a set L and a time bound t, the t-time-bounded* nondeterministic instance complexity *of x with respect to L is defined as*

$$nic^t(x : L) = \min\{\, |M| \,:\, M \text{ is an } L\text{-consistent } t\text{-time-bounded nondeterministic machine, and } M \text{ decides correctly on } x \,\}\ .$$

Similarly as in the deterministic case in [3], we collect all languages with prescribed upper bounds on the running time and nondeterministic instance complexity in a complexity class.

Definition 2. *Let F_1 and F_2 be two classes of functions. We define*

$$\mathsf{NIC}[F_1, F_2] = \{L : \text{there exist } s \in F_1 \text{ and } t \in F_2 \text{ such that for all } x \in \Sigma^* \\ nic^t(x : L) \le s(|x|)\}\ .$$

A particularly interesting choice for the classes F_1 and F_2 is to allow polynomial running time, but only logarithmic descriptions for the machines. This leads to the class $\mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$ which plays a central role in this paper. Similarly as in the deterministic case (cf. [3]), the next proposition locates this class between the nonuniform classes NP/log and $\mathsf{NP}/\mathsf{poly}$.

Proposition 3. $\mathsf{NP}/\mathsf{log} \subseteq \mathsf{NIC}[\mathsf{log}, \mathsf{poly}] \subseteq \mathsf{NP}/\mathsf{poly}$.

Proof. For the first inclusion, let $L \in \mathsf{NP}/\mathsf{log}$. Let M be a nondeterministic Turing machine with logarithmic advice that decides L and let a_n be the advice given to M for inputs of length n. We define a collection of programs M_{n,a_n} for L as follows. On input x the machine M_{n,a_n} first checks, whether the length of the input is n. For this we need to code the number n into M_{n,a_n}. If $|x| \neq n$, then M_{n,a_n} rejects. Otherwise, M_{n,a_n} simulates M on input x with advice a_n which is also coded into M_{n,a_n}. Essentially, the machines M_{n,a_n} are constructed by hardwiring n and a_n into M, and thus the size of M_{n,a_n} is logarithmic in n. Therefore $L \in \mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$.

For the second inclusion, let $L \in \mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$. Then there exist a constant c and a polynomial p such that for all x we have $nic^p(x : L) \le c \log |x| + c$. We construct a nondeterministic Turing machine M with polynomial advice that accepts exactly L. The advice of M for length n consists of all nondeterministic Turing machines $M_1, \ldots, M_m$ of size at most $c \log n + c$ which are consistent with L. Note that for each input length n, there are only polynomially many machines of the appropriate size $\le c \log n + c$. Hence polynomial advice suffices to encode the whole list $M_1, \ldots, M_m$. On input x, the machine M simulates each M_i on x for at most $p(|x|)$ steps. If any of the M_i accepts, then M accepts as well, otherwise it rejects.

We claim, that $L(M) = L$. For, if $x \in L$, then there is a nondeterministic L-consistent Turing machine M_i such that $M_i(x)$ accepts and $|M_i| \le c \log |x| + c$. Thus, also $M(x)$ accepts. If, on the other hand, M accepts x, then so does some M_i which is consistent with L. Therefore, $x \in L$ because $L(M_i) \subseteq L$. □

In fact, the inclusions in Proposition 3 are proper as we will show in Theorem 5 below. For the proof we need the following notion:

Definition 4 (Buhrman, Fortnow, Laplante [8]). *For a time bound t, the* nondeterministic decision complexity of x, *denoted* $CND^t(x)$, *is the minimal size of a t-time-bounded nondeterministic Turing machine M with $L(M) = \{x\}$.*

As already noted in [4], the *CND* measure provides an upper bound to the *nic* measure, i.e., for any language L and time bound t there is a constant $c > 0$ such that $nic^t(x : L) \leq CND^t(x) + c$ for all $x \in \Sigma^*$. By a simple counting argument, it follows that for any length n there exist strings x of length n with $CND(x) \geq n$, where $CND(x)$ is the minimal size of a nondeterministic Turing machine M with $L(M) = \{x\}$ (i.e., the time-unbounded *CND* measure).

Inspired by a similar result in [3], we now prove the following separations:

Theorem 5

1. *For every constant $c > 0$, $\mathsf{NP}/n^c \not\supseteq \mathsf{NIC}[\log, \mathrm{poly}]$.*
2. *$\mathsf{NIC}[\log, \mathrm{poly}] \not\supseteq \mathsf{P}/\mathrm{lin}$.*

Proof. For the first item, let $0 < c < d$ be natural numbers. Diagonalizing against all NP machines and all advice strings, we inductively define a set A with $A \in \mathsf{NIC}[\log, \mathrm{poly}]$, but $A \notin \mathsf{NP}/n^c$. Let $(N_i)_{i\in\mathbb{N}}$ be an enumeration of all NP machines, in which every machine occurs infinitely often. In step n we diagonalize against the machine N_n and every advice string of length $\leq n^c$ which N_n might use for length n. Let $x_1, \ldots, x_{2^n}$ be the lexicographic enumeration of all strings in Σ^n and let $S_n = \{x_1, \ldots, x_{n^d}\} \subseteq \Sigma^n$. For each string w of length at most n^c, let $A_w = \{x \in S_n : N_n(x) \text{ accepts under advice } w\}$. Since there are only 2^{n^c} such sets, but 2^{n^d} subsets of S_n, there must be one which is not equal to any A_w. For every n, let A_n be one such set, and let $A = \bigcup_n A_n$. By construction, $A \notin \mathsf{NP}/n^c$.

We still have to show $A \in \mathsf{NIC}[\log, \mathrm{poly}]$. For each string s, let $\widetilde{s}$ be the substring of s which has all leading zeros deleted. For each n and each $a \in A_n$, let $M_{n,\widetilde{a}}$ be the following machine: on input x, the machine $M_{n,\widetilde{a}}$ checks whether $|x| = n$ and $\widetilde{x} = \widetilde{a}$. If this test is positive, then $M_{n,\widetilde{a}}$ accepts, otherwise it rejects. The machine $M_{n,\widetilde{a}}$ is of size $O(\log n)$, as both n and $\widetilde{a}$ are of length $O(\log n)$ (Observe that the first n^d elements in the lexicographic order of Σ^n have no 1's appearing before the last $\log n^d$ bits). Thus $A \in \mathsf{NIC}[\log, \mathrm{poly}]$.

For the second item, let A be a set that contains exactly one element x per length with $CND(x) \geq |x|$. Obviously, $A \in \mathsf{P}/\mathrm{lin}$ because A contains exactly one string per length and this element can be given as advice. On the other hand, $A \notin \mathsf{NIC}[\log, \mathrm{poly}]$. Assume on the contrary, that $A \in \mathsf{NIC}[\log, \mathrm{poly}]$. Then there are a constant c and a polynomial p, such that for each $x \in A$, there is an A-consistent p-time-bounded machine M_x of size $\leq c \log |x| + c$ which accepts x. We modify M_x to a machine M'_x such that $L(M'_x) = \{x\}$ and $|M'_x| \leq c' \log |x| + c'$ for some constant c'. This machine M'_x works as follows: on input y, the machine M'_x first checks, whether $|y| = |x|$. If not, it rejects. Otherwise, it simulates $M_x(y)$. Thus for all $x \in A$, $CND(x) \leq c' \log |x| + c'$, contradicting the choice of A. □

From Theorem 5 we infer that both inclusions in Proposition 3 are strict:

Corollary 6. $\mathsf{NP}/\log \subsetneq \mathsf{NIC}[\log, \mathrm{poly}] \subsetneq \mathsf{NP}/\mathrm{poly}$.

4 Proof Systems with Advice

Our general model of computation for proof systems f with advice is a polynomial-time Turing transducer with several tapes: an input tape containing the proof π, possibly several work tapes for the computation of the machine, an output tape where we output the proven element $f(\pi)$, and an advice tape containing the advice. We start with a quite flexible definition of proof systems with advice for arbitrary languages, generalizing the notion of propositional proof systems with advice from [1] and [2].

Definition 7. *Let $k : \mathbb{N} \to \mathbb{N}$ be a function on natural numbers. We say that a proof system f for L* uses k bits of advice*, abbreviated f is a ps/k for L, if there exists an advice function $h : \mathbb{N} \to \Sigma^*$ and an advice selector function $\ell : \Sigma^* \to 1^*$ such that*

1. *ℓ is computable in polynomial time,*
2. *$f(\pi)$ is computable in polynomial time with advice $h(|\ell(\pi)|)$, i.e., for some fixed polynomial-time computable function g, $f(\pi) = g(\pi, h(|\ell(\pi)|))$, and*
3. *for all π, the length of the advice $h(|\ell(\pi)|)$ is bounded by $k(|\pi|)$.*

For a class F of functions, we denote by ps/F the class of all ps/k with $k \in F$.

We say that f *uses k bits of input advice* if ℓ has the special form $\ell(\pi) = 1^{|\pi|}$. On the other hand, in case $\ell(\pi) = 1^{|f(\pi)|}$, then f is said to *use k bits of output advice*. The latter notion is only well-defined if we assume that the length of the output $f(\pi)$ (in case $f(\pi)$ is defined) does not depend on the advice.

We note that proof systems with advice are a quite powerful concept, as for every language $L \subseteq \Sigma^*$ there exists a proof system for L with only one bit of advice. In contrast, the class of all languages for which proof systems without advice exist coincides with the class of all recursively enumerable languages.

The above definition of a proof system with advice allows a very liberal use of advice, in the sense that for each input, the advice string used is determined by the advice selector function ℓ. For $L = \mathrm{TAUT}$ this general definition coincides with our definition of propositional proof systems with advice from [2]. In [1] and [2], concrete proof systems arising from extensions of *EF* were investigated, which indeed require this general framework with respect to the advice.

In the next proposition we observe that proof systems with input advice are already as powerful as our general model of proof systems with advice.

Proposition 8. *Let $k : \mathbb{N} \to \mathbb{N}$ be a monotone function, $L \subseteq \Sigma^*$, and f be a ps/k for L. Then there exists a proof system f' for L with k bits of input advice such that f and f' are p-equivalent.*

Proof. We choose a polynomial-time computable bijective pairing function $\langle \cdot, \cdot \rangle$ on $\mathbb{N}$ such that $\langle n_1, n_2 \rangle \geq n_1 + n_2$ for all numbers n_1 and n_2. Let f be a ps/k for L with advice function h and advice selector ℓ. We define a proof system f' for L with input advice as follows: on input π' of length n the function f' first computes the two unique numbers n_1 and n_2 such that $n = \langle n_1, n_2 \rangle$. It then

interprets the first n_1 bits $\pi'_1 \dots \pi'_{n_1}$ of π' as an f-proof π and checks whether $\ell(\pi) = 1^{n_2}$. If this is the case, $f'(\pi') = f(\pi)$, otherwise f' outputs a fixed element $x_0 \in L$. Obviously, $f'(\pi')$ is computable with advice $h(|\ell(\pi)|) = h(n_2)$ whose length is bounded by $k(n_1) \leq k(n)$. This shows that f' is a ps/k for L with input advice.

The p-simulation of f by f' is computed by the function $\pi \mapsto \pi' = \pi 1^m$ where $m = \langle |\pi|, |\ell(\pi)| \rangle - |\pi|$. The converse simulation $f' \leq_p f$ is given by

$$\pi' \mapsto \begin{cases} \pi = \pi'_1 \dots \pi'_{n_1} & \text{if } |\pi'| = \langle n_1, n_2 \rangle \text{ and } \ell(\pi) = 1^{n_2} \\ \pi_0 & \text{otherwise,} \end{cases}$$

where π_0 is a fixed f-proof of x_0. □

Cook and Krajíček [1] showed, that TAUT has a $ps/1$ with input advice which p-simulates every ps/log for TAUT, where the p-simulation is computed by a polynomial-time algorithm using $O(\log n)$ bits of advice. The proof of this result easily generalizes to arbitrary languages L, thus yielding:

Theorem 9. *For every language L there exists a proof system P with 1 bit of input advice such that P simulates all ps/log for L. Moreover, P p-simulates all advice-free proof systems for L.*

Proof. Let $\langle \cdot, \dots, \cdot \rangle$ be a polynomial-time computable tupling function on Σ^* which is length injective, i.e., $|\langle x_1, \dots, x_n \rangle| = |\langle y_1, \dots, y_n \rangle|$ implies $|x_i| = |y_i|$ for $i = 1, \dots, n$. We define the proof system P as follows. P-proofs are of the form $w = \langle \pi, 1^T, 1^a, 1^m \rangle$ with $\pi, T, a \in \Sigma^*$ and $m \in \mathbb{N}$ (here 1^T and 1^a denote unary encodings of T and a, respectively).

The proof system P uses one bit $h(|w|)$ of advice, where $h(|w|) = 1$ if and only if the transducer T with advice a only outputs elements from L for inputs of length $|\pi|$. Note that by the length injectivity of $\langle \cdot, \dots, \cdot \rangle$, the advice bit can in fact refer to T, a, and $|\pi|$. Now, if $h(|w|) = 1$ and T on input π with advice a outputs y after at most m steps, then $P(w) = y$. Otherwise, $P(w)$ is undefined.

In case Q is a proof system computed by some polynomial-time transducer T without (i.e. zero bits of) advice, then Q is p-simulated by P via the polynomial-time computable function $\pi \mapsto \langle \pi, 1^T, 1^\varepsilon, 1^{p(|\pi|)} \rangle$, where p is a polynomial bound for the running time of T (and ε is the empty string). On the other hand, if T uses advice $h(|\ell(\pi)|)$ of at most logarithmic length, then Q is simulated by P via the function $\pi \mapsto \langle \pi, 1^T, 1^{h(|\ell(\pi)|)}, 1^{p(|\pi|)} \rangle$. □

In contrast, it seems unlikely that a similar result holds for output advice (cf. [2] where we investigated this problem for propositional proof systems).

5 Polynomially Bounded Proof Systems with Advice

For any language L, we now investigate the question whether L has a polynomially bounded proof systems with advice. We obtain different characterizations of this question, depending on

– whether we use input or output advice,
– which amount of advice the proof system may use, and
– the complexity of the proven language L.

We first consider proof systems with output advice. Similarly as in the classical result by Cook and Reckhow [6], we obtain the following equivalence:

Theorem 10. *Let $L \subseteq \Sigma^*$ be a language and let $k : \mathbb{N} \to \mathbb{N}$ be a function. Then L has a polynomially bounded ps/k with output advice if and only if $L \in \mathsf{NP}/k$.*

Proof. For the forward implication, let P be a polynomially bounded ps/k with output advice for L and let p be a bounding polynomial for P. We construct an NP/k machine M which uses the same advice as P and decides L. On input x, the machine M guesses a P proof w of size $\leq p(|x|)$ and checks whether $P(w) = x$. If so, M accepts, otherwise M rejects.

For the backward implication, let N be an NP/k machine deciding L with advice function h. We define a proof system P for L with k bits of output advice. Again, both P and N use the same advice. On input $\pi = \langle w, x\rangle$ the proof system P checks, whether w is an accepting computation of N on input x with advice $h(|x|)$. If so, then $P(\pi) = x$. Otherwise, $P(\pi)$ is undefined. □

Given this result, we can now concentrate on input advice. In view of Theorem 9, input advice appears to be a stronger concept than output advice (as we probably cannot expect a similar result as Theorem 9 for output advice, cf. [2] and also Corollary 14 and Proposition 18 below for further results supporting this claim). Surprisingly, the advantage of input advice seems to vanish when we allow a polynomial amount of advice.

Theorem 11. *Let $L \subseteq \Sigma^*$ be any language. Then L has a polynomially bounded ps/poly with output advice if and only if L has a polynomially bounded ps/poly with input advice.*

Proof. The forward direction is a simple application of Proposition 8.

For the backward implication, let f_{in} be a ps/poly with input advice for L bounded by some polynomial p. Let a_n be the polynomially length-bounded advice used by f_{in} on inputs of length n.

We define a polynomially bounded ps/poly f_{out} for L with output advice as follows. Inputs x for f_{out} are interpreted as pairs $x = \langle \pi, y\rangle$. If $|\pi| \leq p(|y|)$ and $f_{in}(\pi) = y$, then $f_{out}(x) = y$. Otherwise, f_{out} is undefined. The computation of f_{out} uses all advice strings for f_{in} up to length $p(|y|)$ as advice. This still results in polynomial-size output advice for f_{out}.

The system f_{out} is correct, because f_{in} is correct. It is complete, because every $y \in L$ has a proof π_y with $|\pi_y| \leq p(|y|)$, implying that $f_{out}(\langle \pi_y, y\rangle) = y$. Hence, f_{out} is a polynomially bounded ps/poly with output advice. □

By Theorems 10 and 11, the existence of polynomially bounded ps/poly with input advice for L is equivalent to $L \in \mathsf{NP}/\mathsf{poly}$. Next, we consider proof systems with only a logarithmic amount of advice. In this case, we get a similar equivalence as before, where the class $\mathsf{NP}/\mathsf{poly}$ is replaced by the instance complexity class $\mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$.

Theorem 12. *For every language L the following conditions are equivalent:*

1. *L has a polynomially bounded $ps/1$ with input advice.*
2. *L has a polynomially bounded ps/log with input advice.*
3. *$L \in \mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$.*

Proof. The implication 1 ⇒ 2 follows by definition.

To prove the implication 2 ⇒ 3, let f be a polynomially bounded ps/log with input advice and bounding polynomial p. For each x we have to construct a program M which is consistent with L and correctly decides x. If $x \notin L$, then M can just always reject. If $x \in L$, then there exists an f-proof π of x of length $\leq p(|x|)$. Let a be the advice for f on inputs of length $|\pi|$. To construct the machine M for x, we hardwire the values of $|x|$, $|\pi|$, and a into M. On input y the machine M checks, whether $|y| = |x|$. If not, it rejects. Otherwise M guesses an f-proof π' of length $|\pi|$ for y and verifies that $f(\pi') = y$ using the advice a. If this test is positive, then M accepts, otherwise M rejects. Clearly, M accepts exactly all elements from L of length $|x|$ which have f-proofs of length $|\pi|$. In particular, M accepts x. Additionally, M is a polynomial-time nondeterministic program of length at most $c + \log|x| + \log|\pi| + |a|$ for some constant c. Therefore $L \in \mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$.

For the remaining implication 3 ⇒ 1, let us assume that there are a polynomial p and a constant c, such that for every x, $nic^p(x : L) \leq c \cdot \log(|x|) + c$. We define a polynomially bounded $ps/1$ f for L with input advice as follows. Proofs in f take the form $\pi = \langle x, w, 1^M \rangle$, where $\langle \cdot, \ldots, \cdot \rangle$ is a polynomial-time computable and length-injective tupling function. The advice for f certifies whether or not M is a polynomial-time Turing machine that is consistent with L. If this is the case and w is an accepting computation of M on input x, then $f(\pi) = x$. Otherwise, $f(\pi)$ is undefined. Note that in the proof π we described the machine M in tally form. Together with the length-injectivity of the tupling function this allows the advice to refer to the machine M (but not to the input x which is given in binary notation).

Now, since $L \in \mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$, for every $x \in L$ there is an L-consistent Turing machine M_x with running time p which accepts x and $|M_x| \leq c \cdot \log|x| + c$. Thus every element $x \in L$ has a polynomial-size f-proof $\langle x, w, 1^{M_x} \rangle$ where w is an accepting path of $M_x(x)$. □

In fact, we can prove a more general version of the preceding theorem, where we replace polynomial upper bounds for the proof length by arbitrary upper bounds. In this way we obtain:

Theorem 13. *For any language L and any function $t : \mathbb{N} \to \mathbb{N}$, $t \in n^{\Omega(1)}$, the following conditions are equivalent:*

1. *L has an $O(t)$-bounded $ps/1$ with input advice.*
2. *L has an $O(t)$-bounded $ps/O(\log t)$ with input advice.*
3. *$L \in \mathsf{NIC}[O(\log t), O(t)]$.*

For a language L we now consider the following three assertions:

A1: L has a polynomially bounded ps/log with output advice.
A2: L has a polynomially bounded ps/log with input advice.
A3: L has a polynomially bounded ps/poly with output advice.

By our results so far, assertions A1, A2, and A3 are equivalent to the statement that L is contained in the classes $\mathsf{NP/log}$, $\mathsf{NIC[log, poly]}$, and $\mathsf{NP/poly}$, respectively. As these classes form a chain of inclusions by Proposition 3, we get the implications A1 $\Rightarrow$ A2 $\Rightarrow$ A3 for every L. Moreover, by Corollary 6, the inclusions $\mathsf{NP/log} \subsetneq \mathsf{NIC[log, poly]} \subsetneq \mathsf{NP/poly}$ are proper. Hence we obtain:

Corollary 14. *There exist languages L for which* A2 *is fulfilled, but* A1 *fails. Likewise, there exist languages L for which* A3 *is fulfilled, but* A2 *fails.*

6 Polynomially Bounded Proof Systems for TAUT

From a practical point of view, it is most interesting to investigate what precisely happens for $L = \mathrm{TAUT}$ (or more generally for problems in coNP). Even though by Corollary 6, $\mathsf{NP/log}$ and $\mathsf{NIC[log, poly]}$ are distinct, they do not differ inside coNP, as the next theorem shows.

Theorem 15. *Let $L \in \mathsf{coNP}$. Then $L \in \mathsf{NP/log}$ if and only if $L \in \mathsf{NIC[log, poly]}$. Moreover, if $L \in \mathsf{NP/log}$, then the advice can be computed in $\mathsf{FP}^{\mathsf{NP[log]}}$.*

Proof. By Proposition 3 we only have to prove the backward implication. For this let L be a language from coNP. Assuming $L \in \mathsf{NIC[log, poly]}$, there exists a polynomial p and a constant c such that $nic^p(x : L) \leq c \log |x| + c$ for all $x \in \Sigma^*$. Let Π^n be the set of all p-time bounded nondeterministic machines M with $|M| \leq c \log n + c$. Let further a_n be the number of machines from Π^n that are not consistent with $L \cap \Sigma^{\leq n}$. As the cardinality of Π^n is bounded by a polynomial in n, the length of the number a_n is logarithmic in n.

We now construct a nondeterministic Turing machine N that uses $c \log n + c + 1$ bits of advice for inputs of length n and decides L. The advice of N for input length n will be the number a_n. On input x of length n, the machine N nondeterministically chooses a_n pairwise distinct machines $M_1, \ldots, M_{a_n} \in \Pi^n$ and strings $x_1, \ldots, x_{a_n} \in \Sigma^{\leq n}$. Next, N verifies that $x_1, \ldots, x_{a_n}$ do not belong to L. As $L \in \mathsf{coNP}$, this can be done in nondeterministic polynomial time. Then N checks whether for each $i = 1, \ldots, a_n$ the machine M_i accepts the input x_i. If any of the tests so far failed, N rejects. Otherwise, if all these tests were positive, we know that every machine in $\Pi^n \setminus \{M_1, \ldots, M_{a_n}\}$ is consistent with $L \cap \Sigma^{\leq n}$. After this verification has successfully taken place, N simulates all remaining machines $M \in \Pi^n \setminus \{M_1, \ldots, M_{a_n}\}$ on input x. If one of these simulations accepts, then also N accepts x, otherwise N rejects.

Since there are only consistent machines left after a_n machines have been deleted, N never accepts any $x \notin L$. On the other hand, the assumption $L \in \mathsf{NIC[log, poly]}$ guarantees that for every $x \in L$ there is a machine in Π^n which

is consistent with L and accepts x. Therefore N correctly decides L, and thus $L \in \mathsf{NP}/\log$, as claimed.

For the additional claim in the theorem, it suffices to observe that using binary search we can compute the advice a_n with at most logarithmically many queries of the form "Do there exist at least m logarithmic-size machines which are inconsistent with $L \cap \Sigma^{\leq n}$?" As this is an NP question, the advice can be computed in $\mathsf{FP}^{\mathsf{NP}[\log]}$. □

By Theorem 11 we already know that TAUT has a polynomially bounded ps/poly with input advice if and only if it has a polynomially bounded ps/poly with output advice. As a corollary to Theorem 15 we obtain the same equivalence for logarithmic advice.

Corollary 16. TAUT *has a polynomially bounded* $ps/\log$ *with input advice if and only if* TAUT *has a polynomially bounded* $ps/\log$ *with output advice.*

Descending to constant advice, this equivalence seems to fail, as we show below. For this we use a result of Buhrman, Chang, and Fortnow [9]:

Theorem 17 (Buhrman, Chang, Fortnow [9]). *For every constant* $k \geq 1$, $\mathsf{coNP} \subseteq \mathsf{NP}/k$ *if and only if* $\mathsf{PH} \subseteq \mathsf{BH}_{2^k}$.

Using this result we conclude that the assertions of the existence of polynomially bounded proof systems with input and output advice appear to be of different strength, as otherwise the equivalence of two collapses of PH of presumably different strength follows.

Proposition 18. *Assume that* TAUT *having a polynomially bounded* $ps/1$ *with input advice implies that* TAUT *has a polynomially bounded* $ps/1$ *with output advice. Then* $\mathsf{PH} \subseteq \mathsf{BH}$ *already implies* $\mathsf{PH} \subseteq \mathsf{D}^{\mathsf{p}}$.

Proof. If the polynomial hierarchy collapses to the Boolean hierarchy, then PH in fact collapses to some level BH_k of BH. By Theorem 17, this means that $\mathsf{coNP} \subseteq \mathsf{NP}/k'$ for some constant k'. Hence by Theorem 10, TAUT has a polynomially bounded ps/k' P with output advice. By Theorem 9, this proof system P is simulated by a proof system P' which only uses 1 bit of input advice. As P is polynomially bounded, this is also true for P'. By our assumption, TAUT also has polynomially bounded $ps/1$ with output advice. By Theorem 10 this implies $\mathsf{coNP} \subseteq \mathsf{NP}/1$ and therefore $\mathsf{PH} \subseteq \mathsf{D}^{\mathsf{p}}$ by Theorem 17. □

So far we have provided different characterizations of the question whether polynomially bounded proof systems with advice exist. At this point it is natural to ask, how likely these assumptions actually are, i.e., what consequences follow from the assumption that such proof systems exist. For TAUT we obtain a series of collapse consequences of presumably different strength as shown in Table 1.

The first line in Table 1 follows from Theorems 10 and 11 and a result of Cai, Chakaravarthy, Hemaspaandra, and Ogihara [10], who have shown that $\mathsf{coNP} \subseteq \mathsf{NP}/\mathsf{poly}$ implies $\mathsf{PH} \subseteq \mathsf{S}_2^{\mathsf{NP}}$. For the second line, the distinction between input and output advice is again irrelevant (Corollary 16). Here we use a result of Arvind, Köbler, Mundhenk, and Torán [4], who showed that TAUT $\in$

Table 1. Consequences of the existence of polynomially bounded proof systems

Assumption *if* TAUT *has a polynomially bounded ...*		**Consequence** *then* PH *collapses to ...*
ps/poly	(input or output advice)	$\mathsf{S}_2^{\mathsf{NP}} \subseteq \Sigma_3^{\mathsf{P}}$
ps/log	(input or output advice)	$\mathsf{P}^{\mathsf{NP}[\log]}$
$ps/O(1)$	(input advice)	$\mathsf{P}^{\mathsf{NP}[\log]}$
$ps/O(1)$	(output advice)	$\mathsf{P}^{\mathsf{NP}[O(1)]} = \mathsf{BH}$
$ps/0$	(no advice)	NP

$\mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$ implies $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{NP}[\log]}$. Finally, the constant-advice case (lines 3 and 4), follows from Theorem 17 in conjunction with Theorems 10 and 12. In comparison, the classical Cook-Reckhow Theorem states that TAUT has an advice-free polynomially bounded proof system if and only if $\mathsf{PH} \subseteq \mathsf{NP}$ (line 5).

Acknowledgements. We thank the anonymous referees for helpful comments and detailed suggestions on how to improve this paper.

References

1. Cook, S.A., Krajíček, J.: Consequences of the provability of $NP \subseteq P/poly$. The Journal of Symbolic Logic 72(4), 1353–1371 (2007)
2. Beyersdorff, O., Müller, S.: A tight Karp-Lipton collapse result in bounded arithmetic. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 199–214. Springer, Heidelberg (2008)
3. Orponen, P., Ko, K., Schöning, U., Watanabe, O.: Instance complexity. Journal of the ACM 41(1), 96–121 (1994)
4. Arvind, V., Köbler, J., Mundhenk, M., Torán, J.: Nondeterministic instance complexity and hard-to-prove tautologies. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 314–323. Springer, Heidelberg (2000)
5. Karp, R.M., Lipton, R.J.: Some connections between nonuniform and uniform complexity classes. In: Proc. 12th ACM Symposium on Theory of Computing, pp. 302–309. ACM Press, New York (1980)
6. Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. The Journal of Symbolic Logic 44(1), 36–50 (1979)
7. Krajíček, J., Pudlák, P.: Propositional proof systems, the consistency of first order theories and the complexity of computations. The Journal of Symbolic Logic 54(3), 1063–1079 (1989)
8. Buhrman, H., Fortnow, L., Laplante, S.: Resource-bounded Kolmogorov complexity revisited. SIAM Journal on Computing 31(3), 887–905 (2001)
9. Buhrman, H., Chang, R., Fortnow, L.: One bit of advice. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 547–558. Springer, Heidelberg (2003)
10. Cai, J.Y., Chakaravarthy, V.T., Hemaspaandra, L.A., Ogihara, M.: Competing provers yield improved Karp-Lipton collapse results. Information and Computation 198(1), 1–23 (2005)

How Many Holes Can an Unbordered Partial Word Contain?*

Francine Blanchet-Sadri[1], Emily Allen[2], Cameron Byrum[3], and Robert Mercaş[4]

[1] University of North Carolina, Department of Computer Science, P.O. Box 26170, Greensboro, NC 27402–6170, USA
blanchet@uncg.edu
[2] Carnegie Mellon University, Department of Mathematical Sciences, 5032 Forbes Ave., Pittsburgh, PA 15289, USA
[3] University of Mississippi, Department of Mathematics, P.O. Box 1848, University, MS 38677, USA
[4] GRLMC, Universitat Rovira i Virgili, Plaça Imperial Tárraco, 1, Tarragona, 43005, Spain
robertmercas@gmail.com

Abstract. Partial words are sequences over a finite alphabet that may have some undefined positions, or "holes," that are denoted by $\diamond$'s. A nonempty partial word is called *bordered* if one of its proper prefixes is compatible with one of its suffixes (here $\diamond$ is compatible with every letter in the alphabet); it is called *unbordered* otherwise. In this paper, we investigate the problem of computing the maximum number of holes a partial word of a fixed length can have and still fail to be bordered.

1 Introduction

Motivated by a practical problem in gene comparison, Berstel and Boasson introduced the notion of *partial words*, or sequences over a finite alphabet that may contain some "holes" denoted by $\diamond$'s [1]. For instance, $a\diamond bca\diamond b$ is a partial word with two holes over the three-letter alphabet $\{a, b, c\}$. Several interesting combinatorial properties of partial words have been investigated, and connections have been made with problems concerning primitive sets of integers, partitions of integers and their generalizations, vertex connectivity in graphs, etc [2].

An unbordered word is a word such that none of its proper prefixes is one of its suffixes. *Unbordered partial words* were defined in [3], and two types of borders were identified: *simple* and *nonsimple*. In this paper, we investigate the maximum number of holes an unbordered partial word of length n over a k-letter alphabet can have.

* This material is based upon work supported by the National Science Foundation under Grant No. DMS–0754154. We thank the referees of a preliminary version of this paper for their very valuable comments and suggestions. A World Wide Web server interface has been established at www.uncg.edu/cmp/research/countingpwords for automated use of the program. This work was done during the fourth author's stay at the University of North Carolina at Greensboro.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 176–187, 2009.

The contents of our paper are as follows: In Section 2, we compute the maximum number of holes a nonsimply bordered partial word of length n over a k-letter alphabet may have, and show that this maximum number is constant over all integers $k \geq 2$. In Section 3, we investigate the maximum number of holes an unbordered partial word of length n over a k-letter alphabet may have, obtaining an upper bound. An exact formula for $k = 2$ and lower bounds for $k \geq 3$ are also derived. Experimental evidence suggests that for $k \geq 4$, the numbers are constant. In Section 4, we conclude with some remarks. We end this section with an overview of basic concepts of combinatorics on partial words.

Let A be a nonempty finite set of symbols called an *alphabet*. Each element $a \in A$ is called a *letter*. A *full word* u over A is a finite sequence of letters from A. A *partial word* u over A is a finite sequence of symbols from $A_\diamond = A \cup \{\diamond\}$, the alphabet A being extended with the "hole" symbol $\diamond$ (a *full word* is a partial word that does not contain the $\diamond$ symbol). We will denote by u_i the symbol at position i of the partial word u.

The *length* of a partial word u is denoted by $|u|$ and represents the total number of symbols in u. The *empty word* is the sequence of length zero and is denoted by ε. For a partial word u, the powers of u are defined recursively by $u^0 = \varepsilon$ and for $n \geq 1$, $u^n = uu^{n-1}$. The set of all words over the alphabet A is denoted by A^*, while the set of all partial words over A is denoted by $A_\diamond^*$.

If u and v are two partial words of equal length, then u is said to be *contained in* v, denoted $u \subset v$, if $u_i = v_i$, whenever $u_i \in A$. Partial words u and v are *compatible* if there exists a partial word w such that $u \subset w$ and $v \subset w$. This is denoted by $u \uparrow v$. A partial word u is a *factor* of a partial word v if there exist partial words x, y such that $v = xuy$. The factor u is *proper* if $u \neq \varepsilon$ and $u \neq v$. We say that u is a *prefix* of v if $x = \varepsilon$ and a *suffix* of v if $y = \varepsilon$.

A partial word u is called *unbordered* if no nonempty partial words x_1, x_2, v, w exist such that $u = x_1 v = w x_2$ and $x_1 \uparrow x_2$. If such nonempty words exist, then there exists a partial word x such that $x_1 \subset x$ and $x_2 \subset x$. In this case, we call u *bordered* and call x a *border* of u. A border x of u is called *minimal* if $|y| < |x|$ implies that y is not a border of u. We have two distinct types of borders: For a partial word $u = x_1 v = w x_2$ where $x_1 \subset x$ and $x_2 \subset x$, we say that x is a *simple* border if $|x| \leq |v|$, and a *nonsimple* border otherwise. A bordered partial word u is called *simply bordered* if a minimal border x exists such that $|u| \geq |2x|$. For example, the word $a\diamond b\diamond bb$ has the simple and minimal border abb and the nonsimple border $abbbb$, and thus it is simply bordered.

2 Simply Bordered Partial Words

Once the number of holes in a partial word of a fixed length reaches a certain bound, the word will have a simple border. In this section, we give a closed formula for that bound and show that it is constant over all alphabets of size at least two.

Let us first recall a result regarding bordered partial words.

Proposition 1 ([4]). *Suppose that u is a nonempty partial word that is bordered. Let x be a minimal border of u. Say $u = x_1v = wx_2$, where $x_1 \subset x$ and $x_2 \subset x$. Then (1) the partial word x is unbordered, and (2) in the case where x_1 is unbordered, $u = x_1u'x_2 \subset xu'x$ for some u'.*

It follows from Proposition 1 that if u is a full bordered word, then $x_1 = x$ is unbordered. In this case, x is the minimal border of u and $u = xu'x$. Thus, a bordered full word is always simply bordered and has a unique minimal border. Since borders for partial words are defined using containment, it is possible to have numerous borders having the same length. Thus, a partial word does not necessarily have a unique minimal border.

In [5], an open problem related to borderedness in the context of partial words was suggested by the fact that every partial word of length five that has more than two holes is simply bordered. The partial word $aa\diamond\diamond b$ shows that this bound on the number of holes for length five is tight. For length six, every partial word with more than two holes is simply bordered as well. What is the maximum number of holes $m_k(n)$ a partial word of length n over an alphabet of size k can have and still fail to be simply bordered? Some values for small n follow: $m_2(5) = 2$, $m_2(6) = 2$, $m_2(7) = 3$, $m_2(8) = 4$, $m_2(9) = 5$, $m_2(10) = 5$, $m_2(11) = 6$, $m_2(12) = 7$, $m_2(13) = 8$, $m_2(14) = 8$, and $m_2(15) = 9$. We end this section by answering this open problem.

Theorem 1. *For $k \geq 2$ and $l \geq 1$, the following equalities hold: $m_k(1) = 1$, $m_k(2l) = 2l - (\lfloor \sqrt{l} \rfloor + \left\lceil \frac{l}{\lfloor \sqrt{l} \rfloor} \right\rceil)$ and $m_k(2l+1) = m_k(2l) + 1$.*

Proof. Let A be a k-letter alphabet where $k \geq 2$, and let a, b be two distinct letters of A. We prove the lower bound by constructing a partial word $w(n)$ of length n over A with $m_k(n)$ holes, that is not simply bordered. Take $w(1) = \diamond$, and for $l \geq 1$,

$$w(2l) = (a\diamond^{\lfloor \sqrt{l} \rfloor - 1})^{\frac{l}{\lfloor \sqrt{l} \rfloor}} \diamond^{l - \lfloor \sqrt{l} \rfloor} b^{\lfloor \sqrt{l} \rfloor}$$
$$w(2l+1) = (a\diamond^{\lfloor \sqrt{l} \rfloor - 1})^{\frac{l}{\lfloor \sqrt{l} \rfloor}} \diamond^{l+1 - \lfloor \sqrt{l} \rfloor} b^{\lfloor \sqrt{l} \rfloor}$$

where a fractional power of the form $(a_0 \dots a_{i-1})^{\frac{mi+j}{i}}$ with $0 \leq j < i$ is equal to $(a_0 \dots a_{i-1})^m a_0 \dots a_{j-1}$. It is easy to check that for this construction we never have a prefix of $w(n)$ of length at most $\lfloor \frac{n}{2} \rfloor = l$ compatible with a suffix. For $n \geq 4$, this is due to the fact that no factor of length $\lfloor \sqrt{l} \rfloor + 1$ of the prefix is compatible with the suffix $\diamond b^{\lfloor \sqrt{l} \rfloor}$, since each such factor has an a among its last $\lfloor \sqrt{l} \rfloor$ positions.

Now, we prove the upper bound. Let us observe that the simply bordered words of odd length are not influenced by the middle character. Hence, this character can always be replaced by a hole so that the number of holes is maximal. Because of that we can only look at the even length case. Let us consider a partial word $w = a_0 \dots a_{2l-1}$ of length $n = 2l \geq 4$ that is not simply bordered. Obviously both a_0 and a_{2l-1} are distinct letters of the alphabet A in

order to avoid a trivial one-letter border. Note that any two factors $a_0 \dots a_{i-1}$ and $a_{2l-i} \dots a_{2l-1}$ differ in at least one position for any $0 < i \leq l$. In order to avoid having the second half of w formed only of letters, we need in the first half at least two occurrences of letters. Let us suppose that a_i is the second occurrence of a letter in the first half of w (the first occurrence is a_0, that is, a_0 and a_i are letters and between them there are only holes). This implies that $a_{2l-i} \dots a_{2l-1} \in A^*$. In other words, the suffix of length l of the word ends with a word of length i over A, since otherwise we again would get compatibility for a shorter factor. Now if we look at the prefix of length $2i$, we observe that we need a second incompatibility relation with the suffix of the same length. This implies that there exists another occurrence of a letter either in the prefix at a position j, with $j \leq 2i$, or in the suffix at position j, with $j > n - 2i$. Continuing the reasoning, and looking at the problem for the following occurrences of letters in each half, we will finally get an expression of the form $i + \frac{l}{i}$ for which we have to find the minimum value, for $0 < i \leq l$. Calculating the first derivative of $i + \frac{l}{i}$ and equating to zero, we get that $i = \sqrt{l}$. Hence the minimum number of letters (it is the number of holes that we wish to be maximized) is $\left\lfloor \sqrt{l} \right\rfloor + \left\lceil \frac{l}{\lfloor \sqrt{l} \rfloor} \right\rceil$, i.e., the number of consecutive occurrences of letters from the end of the word plus the number of occurrences of letters in the first half of the word. Furthermore, we observe that the upper bound coincides with the lower bound and the obtained computer values. □

Note that Theorem 1 implies that the equality $m_k(n) = m_2(n)$ holds for all $k \geq 2, n \geq 1$.

3 Bordered Partial Words

The previously defined concept of the maximum number of holes a "nonsimply bordered" partial word may have can be extended to an "unbordered" partial word. Let $\hat{m}_k(n)$ be the maximum number of holes a partial word of length n over a k-letter alphabet can have and still fail to be bordered. For all integers $k \geq 2$ and $n \geq 1$, the inequality

$$\hat{m}_k(n) \leq m_2(n) \tag{1}$$

holds. To see this, consider a partial word u of length n over a k-letter alphabet with more than $m_k(n)$ holes. The word u necessarily has a simple border, so u is bordered and $\hat{m}_k(n)$ cannot be greater than $m_k(n)$. The inequality then follows by Theorem 1 which implies that $m_k(n) = m_2(n)$.

We now give an interval of values for $\hat{m}_k(n)$ for $k \geq 3$ and $n \geq 1$. We start with two lemmas.

Lemma 1. *The inequality* $\hat{m}_k(st) \geq (s-1)(t-1)$ *holds for all integers* $k \geq 3$ *and* $s, t \geq 1$.

Proof. Let a, b, c be distinct letters of an alphabet of size at least three. For all integers $i, j \geq 0$, $(a\diamond^i)^j bc^i$ is an unbordered word of length $(i+1)(j+1)$. Indeed, assume that $i, j \geq 1$ (the case where $i = 0$ or $j = 0$ is similar). For a border of length l with $1 \leq l \leq i$, the prefix $a\diamond^{l-1}$ will correspond to the suffix c^l. If $i+1 \leq l \leq j(i+1)$, an a will appear within the last $(i+1)$ positions of the prefix, and the corresponding position in the suffix will be b or c. If $(i+1)j+1 \leq l \leq (i+1)j+i$, the prefix will end with $bc^{i'}$ for some $0 \leq i' < i$. Since the suffix will end with c^i, the b in the prefix will not agree with the corresponding letter of the suffix. An unbordered word of length st can be constructed as described above with $i = s-1$ and $j = t-1$ with $(s-1)(t-1)$ holes. □

Lemma 2. *The equality* $\hat{m}_k(s^2) = (s-1)^2$ *holds for all integers* $k \geq 3$ *and* $s \geq 1$.

Proof. To demonstrate the lower bound $\hat{m}_k(s^2) \geq (s-1)^2$, construct a word as in the proof of Lemma 1 with $i = j = s-1$. This unbordered word will have length s^2 and $(s-1)^2$ holes. To demonstrate the upper bound $\hat{m}_k(s^2) \leq (s-1)^2$, assume u is an unbordered word with $h = (s-1)^2$ holes. It suffices to show that the length of u is at least s^2, which is equivalent to showing that u has at least $2\sqrt{h}+1 = 2(s-1)+1 = 2s-1$ letters. Without loss of generality, assume that u begins with an a. In order for u to be constructed such that the number of holes is maximized, we can assume that the a is followed by a string of $\diamond$'s of length l and then another letter. Thus, if we look at the last $l+1$ symbols in the word u, they must all be letters different from a. Otherwise, we would be able to construct a suffix of length at most l which would be compatible with the corresponding prefix. Repeating this procedure, the next letter is going to appear at most every $l+1$ symbols. In order to maximize the number of holes in u with respect to its length, u must be constructed so each repetition of holes has the same length. Thus, if there are r repetitions of the l holes, then there are at most rl total holes, and at least $r+l+1$ letters, since there is one letter for every repetition of holes and $l+1$ letters at the end. If we minimize the function $r+l+1$ with respect to $rl = h$ we get $r+l+1 \geq \sqrt{h}+\sqrt{h}+1 = 2\sqrt{h}+1$. Hence, there are at least $2s-1$ letters, and there are $(s-1)^2$ holes. So the length of u is at least s^2, and $\hat{m}_k(s^2) \leq (s-1)^2$. □

Theorem 2. *The inequalities* $(\lfloor\sqrt{n}\rfloor - 1)^2 \leq \hat{m}_k(n) \leq (\lceil\sqrt{n}\rceil - 1)^2$ *hold for all integers* $k \geq 3$ *and* $n \geq 1$.

Proof. The fact that $\hat{m}_k(n)$ is increasing is straightforward, since creating a word of length $n+1$ that is unbordered and has as many holes as the word of length n is easily done just by adding at the end of the word of length n a letter different from the one at the first position. Obviously, the newly created word will be bordered only if the word of length n is bordered. The bounds are an immediate consequence of Lemma 2. □

We end this section by refining the upper bound (1).

Proposition 2. *For all integers $k \geq 2$ and $n \geq 2$, we have the upper bound*

$$\hat{m}_k(n) \leq \left\lfloor n - \sqrt{\frac{2k}{k-1}(n-1)} \right\rfloor$$

Proof. Consider a partial word u of length n over a k-letter alphabet. Say $u = x_1 v = w x_2$, for some partial words x_1, x_2, v and w with x_1, x_2 of length i. For u not to have a border of length i, there must exist a pair of corresponding positions from x_1, x_2 whose letters are noncompatible. Since there exist $n-1$ possible border lengths for u, there must exist at least $n-1$ such pairs of noncompatible letters for u to be unbordered.

For a given number of letters $n-h$, the maximum number of noncompatible pairs will occur when each symbol of the alphabet appears equally, which would be $\frac{n-h}{k}$ times. Thus, the maximum number of noncompatible pairs is bounded above by

$$\left(\frac{n-h}{k}\right)^2 (k-1+k-2+\cdots+1) = \left(\frac{n-h}{k}\right)^2 \left(\frac{k(k-1)}{2}\right)$$

If there are strictly less than $n-1$ noncompatible pairs of letters in u, then u is necessarily bordered. So when $n-1 > \frac{(n-h)^2(k-1)}{2k}$ holds, u will be bordered. Thus, a word with $h > n - \sqrt{\frac{2k}{k-1}(n-1)}$ holes is necessarily bordered. So we have $\hat{m}_k(n) \leq \lfloor n - \sqrt{\frac{2k}{k-1}(n-1)} \rfloor$, for all integers $k \geq 2$ and $n \geq 2$. □

3.1 A Formula for $\hat{m}_2(n)$

First, we consider the 2-letter alphabet $\{a, b\}$. For $n \geq 1$, the upper bound

$$\hat{m}_2(n) \leq \lfloor n - 2\sqrt{n-1} \rfloor \tag{2}$$

follows from Proposition 2 by letting $k = 2$ (note that the case when $n = 1$ is trivial since $\diamond$ is an unbordered word of length one with one hole). We will show that this upper bound is also a lower bound.

Proposition 3. *For all integers $i, j, k \geq 0$ where $k \leq i$, the partial word given by $(a\diamond^i)^j a\diamond^k a b^{i+1}$ is an unbordered word of length $(i+1)(j+1)+k+2$.*

Proof. Assume that $i, j \geq 1$ (the other cases are similar). Consider a prefix of length l with $1 \leq l < (i+1)$. This gives us the prefix $a\diamond^{l-1}$ and the corresponding suffix b^l. Thus, there is no border of this length. Next, consider a prefix of length l with $(i+1) \leq l < (i+1)j + k + 2$. Since an a will appear within at least the last $i+1$ letters of the prefix and the corresponding position in the suffix will be b, there cannot be a border of this length. Now, consider a prefix of length l with $(i+1)j + k + 2 \leq l \leq (i+1)(j+1) + k + 1$. The prefix ends with $ab^{i'}$ with $i' \leq i$. Since the suffix ends with b^{i+1}, the last a in the prefix does not agree with the corresponding b in the suffix. □

Proposition 4. *For all integers $n \geq 5$, we have the lower bound*

$$\hat{m}_2(n) \geq \lfloor n - 2\sqrt{n-1} \rfloor$$

Proof. First, assume there exists an integer $l \geq 2$ such that $n = l^2 + 1$. We construct the binary word $(a\diamond^{l-1})^{l-1}ab^l$ of length l^2+1 which is unbordered for all integers $l \geq 2$ by Proposition 3. This word has $(l-1)^2$ holes. Making the substitution $n = l^2 + 1$ we have

$$\lfloor n - 2\sqrt{n-1} \rfloor = \lfloor l^2 + 1 - 2\sqrt{l^2} \rfloor = l^2 + 1 - 2l = (l-1)^2$$

Thus, there exists an unbordered word of length n with $\lfloor n - 2\sqrt{n-1} \rfloor$ holes.

Now, assume n cannot be written in the form $l^2 + 1$ for any integer l. Let

$$i = \left\lfloor \frac{-1 + \sqrt{1 + 4(n-2)}}{2} \right\rfloor + 1, j = \lceil \sqrt{n} \rceil - 3, k = n - (i+1)(j+1) - 2$$

Let $u = (a\diamond^i)^j a \diamond^k ab^{i+1}$ whose length is given by $(i+1)(j+1) + k + 2$ which is equivalent to $(i+1)(j+1) + n - (i+1)(j+1) - 2 + 2 = n$. The number of holes in u is given by $ij + k = ij + n - (i+1)(j+1) - 2 = n - i - j - 3 = n - \left\lfloor \frac{-1+\sqrt{1+4(n-2)}}{2} \right\rfloor - \lceil \sqrt{n} \rceil - 1$. In order to show that u has $\lfloor n - 2\sqrt{n-1} \rfloor$ holes, it suffices to show $\left\lfloor \frac{-1+\sqrt{1+4(n-2)}}{2} \right\rfloor + \lceil \sqrt{n} \rceil + 1 = \lceil 2\sqrt{n-1} \rceil$. First we note that for any integer $n \geq 5$, there exists a unique integer $m \geq 2$ such that $(m-1)^2 < n \leq m^2$. The next four bounds will be useful:

First, for $(m-1)(m-2) + 2 \leq n < m(m-1) + 2$, we have that $0 \leq 4(m-1)(m-2) \leq 4(n-2) < 4m(m-1)$. Thus, after adding 1 to, taking the square root of, subtracting 1 from, and dividing by 2 each part of the inequality yields $m - 2 \leq \frac{-1+\sqrt{1+4(n-2)}}{2} < m - 1$, and we get $\left\lfloor \frac{-1+\sqrt{1+4(n-2)}}{2} \right\rfloor = m - 2$.

Second, for $(m-1)^2 < n \leq m^2$, we have that $(m-1) < \sqrt{n} \leq m$ and so $\lceil \sqrt{n} \rceil = m$.

Third, for $(m-1)^2 + 1 < n \leq m(m-1) + 1$, we have $0 \leq (m-1)^2 + 1 < n \leq m(m-1) + 1.25$. Thus, after subtracting 1 from, taking the square root of, and multiplying by 2 each part of the inequality, this yields $2m - 2 < \lceil 2\sqrt{n-1} \rceil \leq 2m - 1$. Hence, we have $\lceil 2\sqrt{n-1} \rceil = 2m - 1$.

Fourth, for $m(m-1)+2 \leq n \leq m^2+1$, we have $m(m-1)+1.25 < n \leq m^2+1$. Thus, after subtracting 1 from, taking the square root of, and multiplying by 2 each part of the inequality, this yields $2m - 1 < 2\sqrt{n-1} \leq 2m$. Thus, we have $\lceil 2\sqrt{n-1} \rceil = 2m$.

Now, if $(m-1)^2 + 1 < n \leq m(m-1) + 1$, then $\left\lfloor \frac{-1+\sqrt{1+4(n-2)}}{2} \right\rfloor = m - 2$, $\lceil \sqrt{n} \rceil = m$, and $\lceil 2\sqrt{n-1} \rceil = 2m - 1$. If $m(m-1) + 2 \leq n \leq m^2$, then $\left\lfloor \frac{-1+\sqrt{1+4(n-2)}}{2} \right\rfloor = m - 1$, $\lceil \sqrt{n} \rceil = m$, and $\lceil 2\sqrt{n-1} \rceil = 2m$.

Finally we claim that u is unbordered. By Proposition 3, it suffices to show that $k \leq i$. This is equivalent to demonstrating

$$n \leq 2\lceil\sqrt{n}\rceil - \left\lfloor \frac{-1+\sqrt{1+4(n-2)}}{2} \right\rfloor + \lceil\sqrt{n}\rceil \left\lfloor \frac{-1+\sqrt{1+4(n-2)}}{2} \right\rfloor - 1$$

Again, let m be the unique integer such that $(m-1)^2 < n \leq m^2$. If $(m-1)^2+1 < n \leq m(m-1)+1$, then $\left\lfloor \frac{-1+\sqrt{1+4(n-2)}}{2} \right\rfloor = m-2$ and $\lceil\sqrt{n}\rceil = m$. So we have $n \leq m(m-1)+1 = 2m-(m-2)+m(m-2)-1$. If $m(m-1)+1 < n \leq m^2$, then $\left\lfloor \frac{-1+\sqrt{1+4(n-2)}}{2} \right\rfloor = m-1$ and $\lceil\sqrt{n}\rceil = m$. We get $n \leq m^2 = 2m-(m-1)+m(m-1)-1$. □

Theorem 3. *For any integer $n \geq 1$, $\hat{m}_2(n) = \lfloor n - 2\sqrt{n-1} \rfloor$.*

Proof. For $n = 1$, the result is trivial as mentioned earlier. For $n = 2$, note that a word with at least one hole necessarily has a border of length one. An unbordered word of length two with no hole is ab, and $\hat{m}_2(2) = 0 = \lfloor 2 - 2\sqrt{1} \rfloor$. For $n = 3$, $\hat{m}_2(3) = 0 = \lfloor 3 - 2\sqrt{2} \rfloor$, and an example of an unbordered word of length three with no hole is abb. As in the case of words of length two, a word that has one hole will be bordered. For $n = 4$, we can argue similarly. Thus, we have as example $abbb$, and $\hat{m}_2(4) = 0 = \lfloor 4 - 2\sqrt{3} \rfloor$. For $n \geq 5$, the result follows from (2) and Proposition 4. □

3.2 A Lower Bound for $\hat{m}_3(n)$

Now, we consider the 3-letter alphabet $\{a, b, c\}$. For $n \geq 2$, the upper bound

$$\hat{m}_3(n) \leq \left\lfloor n - \sqrt{3(n-1)} \right\rfloor \tag{3}$$

follows from Proposition 2 by letting $k = 3$. We will give a lower bound for $\hat{m}_3(n)$.

Proposition 5. *For all integers $i, j, k \geq 0$ with $k \leq i$, the partial word given by $(a\diamond^i)^j a\diamond^k c^i b$ is an unbordered word of length $(i+1)(j+1)+k+1$.*

Proof. Assume that $i, j, k > 0$ (the other cases are similar). Consider a possible border length l with $1 \leq l \leq i+1$. This yields a prefix that begins with a and a suffix which begins in b or c, so there is no border of length l. If $i+2 \leq l \leq j(i+1)+1$, we have the letter a within the last $i+1$ positions of the prefix which will correspond with c or b in the last $i+1$ positions of the suffix, so there is no border of this length. If $j(i+1)+2 \leq l \leq j(i+1)+1+k$, we have a appearing within the last $k+1$ positions of the prefix, and since $k \leq i$, the last $k+1$ positions of the suffix are c's and b's. Finally, if $j(i+1)+k+2 \leq l \leq (j+1)(i+1)+k$, we have a prefix that ends in c and a suffix which ends in b. □

Proposition 6. *For all integers $i, j \geq 2$ and $k \geq 0$, the partial word given by $(a\diamond^i)^j(b\diamond^{i+1})^k c^i b$ is an unbordered word of length $(i+1)(j+k+1)+k$.*

Proof. Assume that $i \geq 2, j \geq 2, k \geq 1$ (the case where $k = 0$ is similar). Consider a possible border length l with $1 \leq l \leq i+1$. Our prefix will begin with a which is not equal to the corresponding b or c in the suffix. For $i+2 \leq l \leq j(i+1)$, we have a within the last $i+1$ positions of the prefix, and the last $i+1$ positions of the suffix are $c^i b$. So there is no border of length l. For $j(i+1)+1 \leq l \leq j(i+1)+k(i+2)$, we have one of the following three cases:

If our prefix ends in b, then we have $l = j(i+1)+m(i+2)+1$ for some integer m with $0 \leq m < k$. In this case, the a at position $l-1-m(i+2)-2(i+1)$ of the prefix will correspond with b at this position of the suffix. So there is no border of length l. If our prefix ends with $b\diamond^{i'}$ such that $1 \leq i' \leq i$, then our prefix contains the letter b within the last $i+1$ positions, but not at the last position. However, the suffix will have c's in all of these positions. If our prefix ends with $b\diamond^{i+1}$ so that we have $l = j(i+1)+m(i+2)$ where $1 \leq m \leq k$, then the a at position $l-m(i+2)-(i+1)$ of the prefix will correspond with b at this position of the suffix. So there is no border of length l.

Finally, consider the case where $j(i+1)+k(i+2)+1 \leq l \leq j(i+1)+k(i+2)+i$. We will have a prefix which ends with c and a suffix which ends with b, so we have no border for this length. □

Proposition 7. *For any integer $n > 9$, we have the lower bound $\hat{m}_3(n) \geq n - \lceil 2\sqrt{n+3} \rceil + 2$.*

Proof. Let $l = \lceil \sqrt{n} \rceil$, so that $l \geq 4$ and $(l-1)^2 < n \leq l^2$. To show that $\hat{m}_3(n) \geq n - \lceil 2\sqrt{n+3} \rceil + 2$ is equivalent to showing that

$$\hat{m}_3(n) \geq \begin{cases} n - 2\lceil \sqrt{n} \rceil + 1 & \text{if } l^2 - 2 \leq n \leq l^2 \\ n - 2\lceil \sqrt{n} \rceil + 2 & \text{if } l(l-1) - 2 \leq n \leq l^2 - 3 \\ n - 2\lceil \sqrt{n} \rceil + 3 & \text{if } (l-1)^2 + 1 \leq n \leq l(l-1) - 3 \end{cases}$$

We consider the following five cases, and in each case demonstrate that there exists an unbordered word of length n with the required number of holes. Note that in each of the cases we have $l = \lceil \sqrt{n} \rceil$.

First, if $(l-1)^2 + 1 \leq n \leq l(l-1) - 3$, then let $u = (a\diamond^i)^j(b\diamond^{i+1})^k c^i b$ where $i = l-2$, $j = (l-1)l - (n+1)$, and $k = n - (l-1)^2$. The length of u is $(i+1)(j+k+1)+k = (l-1)((l-1)l-(n+1)+n-(l-1)^2+1)+n-(l-1)^2 = n$. The number of holes in u is $ij+(i+1)k = (l-2)(l(l-1)-(n+1))+(l-1)(n-(l-1)^2) = n-2l+3$. By Proposition 6, to show u is unbordered it suffices to show $i, j \geq 2$. This case only holds for $l \geq 4$, so $i \geq 2$. Since $n \leq l(l-1)-3$, we have $2 \leq l(l-1)-n-1 = j$. Thus, there exists an unbordered word of length n with $n - 2\lceil \sqrt{n} \rceil + 3$ holes.

Second, if $l(l-1)-2 \leq n \leq l(l-1)$, then let $u = (a\diamond^i)^j a\diamond^k c^i b$ where $i = l-1$, $j = l-3$, and $k = n-(l-1)^2$. The length of u is $(i+1)(j+1)+k+1 = l(l-2)+n-l^2+2l-1+1 = n$, and the number of holes in u is $ij+k = (l-1)(l-3)+n-l^2+2l-1 = n-2l+2$. By Proposition 5, u is unbordered if

$k \leq i$. Since $n \leq l(l-1)$, we have $n - l^2 + 2l - 1 = k \leq l - 1 = i$. Thus, there exists an unbordered word of length n with $n - 2\lceil\sqrt{n}\rceil + 2$ holes.

Third, if $l(l-1)+1 \leq n \leq l^2-4$, then let $u = (a\diamond^i)^j(b\diamond^{i+1})^k c^i b$ where $i = l-1$, $j = l^2 - 2 - n$, and $k = n - l(l-1)$. The length of u is $(i+1)(j+k+1)+k = l(l^2-2-n+n-l^2+l+1)+n-l(l-1) = n$, and the number of holes in u is $ij+(i+1)k = (l-1)(l^2-2-n)+l(n-l(l-1)) = n-2l+2$. By Proposition 6, u is unbordered if $i, j \geq 2$. Since $n \geq 7$, it must be that $l \geq 3$, and so $i \geq 2$. Since $n \leq l^2 - 4$, we have $2 \leq l^2 - 2 - n = j$. Thus, there exists an unbordered word of length n with $n - 2\lceil\sqrt{n}\rceil + 2$ holes.

Fourth, if $n = l^2 - 3$, then let $u = (a\diamond^i)^2(b\diamond^{i+1})^k c^i b$ where $i = l-2$, $j = 2$, and $k = l-3$. The length of u is $(i+1)(j+k+1)+k = (l-1)(2+l-3+1)+l-3 = l^2-3 = n$, and the number of holes in u is $ij+(i+1)k = (l-2)(2)+(l-1)(l-3) = l^2-2l-1 = l^2-3-2l+2 = n-2l+2$. By Proposition 6, u is unbordered since $j = 2$ and $i \geq 2$, because $n \geq 7$ and $l \geq 4$. Thus, there exists an unbordered word of length n with $n - 2\lceil\sqrt{n}\rceil + 2$ holes.

Fifth, if $l^2 - 2 \leq n \leq l^2$, then let $u = (a\diamond^i)^j a\diamond^k c^i b$ where $i = l-1$, $j = l-2$, and $k = n-l(l-1)-1$. The length of u is $(i+1)(j+1)+k+1 = l(l-1)+n-l(l-1)-1+1 = n$. The number of holes in u is $ij+k = (l-1)(l-2)+n-l(l-1)-1 = n-2l+1$. By Proposition 5, u is unbordered if $k \leq i$. Since $n \leq l^2$, we have $n - l^2 + l - 1 = k \leq l - 1 = i$. Thus, there exists an unbordered word of length n with $n - 2\lceil\sqrt{n}\rceil + 1$ holes. □

Note that our upper bound and lower bound for $\hat{m}_3(n)$ are equal for $n \leq 27$. We believe that our lower bound is tight and have the following conjecture.

Conjecture 1. The equality $\hat{m}_3(n) = n - \lceil 2\sqrt{n+3}\rceil + 2$ holds for all $n \geq 6$.

3.3 A Lower Bound for $\hat{m}_4(n)$

Finally, we consider the 4-letter alphabet $\{a, b, c, d\}$. By letting $k = 4$ in Proposition 2, we have the upper bound

$$\hat{m}_4(n) \leq \left\lfloor n - \sqrt{\frac{8}{3}(n-1)} \right\rfloor \tag{4}$$

for $n \geq 2$. We will give a lower bound for $\hat{m}_4(n)$.

Proposition 8. *The partial word $a\diamond^i(b\diamond^{i+1})^j c^i d$ is an unbordered word of length $(i+2)(j+1)+i$, for all $i, j \geq 0$ and distinct letters a, b, c, d.*

Proof. We assume that $i, j \geq 1$ (the other cases are similar). Consider a border length l. If $1 \leq l \leq i+1$, then we have a prefix which begins with a and a suffix which begins with c or d. If $i+2 \leq l < (j+1)(i+2)$, then the prefix ends in either $b\diamond^{i'}$ or $b\diamond^{i+1}$, where $0 \leq i' \leq i$. If the prefix ends with $b\diamond^{i'}$, then we have the letter b appearing within the last $i+1$ positions of the prefix which will correspond with either c or d in the suffix. If the prefix ends with $b\diamond^{i+1}$, then our prefix will begin with a, and our suffix will begin with b. If $(j+1)(i+2) \leq l \leq 2i+1+j(i+2)$, then our prefix ends with the letter c while our suffix ends with the letter d. In each case, there is no border of length l. □

Proposition 9. *For integers $n \geq 7$, we have the lower bound*

$$\hat{m}_4(n) \geq \begin{cases} l(l-2) & \text{if } n = l^2 - 2 \text{ for some integer } l \\ l^2 - l - 1 & \text{if } n = l^2 + l - 2 \text{ for some integer } l \\ \hat{m}_3(n) & \text{otherwise} \end{cases}$$

Proof. First, suppose that $n = l^2 - 2$ for some integer l. Let $i = j = l - 2$. The word $a\diamond^i(b\diamond^{i+1})^j c^i d$ is unbordered by Proposition 8. The length of this word is $2i + 2 + j(i + 2) = 2(l - 2) + 2 + (l - 2)l = l^2 - 2 = n$. The number of holes in the word is $i + j(i + 1) = l - 2 + (l - 2)(l - 1) = l(l - 2)$.

Next, suppose that $n = l^2 + l - 2$ for some integer l. Now let $i = l - 2, j = l - 1$. We have the word $a\diamond^i(b\diamond^{i+1})^j c^i d$, which is unbordered by Proposition 8. The length of this word is $2i + 2 + j(i + 2) = 2(l - 2) + 2 + (l - 1)l = l^2 + l - 2 = n$. The number of holes in this word is $i + j(i + 1) = l - 2 + (l - 1)(l - 1) = l^2 - l - 1$.

For all other n, consider an unbordered word with $\hat{m}_3(n)$ holes, which is still unbordered over an alphabet of size 4. □

Note that our lower bound can be improved when $n = 24, 35, 48, 63, 80$ and 99. For instance, $\hat{m}_4(24) = 16 > \hat{m}_3(24) = 15$.

$\hat{m}_4(24) = 16$	$a\diamond^1 b\diamond^2 c\diamond^2 c\diamond^3 b\diamond^8 add$
$\hat{m}_4(35) = 25$	$a\diamond^2 a\diamond^2 b\diamond^3 c\diamond^3 c\diamond^4 b\diamond^{11} addd$
	$a\diamond^2 d\diamond^2 d\diamond^3 b\diamond^3 b\diamond^4 a\diamond^{11} cccd$
$\hat{m}_4(48) = 36$	$a\diamond^1 b\diamond^2 b\diamond^2 c\diamond^3 c\diamond^3 c\diamond^3 a\diamond^4 a\diamond^{18} bddd$
	$a\diamond^1 d\diamond^2 a\diamond^2 a\diamond^3 b\diamond^3 b\diamond^3 b\diamond^4 c\diamond^{18} dcdd$
	$a\diamond^1 d\diamond^2 d\diamond^2 b\diamond^3 b\diamond^3 b\diamond^3 a\diamond^4 a\diamond^{18} cccd$
$\hat{m}_4(63) \geq 49$	$a\diamond^2 a\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^4 c\diamond^4 a\diamond^5 c\diamond^{22} dcddd$
	$a\diamond^2 a\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 c\diamond^4 c\diamond^4 a\diamond^5 c\diamond^{22} bdddd$
	$a\diamond^2 a\diamond^3 d\diamond^3 a\diamond^3 d\diamond^3 d\diamond^4 b\diamond^4 b\diamond^5 a\diamond^{22} ccccd$
$\hat{m}_4(80) \geq 64$	$a\diamond^1 b\diamond^2 b\diamond^2 c\diamond^3 c\diamond^3 c\diamond^3 c\diamond^3 c\diamond^3 c\diamond^3 c\diamond^3 a\diamond^4 a\diamond^{34} bddd$
	$a\diamond^1 d\diamond^2 a\diamond^2 a\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^4 c\diamond^{34} dcdd$
	$a\diamond^1 d\diamond^2 d\diamond^2 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 a\diamond^4 a\diamond^{34} cccd$
$\hat{m}_4(99) \geq 81$	$a\diamond^2 a\diamond^2 b\diamond^3 c\diamond^3 c\diamond^3 c\diamond^3 c\diamond^3 c\diamond^3 c\diamond^3 c\diamond^3 c\diamond^3 c\diamond^4 b\diamond^{43} addd$
	$a\diamond^2 d\diamond^2 d\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^3 b\diamond^4 a\diamond^{43} cccd$

This leads us to the following conjecture.

Conjecture 2. The equality $\hat{m}_4(l^2 - 1) = (l - 1)^2$ holds for all $l > 2$.

4 Conclusion

The following conjecture is somehow natural, since increasing the length of a partial word by one is possible through the addition of at most one hole.

Conjecture 3. The inequalities $\hat{m}_k(n) \leq \hat{m}_k(n + 1) \leq \hat{m}_k(n) + 1$ hold for all $k \geq 2, n \geq 1$.

Proposition 10. *If Conjecture 3 holds, then for all $k \geq 3$ and $n \geq 2$:*

$$\hat{m}_k(n) \leq \min\left(\left\lfloor n - \sqrt{\frac{2k}{k-1}(n-1)} \right\rfloor, n+1-2\lfloor\sqrt{n}\rfloor\right) \tag{5}$$

Proof. Using Proposition 2 we have the first bound. Now, from Lemma 2 and Theorem 2 we have that $\hat{m}_k(\lfloor\sqrt{n}\rfloor^2) = (\lfloor\sqrt{n}\rfloor - 1)^2$. Since Conjecture 3 holds, it follows that $\hat{m}_k(n) \leq \hat{m}_k(\lfloor\sqrt{n}\rfloor^2) + n - \lfloor\sqrt{n}\rfloor^2 = (\lfloor\sqrt{n}\rfloor - 1)^2 + n - \lfloor\sqrt{n}\rfloor^2 = n + 1 - 2\lfloor\sqrt{n}\rfloor$. □

Furthermore, we notice that for $k = 4$ and $n \geq 64$, the first bound in (5) is always greater than the second one. This implies that for $n \geq 64$, $\hat{m}_4(n)$ is bounded by $n + 1 - 2\lfloor\sqrt{n}\rfloor)$. Moreover, if Conjecture 3 is true, then this bound should be increasing. It follows that the upper bound for $\hat{m}_4(l^2)$ equals the upper bound for $\hat{m}_4(l^2 - 1)$, for any $l > 2$, and, if Conjecture 2 holds, the bound is sharp in these cases.

Let us denote by $UB(n)$ the upper bound (4) if Conjecture 3 does not hold, and the upper bound $\min(\left\lfloor n - \sqrt{\frac{8}{3}(n-1)} \right\rfloor, n + 1 - 2\lfloor\sqrt{n}\rfloor)$ otherwise, and by $LB(n)$ the lower bound of Proposition 9. Computer obtained data on all unbordered words of length $n \leq 1,000,000$ shows that $LB(n)$ differs from $UB(n)$ by at most 1 when Conjecture 3 holds. For instance, if $n \leq 10,000$, $LB(n)$ and $UB(n)$ differ for $5,131 + 93$ lengths (the 93 disappearing when Conjecture 2 holds). Note that the percentage of mismatches between $LB(n)$ and $UB(n)$ is actually decreasing, the more words we consider.

	$N = 10,000$	$N = 100,000$	$N = 1,000,000$
$\max_{n=1}^{N}(UB(n) - LB(n)) =$	38	114	366
If Conjecture 3 holds, then $\sum_{n=1}^{N}(UB(n) - LB(n)) =$	5,224	50,692	502,474
If Conjectures 2 and 3 hold, then $\sum_{n=1}^{N}(UB(n) - LB(n)) =$	5,131	50,383	501,481

Conjecture 4. The equality $\hat{m}_k(n) = \hat{m}_4(n)$ holds for all $k \geq 4, n \geq 1$.

References

1. Berstel, J., Boasson, L.: Partial words and a theorem of Fine and Wilf. Theoretical Computer Science 218, 135–141 (1999)
2. Blanchet-Sadri, F.: Algorithmic Combinatorics on Partial Words. Chapman & Hall/CRC Press (2007)
3. Blanchet-Sadri, F.: Primitive partial words. Discrete Applied Mathematics 148, 195–213 (2005)
4. Blanchet-Sadri, F., Davis, C., Dodge, J., Mercaş, R., Moorefield, M.: Unbordered partial words. Discrete Applied Mathematics (2008), doi:10.1016/j.dam.2008.04.004
5. Blanchet-Sadri, F.: Open problems on partial words. In: Bel-Enguix, G., Jiménez-López, M., Martín-Vide, C. (eds.) New Developments in Formal Languages and Applications, pp. 11–58. Springer, Berlin (2007)

An Answer to a Conjecture on Overlaps in Partial Words Using Periodicity Algorithms*

Francine Blanchet-Sadri[1], Robert Mercaş[2], Abraham Rashin[3], and Elara Willett[4]

[1] Department of Computer Science, University of North Carolina, P.O. Box 26170, Greensboro, NC 27402–6170, USA
blanchet@uncg.edu
[2] GRLMC, Universitat Rovira i Virgili, Plaça Imperial Tárraco, 1, Tarragona, 43005, Spain
robertmercas@gmail.com
[3] Department of Mathematics, Rutgers University, 110 Frelinghuysen Rd., Piscataway, NJ 08854–8019, USA
[4] Department of Mathematics, Oberlin College, 10 North Professor St., King 205, Oberlin, OH 44074-1019, USA

Abstract. We propose an algorithm that given as input a full word w of length n, and positive integers p and d, outputs (if any exists) a maximal p-periodic partial word contained in w with the property that no two holes are within distance d. Our algorithm runs in $O(nd)$ time and is used for the study of freeness of partial words. Furthermore, we construct an infinite word over a five-letter alphabet that is overlap-free even after the insertion of an arbitrary number of holes, answering affirmatively a conjecture from Blanchet-Sadri, Mercaş, and Scott.

1 Introduction

In [1], Manea and Mercaş extend the concept of repetition-freeness of full words to partial words which are sequences over a finite alphabet that may contain some "do not know" symbols called "holes." There, several problems regarding cube-freeness are investigated. Following the same lines, in [2], Blanchet-Sadri, Mercaş and Scott consider the concepts of square- and overlap-freeness. In these papers, the authors investigate the existence of infinite full words into which arbitrarily many holes can be inserted without introducing repetitions (inserting a hole is defined as replacing a letter with a hole in a fixed position of a word, the length of the word remaining the same). A constraint that no two holes can be placed one or two positions apart is needed, to avoid trivial squares and cubes. This

* This material is based upon work supported by the National Science Foundation under Grant No. DMS–0754154. We thank the referees of a preliminary version of this paper for their very valuable comments and suggestions. This work was done during the second author's stay at the University of North Carolina at Greensboro. A World Wide Web server interface has been established at www.uncg.edu/cmp/research/freeness2 for automated use of the program.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 188–199, 2009.

problem is equivalent to determining whether an infinite partial word w can be found such that, none of its factors of length kp, for any positive integer p and rational $k \geq 2$, is 2-valid and p-periodic. A partial word is called d-valid if any positions i, j satisfying $0 < |i - j| \leq d$ are not both holes.

A well-known result of Thue states that over a binary alphabet there exist infinitely many overlap-free infinite full words [3,4]. In [1], the question was raised as to whether there exist overlap-free infinite partial words, and to construct them over a binary alphabet if such exist. In [2] and [5], the authors settle this question by showing that over a two-letter alphabet there exist overlap-free infinite partial words with one hole, and none exists with more than one hole. Moreover, in [2] it is shown that there exist infinitely many overlap-free infinite partial words with an arbitrary number of holes over a three-letter alphabet. There, it is also proved that there exists an infinite overlap-free word over a six-letter alphabet that remains overlap-free after an arbitrary selection of its letters are changed to holes, and none exists over a four-letter alphabet. The case of a five-letter alphabet remained open.

Conjecture 1 ([2]). There exists an infinite word over a five-letter alphabet that remains overlap-free after an arbitrary 2-valid insertion of holes.

In this paper, we investigate the question of which finite full words can be made periodic by insertion of holes, with the restriction that no two holes be too close together. More precisely, we present an algorithm that determines whether a finite full word w of length n contains a d-valid p-periodic partial word, in $O(nd)$ time. As a consequence, we give a positive answer to Conjecture 1. An overview of basic concepts of combinatorics on partial words follows.

Let A be a nonempty finite set of symbols called an *alphabet*. Each element $a \in A$ is called a *letter*. A *full word* over A is a finite sequence of letters from A. A *partial word* over A is a finite sequence of letters from $A_\diamond = A \cup \{\diamond\}$, the alphabet A extended with the hole symbol $\diamond$ (a full word is a partial word that does not contain the $\diamond$ symbol). A partial word u of length n over A can be viewed as a function $u : \{0, \ldots, n-1\} \to A_\diamond$. The *length* of a partial word u is denoted by $|u|$ and represents the total number of symbols in u. The *empty word* is the sequence of length zero and is denoted by ε. The powers of a partial word u are defined recursively by $u^0 = \varepsilon$ and for $n \geq 1$, $u^n = uu^{n-1}$. The set containing all finite full words over the alphabet A is denoted by A^*, while the set of all finite partial words over the alphabet A is denoted by $A^*_\diamond$.

A *strong period* of a partial word u over A is a positive integer p such that $u(i) = u(j)$ whenever $u(i), u(j) \in A$ and $i \equiv j \ mod\ p$. In such a case, we say u is *strongly p-periodic*. A *weak period* of u is a positive integer p such that $u(i) = u(i + p)$ whenever $u(i), u(i + p) \in A$. In such a case, we say u is *weakly p-periodic*. The word $abba\diamond bacb$ is weakly 3-periodic, but not strongly 3-periodic.

If u and v are two partial words of equal length, then u is said to be *contained* in v, denoted $u \subset v$, if $u(i) = v(i)$, for all $u(i) \in A$. Partial words u and v are *compatible*, denoted by $u \uparrow v$, if there exists w such that $u \subset w$ and $v \subset w$. A partial word u is a *factor* of a partial word v if $v = xuy$ for some x, y. The factor u is *proper* if $u \neq \varepsilon$ and $u \neq v$. We say that u is a *prefix* of v if $x = \varepsilon$

and a *suffix* of v if $y = \varepsilon$. A full word u is said to contain an overlap if it contains a factor $avava$ (two overlapping occurrences of the word ava) with a a letter and v a word [6]. In [1], this definition was extended to partial words, an overlap being considered a factor $a_0v_0a_1v_1a_2$ with $v_0 \uparrow v_1$, and a_0, a_1, a_2 pairwise compatible letters ($a_0v_0a_1v_1a_2 \subset avava$, for some letter a and word v), and it can be generalized as follows: A partial word $a_0v_0a_1v_1a_2$, where $v_0 \uparrow v_1$, is a *weak overlap* if $a_0 \uparrow a_1$ and $a_1 \uparrow a_2$ (because $a_0v_0a_1 \uparrow a_1v_1a_2$), and a *strong overlap* if a_0, a_1, a_2 are pairwise compatible symbols. Note that a strong overlap is also a weak overlap. A partial word is *weakly overlap-free* (respectively, *strongly overlap-free*) if none of its factors is a weak (respectively, strong) overlap.

2 Periodic Partial Words with No Two Holes within a Fixed Distance

We say two positions i, j in a partial word u are *d-proximal* if $0 < |j - i| \leq d$, where d denotes a positive integer. We say that u obeys the *hole constraint* d (or u is *d-valid*) if no two holes in u are d-proximal. When the value of d is clear from context, we may suppress reference to it, simply referring to the "hole constraint" or to "proximal" positions.

Let w be a length n full word defined over an alphabet A of size k. In this section, we present an $O(nd)$ time algorithm, which finds, for given positive integers d and p both less than n, a *d-valid* p-periodic partial word contained in w, if any exists. In other words, it determines whether it is possible to insert holes into w with no two holes within distance d, such that the resulting partial word has strong period p. If this is possible, such a word is returned.

In order to work with words of length n more easily, we write them in rows of length p. For a partial word u and for an integer x, $0 \leq x < p$, we will call *column* x the sequence of positions (or letters at these positions) $x, x + p, \ldots, x + lp$, where l is the maximal integer such that $x + lp < n$. For example in Figure 1, if $w = abadecabbdeeaba$, $p = 6$, and $d = 2$, then $u = abadecab\diamond de\diamond aba$ is obtained using our algorithm. For an integer x, $0 \leq x < p$, let $S_x = \{w(i) \mid 0 \leq i < n, i \equiv x \bmod p\}$ be the set of distinct letters appearing in column x of w. We construct a new set of partial words $\Omega = \{\omega \mid \omega(i) \in S_i\}$, and call $u \subset w$ a partial word *induced* by the choice $\omega \in S_0 \times S_1 \times \cdots \times S_{p-1}$, if $u \subset \omega^l$, for some rational l. Now, u, induced by ω, is d-valid if and only if for any two proximal positions i and j ($0 \leq i, j < n$, $0 < |i - j| \leq d$), it is not the case that $u(i) = \diamond = u(j)$.

Remark 1. The choice of letters $\omega \in \prod_{x=0}^{p-1} S_x$ induces a d-valid word if and only if for every two proximal positions i, j, $u(i) = \omega(i \bmod p)$ or $u(j) = \omega(j \bmod p)$.

a	b	a	d	e	c			a	b	a	d	e	c		
a	b	b	d	e	e			a	b	$\diamond$	d	e	$\diamond$		
a	b	a						a	b	a					

Fig. 1. The words w and u with columns 2 and 5 highlighted

This suggests a geometric approach for determining which choices of letters do not cause a hole constraint violation. For $(a_0, b_0) \in A^2$, let the *cross centered at* (a_0, b_0) be the set $+(a_0, b_0) = \{(a, b) \in A^2 \mid a = a_0 \text{ or } b = b_0\}$. Then, the choices of letters a for column x and b for column y that do not cause any hole constraint violations, are precisely those in the intersection of the crosses centered at $(w(i), w(j))$, for i, j proximal positions in columns x, y.

The subsets of A^2 formed by intersecting crosses, however, are of special forms. The following theorem describes these forms, and shows that they can be determined in l-linear time, where l is the number of crosses that need to be intersected (the number of distinct ordered pairs (i, j) where i, j are proximal positions in columns x, y, respectively).

Theorem 1. *Considered as a set of entries in a $k \times k$ matrix, any set T formed by intersecting crosses must be either: (1) FULL: A^2; (2) CROSS(a_0, b_0): a cross $+(a_0, b_0)$; (3) ROW(a_0): a row of the matrix $\{(a_0, b) \mid b \in A\}$; (4) COL$(b_0)$: a column of the matrix $\{(a, b_0) \mid a \in A\}$; (5) TWO$((a_1, b_1), (a_2, b_2))$: a set of two points (a_1, b_1) and (a_2, b_2) in neither the same row nor column; (6) ONE(a_0, b_0): a singleton set $\{(a_0, b_0)\}$; or (7) NULL: the null set $\emptyset$.*

Proof. Let m be the number of crosses that are intersected: $T = \bigcap_{s=1}^{m} +(a_s, b_s)$. If $m = 0$, then $T = A^2$ is FULL. The form FULL is only possible for $m = 0$. If $m = 1$, then $T = +(a_1, b_1)$ is CROSS(a_1, b_1). Now suppose that $m > 1$ and let $T' = \bigcap_{s=1}^{m-1} +(a_s, b_s)$. We consider what happens when we intersect $+(a_m, b_m)$ with T', for T' in each of the above forms.

Let $T' = \text{CROSS}(a_0, b_0)$. If $(a_m, b_m) = (a_0, b_0)$, then $T' = +(a_m, b_m)$, so $T = T'$. If $a_m = a_0$ and $b_m \neq b_0$, then $T = \text{ROW}(a_0)$. If $b_m = b_0$ and $a_m \neq a_0$, then $T = \text{COL}(b_0)$. If $a_m \neq a_0$ and $b_m \neq b_0$, then $T = \text{TWO}((a_0, b_m), (a_m, b_0))$. Therefore, intersecting $+(a_m, b_m)$ with a CROSS matrix results in a CROSS, ROW, COL, or TWO matrix, as depicted in Figure 2. a). If $T' = \text{ROW}(a_0)$ and $a_m = a_0$, then $T = T' \subset +(a_m, b_m)$. Otherwise, $T = \text{ONE}(a_0, b_m)$. If $T' = \text{COL}(b_0)$ and $b_m = b_0$, then $T = T' \subset +(a_m, b_m)$. Otherwise, $T = \text{ONE}(a_m, b_0)$.

Now, let $T' = \text{TWO}((a, b), (a', b'))$. If (a_m, b_m) is equal to (a, b') or to (a', b), then $T = T' \subset +(a_m, b_m)$. Now, if $a = a_m$ or $b = b_m$ then $(a, b) \in +(a_m, b_m)$,

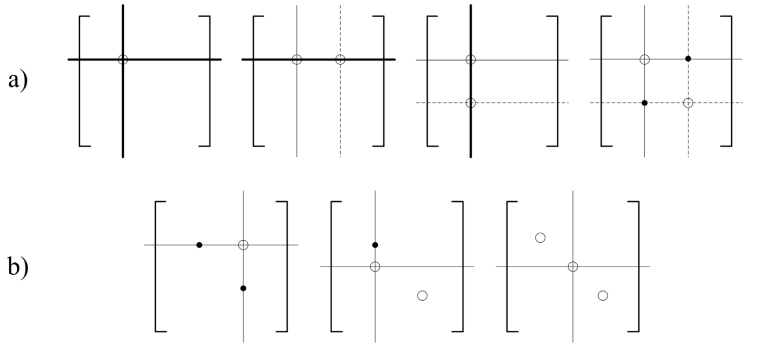


Fig. 2. Intersection of different matrices

but $(a', b') \notin +(a_m, b_m)$, and so $T = \text{ONE}(a, b)$. Similarly, if $a' = a_m$ or $b' = b_m$ then $T = \text{ONE}(a', b')$. Finally, if $a \neq a_m$, $b \neq b_m$, $a' \neq a_m$ and $b' \neq b_m$, then $(a, b), (a', b') \notin +(a_m, b_m)$, so $T = \text{NULL}$. Therefore, intersecting $+(a_m, b_m)$ with a TWO matrix results in a TWO, ONE, or NULL matrix, as depicted in Figure 2, b). If $T' = \text{ONE}(a_0, b_0)$ and $a_m = a_0$ or $b_m = b_0$, then $T' \subset +(a_m, b_m)$, so $T = T'$. Otherwise, $T = \text{NULL}$. Finally, if $T' = \text{NULL}$, then $T = \text{NULL}$. □

Now, returning to the question of which $\omega \in \prod_{x=0}^{p-1} S_x$ induce d-valid partial words, for two columns $x, y < p$, we define the constraint matrix M^{xy}, to be a $k \times k$ matrix such that, for all $a, b \in A$, $M^{xy}(a, b)$ is $*$ if for every pair of proximal positions i, j in columns x, y, $(a, b) \in +(w(i), w(j))$, and 0 otherwise. Note that, trivially, the constraint matrix from x to y is the transpose of the constraint matrix from y to x, and that $\omega \in \prod_{x=0}^{p-1} S_x$ induces a d-valid partial word if and only if for every $x, y \in \{0, \ldots, p-1\}$, $M^{xy}(\omega(x), \omega(y)) = *$.

The result of Theorem 1 is that the constraint matrices can be classified into a few simple types. Therefore, in practice, we store constraint matrices as objects that encode the form of the matrix (FULL, CROSS, TWO, etc.), and at most four characters to denote rows and columns (querying the position of stars in row a_0 of the object $< \text{TWO}, (a, b), (a', b') >$ yields b if $a_0 = a$, b' if $a_0 = a'$ and NONE otherwise). These can be constructed and read in constant time.

Remark 2. If columns x, y are proximal, that is x, y contain proximal positions, then $0 < |x - y| \leq d$ or $0 < p - |x - y| \leq d$.

Fix some variables that will be shared by the algorithms: a table of constraint matrices, M; sets F_{ROW}, F_{ONE}, F_{TWO} and F_{CROSS}, where F_{FORM} contains (x, y) for which M^{xy} is of form FORM; a list of letters ω, where $\omega(x)$ is the letter chosen for column x. The following lemmas will be useful in proving the validity of our algorithms.

Lemma 1. *If $0 \leq x, y < p$ with $0 < |x - y| \leq d$, then M^{xy} is not FULL.*

Proof. The positions x and y in w are proximal since $0 < |x - y| \leq d$. Therefore at least one cross (namely, that centered at $(w(x), w(y))$ is used in the creation of the matrix M^{xy}, so it cannot be FULL. □

Furthermore, it follows from Theorem 1 that the types of constraints that one column can exert on another are limited.

Lemma 2. *If two columns x, y with $0 \leq x < y < p$, contain each at least two different letters, and M^{xy} is a CROSS matrix, then $|x - y| \geq \max\{p - d, d + 1\}$.*

Proof. Since M^{xy} is not a FULL matrix, by Remark 2, we have that $|x - y| \leq d$ or that $p - d \leq |x - y|$. Suppose that $|x - y| \leq d$, and let $y + sp$ be a position in column y, where $y \leq y + sp < n$. Thus, $x + sp$ is a position in column x, since $0 \leq x \leq x + sp < y + sp < n$. Furthermore, every position in column y is proximal to some position in column x. Since M^{xy} is a CROSS matrix, all ordered pairs $(w(i), w(j))$, for i, j proximal positions in columns x, y, must be equal. Therefore all letters in column y of w are equal, a contradiction. Therefore $|x - y| > d$ and $|x - y| \geq p - d$, so $|x - y| \geq \max\{p - d, d + 1\}$. □

There exist even more restrictions regarding CROSS matrices.

Algorithm 1. Initalizing the matrices

```
1: for (x, y) columns within d do
2:     M^{xy} := FULL
3: for i = 0 to n − y step p do
4:     intersect M^{xy} with cross centered at (w(x + ip), w(y + ip))
```

Lemma 3. *Let x_1, x_2, x_3 be distinct columns with at least two different letters each. If $M^{x_2x_3}$ and $M^{x_1x_3}$ are CROSS matrices, then $M^{x_1x_2}$ is neither a FULL nor a CROSS matrix.*

Henceforth, by *columns within d* we mean columns x, y such that $0 < |x-y| \leq d$ or $0 < p - |x - y| \leq d$. Any other pair of columns is necessarily related by a FULL constraint matrix and therefore can be ignored. Algorithm 1 computes all non-FULL constraint matrices of w in $O(nd)$ time.

Corollary 1. *The forms (as per Theorem 1) of all the non-FULL constraint matrices for w can be determined in $O(nd)$ time via Algorithm 1.*

Note that given two proximal columns x and y, and a letter a chosen for column x, there are either zero, one, or $\|S_y\|$ choices of a letter for column y that do not conflict with the choice of letter a for column x. This observation suggests an algorithm for labeling multiple columns. Let us now construct a directed graph G that has vertex set $\{0, \ldots, p-1\}$ and edge set consisting of edges (x, y) labelled by M^{xy} when columns x, y are within d.

Theorem 2. *For a column x and a letter $a \in S_x$, Algorithm 2 correctly chooses letters for some additional columns such that, after the completion of this algorithm no undetermined column is constrained by an already determined column. Additionally, if the constraint matrices have already been computed, the running-time of Algorithm 2 is $O(m)$, where m is the number of edges that are traversed.*

Proof. The problem of finding a choice of letters for the columns is equivalent to finding a labeling of the vertices of G, such that every vertex x is labeled with a letter $\omega(x)$ that occurs in column x of w, and for any two columns x and y within d, the $(\omega(x), \omega(y))$-entry of M^{xy} is a $*$. If such a labeling exists, then it induces a p-periodic d-valid partial word contained in w, by replacing every non-$\omega(x)$ letter in each column x with a hole.

The algorithm starts by assuming a labeling of vertex x by the letter a, and then performs a breadth-first search on the graph G, starting at x. This is implemented using a queue. Suppose that a vertex y has been marked by letter b and that we are now traversing an edge from y to z. Then the constraint matrix M^{yz} either uniquely determines the label c on the sink vertex z, or it imposes no constraint at all, or there are no choices, in which case the algorithm immediately fails. In the former case, either the unique label is applied ($\omega(z)$ is set to c) and vertex z is added to the queue for later traversal, or if $\omega(z)$ has already been set to a different value, the algorithm fails because there cannot be any labeling of G with $\omega(x) = a$ and $\omega(z)$ with its original value. In the case when no constraint

Algorithm 2. Fill(x, a)

```
 1: initialize Q to be an empty queue accepting columns
 2: choose letter a for column x
 3: add x to Q
 4: while dequeue y from Q do
 5:    let b = ω(y)
 6:    for z a neighbor of y do
 7:       let row be the b row of the matrix M^{yz}
 8:       remove edges between y and z
 9:       if row has all ∗'s then
10:          next (go to line 4)
11:       else if row has exactly one ∗, say at position c then
12:          if letter c has already been chosen for column z then
13:             next (go to line 4)
14:          else if column z is unlabeled then
15:             choose letter c for column z
16:             add z to Q
17:             next (go to line 4)
18:       undo all recent labelings and edge erasures
19:       return false
20: return true
```

is imposed (the b row is filled with $*$'s), this matrix is ignored, since for any value of $\omega(z)$ the matrix will not cause a contradiction. In all cases, the edge (y, z) and its opposite (z, y) are marked as having been traversed, so that they will not be visited again. In conclusion, an undetermined column is marked exactly when it is constrained by an already determined column, thus, ensuring that at the end of the algorithm no determined column will constrain an undetermined column. This algorithm visits m edges, no more than once each. On each edge, it performs a constant time operation. Thus, Algorithm 2 runs in $O(m)$ time.

Please note that undoing all recent labelings and edge erasures, while keeping the algorithm's runtime within $O(m)$, is solved in constant time by implementing data structures that could be "marked" in a particular state, and reset to this state later on. These data structures are used for the sets of neighbors of a vertex, the sets F_{FORM} (of edges of each type), and the set of labeled vertices. While, all the F_{FORM}'s and labelings can be reset in constant time, the vertex neighbor sets can be reset in $O(l)$ time, where l is the number of vertices visited during this run of the algorithm. Since the number of vertices visited is less than the number of edges visited, $l < m$, the overall algorithm runs in $O(m)$ time. □

The next lemma will help us prove that we never need to run Algorithm 2 ("Fill(x, a)") on a vertex more than twice.

Lemma 4. *Suppose that x and y are vertices of G such that $M^{xy} = TWO((a, b), (a', b'))$, Fill$(x, a)$ returns true, and $\omega \in \prod_{z=0}^{p-1} S_z$ induces a d-valid partial word u with $\omega(x) = a'$. Then, there exists a choice ω' of letters for the columns, that induces a d-valid partial word with $\omega'(x) = a$.*

Algorithm 3. Traversing the entire graph

```
initialize matrices
for (x, y) columns within d do
    if M^{xy} = NULL then
        return false
    add (x, y) to F_FORM
for column x do
    if ||S_x|| = 1 then
        Fill(x, w(x))
while exists (x, y) with M^{xy} of form ROW(a), in F_ROW do
    if not Fill(x, a) then return false
while exists (x, y) with M^{xy} of form ONE(a, b), in F_ONE do
    if not Fill(x, a) then return false
while exists (x, y) with M^{xy} of form TWO((a, b), (a', b')), in F_TWO do
    if not Fill(x, a) and not Fill(x, a') then return false
for column x do
    if column x is unlabeled then
        choose w(x) for column x
for i from 0 to n − 1 do
    let u(i) be w(i) if w(i) = ω(i mod p) and ⋄ otherwise
return u
```

Proof. Let T be the set of vertices of G that are labeled by Fill(x, a), and Q be the labeling of T. For every vertex x of G, let $\omega'(x) = Q(x)$ if $x \in T$ and $\omega'(x) = \omega(x)$ otherwise. Since the labeling Q of T was generated by Fill(x, a), we know that no letter choice for a vertex outside T is constrained by any of the letter choices specified in Q. Furthermore, since ω induced a d-valid partial word, we know that no constraint matrix is violated by two letter choices in ω. Therefore the letter choices in ω' do not violate any constraint matrices, so ω' induces a d-valid partial word. Also, clearly $\omega'(x) = a$, so we have our result. □

The next algorithm traverses all edges corresponding to non-FULL matrices and finds a consistent labeling of the vertices of G if any exists.

Theorem 3. *Algorithm 3 returns a d-valid p-periodic partial word contained in w, unless no such word exists. The running-time of the algorithm is $O(nd)$.*

Proof. If there is a NULL matrix between two columns, then no consistent labeling of the vertices exists, so the algorithm fails. If any column in w has all letters equal, then that letter must be assigned for the column, and Fill$(x, w(x))$ ran. There can only be one consistent labeling of all vertices if it succeeds (note that the determination of whether a column has only one character can be performed in $O(\frac{n}{p})$ time, and thus, it can be performed for all columns in $O(n)$ time). Similarly, if there is a ROW or ONE matrix M^{xy} with a $*$ in row a, then a must be chosen for column x. We run Fill(x, a), and it must succeed for there to be a consistent labeling of the vertices of G.

If $M^{xy} = \text{TWO}((a, b), (a', b'))$ then we know that any consistent labeling of the vertices of G must have column x labeled with either a or a'. But by

Lemma 4, if some consistent labeling of G exists and Fill(x, a) returns true, then there exists a consistent labeling of G that agrees on all choices of letters made by Fill(x, a). Therefore in this case we can simply continue. Otherwise we try Fill(x, a'). If this fails, then we return false.

At this point in the algorithm, any unlabeled vertices x, y are related by either a FULL or CROSS matrix, since all other types of matrices have already been taken into account. Consider a graph T' with the so-far unlabeled vertices of G as the vertex set, and an edge between x and y if and only if M^{xy} is a CROSS matrix. We can satisfy all remaining constraints (the CROSS matrices) by considering every connected component of T' separately. But, by Lemma 3, this graph has no connected components of size greater than two (since only crosses are left, connecting more than two of them falls in Lemma 3).

We claim that we can label any remaining vertex x with $w(x)$ (the first letter appearing in column x) without introducing any new contradictions. This is clearly true for any isolated vertex in T', since these are unconstrained. Now consider x, y vertices in T' related by CROSS(a, b). Every proximal pair of positions i, j in columns x, y must have $w(i) = a$ and $w(j) = b$. But between any two columns that have proximal pairs, at least one of them has its first (top) position proximal to some position in the other column. Therefore $w(x) = a$ or $w(y) = b$ (or both). Therefore these choices satisfy the constraint matrix. If the algorithm reached this step, then there exists a p-periodic d-valid partial word contained in w, namely the one induced by ω.

Each matrix is visited at most twice (this worst case scenario is achieved precisely if the edge is examined twice in the loop starting on line 13). There are at most $2pd$ matrices in question, and analyzing a row of a matrix takes constant time. Thus, the running-time is $O(pd)$ plus the running-time of checking which columns are uniform, and of constructing the constraint matrices ($O(nd)$ by Corollary 1). Therefore, the total running-time of Algorithm 3 is $O(nd)$. □

3 Short Factors in the Images of Morphisms

Let $\beta : A^* \to A^*$ be a non-erasing prolongable morphism on $z_0 \in A$. For $m \geq 0$, let $z_{m+1} = \beta(z_m)$, and $w = \lim_{m\to\infty} z_m$ the fixed point of β. Let $F_m(y)$ denote the set of length m factors of y. Since z_m is a proper prefix of z_{m+1}, for any $n \geq 1$:

$$F_n(z_0) \subseteq F_n(z_1) \subseteq F_n(z_2) \subseteq \cdots \subseteq \bigcup_{m\geq 0} F_n(z_m) = F_n(w)$$

But since $F_n(w)$ is finite, having at most $|A|^n$ elements, using the pigeonhole principle, the chain must become constant after finitely many steps.

Lemma 5. *Let $m \geq 0, n \geq 1$ be such that $\emptyset \neq F_n(z_m) = F_n(z_{m+1})$. Then $F_n(z_m) = F_n(w)$.*

Suppose now, that $q = \min\{|\beta(a)| \mid a \in A\} \geq 2$ and $F_2(z_m) = F_2(w)$. In other words, β maps every letter of the alphabet to a word of length at least two, and

all length two factors of w are factors of z_m. For all $i \geq 0$, it can be shown that $F_{q^i+1}(z_{m+i}) = F_{q^i+1}(w)$. Moreover, let $s = \max\{|\beta(a)| \mid a \in A\}$. Then it can also be shown that the set of length n factors of w can be computed in $O(n^{\log_q s})$ time. Hence, for any $n \geq 2$, $z_{m+\lceil \log_q(n-1) \rceil}$ has all length n factors of w, and this set can be computed in polynomial time. Furthermore, if β is a uniform morphism, we have $q = s$ and $F_n(w)$ is computable in $O(n)$ time. Note that in some cases we can discard the requirement $q \geq 2$, by taking a higher iteration of the morphism (for $\mu : a \mapsto abc$, $b \mapsto ac$, $c \mapsto b$, the square $\mu^2 : a \mapsto abcacb$, $b \mapsto abcb$, $c \mapsto ac$, can be used in the above theorems, since it generates the same fixed point).

4 An Overlap-Free Word over an Alphabet of Size Five

Note that the definition of weak overlap generalizes the overlap definitions used in [2] and [5], since here a factor is considered to be an overlap of length $2p+1$ if it has p as a weak period, while in [2,5], the factor had to have a strong period p. In this section, we generate an infinite full word over a 5-letter alphabet, which remains weakly overlap-free after any 2-valid insertion of holes. First, define a morphism $\gamma : \{a, b, c, d\}^* \to \{a, b, c, d\}^*$ with $\gamma(a) = ad$, $\gamma(b) = ac$, $\gamma(c) = cb$, and $\gamma(d) = ca$. Since a is a prefix of $\gamma(a)$, γ is prolongable. Thus, we define the fixed point of γ, $\Gamma = \lim_{i \to \infty} \gamma^i(a)$. Consider some properties of Γ.

Remark 3. Both $\gamma^3(a) = adcacbad$ and $\gamma^4(a) = adcacbadcbacadca$ have only ac, ad, ba, ca, cb and dc as their length two factors. Thus, by Lemma 5, these are the only length two factors of Γ.

Lemma 6. *The infinite full word Γ is square-free.*

Proof. It suffices to show that every $\gamma^n(a)$ is square-free. Clearly $\gamma^0(a) = \varepsilon$ is square-free. Now let $n \geq 0$ and assume that $\gamma^n(a)$ is square-free. Suppose, for contradiction, that $\gamma^{n+1}(a)$ has a square factor of length $2p$ starting at position i. Since the letters b and d appear only at odd positions of $\gamma^{n+1}(a)$, hence, if p is odd, the factor would be in $\{a, c\}^*$. Since all binary words of length 4 contain squares, it must be that $p = 1$, which is a contradiction according to Remark 3 ($p = 1$ in order to avoid the contradiction of having squares).

Therefore p must be even. If i is even, since $\gamma^{n+1}(a) = \gamma(\gamma^n(a))$ it follows that $\gamma^n(a)$ contains a square, contradicting the initial assumption. Hence, i is odd. Since, $\gamma(f)$ ends in a different letter for all $f \in \{a, b, c, d\}$, it follows that we have a factor that is a square starting at position $i - 1$, which is an even position. Following the previous reasoning we again reach a contradiction. □

Now let $\delta : \{a, b, c, d\}^* \to \{f, g, h, i, j\}^*$ be a morphism defined by $\delta(a) = fgifh$, $\delta(b) = fghij$, $\delta(c) = jigjh$, and $\delta(d) = jihgf$. We claim that $\delta(\Gamma)$ is overlap-free after an arbitrary (2-valid) insertion of holes.

Proposition 1. *There are no factors of $\delta(\Gamma)$ of length ≤ 21 that can be turned into weak overlaps by any 2-valid insertion of holes.*

Proof. Using a variant of Algorithm 3, we checked that for $p \leq 10$, there is no factor of $\delta(\Gamma)$ of length $2p+1$ that contains a 2-valid weakly-p-periodic word. □

Recall the following result from [2].

Remark 4. [2] Full words $t = t_0t_1t_2$ and $s = s_0s_1s_2$ contain compatible 2-valid partial words if and only if for some i, $t_i = s_i$.

Lemma 7. *In $\delta(\Gamma)$, any two length seven sequences starting with the same character will contain at least three consecutive mismatches if they are not identical.*

Proof. According to Remark 3 the only length two factors of Γ are ac, ad, ba, ca, cb and dc. We prove the lemma for sequences starting with letter f, the other cases being similar. If a sequence starts with f, then it must be either $fgifhji$, a prefix of both $\delta(ac)$ and $\delta(ad)$, $fghijfg$, prefix of $\delta(ba)$, $fjigjhf$, first factor starting with f in both $\delta(dca)$ and $\delta(dcb)$, or, $fhjigjh$ and $fhjihgf$, suffixes of $\delta(ac)$ and $\delta(ad)$. It is easy to check that each two of these blocks contain three consecutive mismatches once aligned. □

Proposition 2. *No factor of $\delta(\Gamma)$ of length $2p + 1 > 21$ with p not divisible by 5 can be turned into a weak overlap by a 2-valid insertion of holes.*

Proof. Assume that there exists $a_0v_0a_1v_1a_2$, a factor that can be transformed into a weak overlap after insertion of holes, where each v_i is a word of length $p-1$ and the a_j's are letters. Since p is not divisible by 5, it follows that the images of δ in a_0v_0 and a_1v_1 will not be aligned. If the second letters of a_0v_0 and a_1v_1 are equal, then we get a contradiction by Lemma 7 (here no two corresponding length seven subwords in each half starting with the same character can be identical). If the two positions do not match, following Remark 4 it must be that either the first or the third positions must match. Using the same technique we get a contradiction in both these cases. Therefore, no factor of $\delta(w)$ of length $2p + 1 > 21$ with p not divisible by 5 can be turned into a weak overlap. □

Proposition 3. *No factor of $\delta(\Gamma)$ of length $2p + 1 > 21$ with p divisible by 5 can be turned into a weak overlap.*

Proof. Assume towards a contradiction that a factor $a_0v_0a_1v_1a_2$ can be transformed into a weak overlap after insertion of holes. Since $|a_0v_0| = 5k$, for some $k > 2$, it follows that the images of δ will be aligned in a_0v_0 and a_1v_1. By looking at the blocks of δ we see that only the images of b and d do not contain three consecutive mismatches once aligned. Hence, we will consider the case when these two images are aligned, the other cases being straightforward by Remark 4.

Note that the only character preceding d in Γ is a, and the only character preceding b is c, while the only character following d in Γ is c, and the only character following b is a. Assume that the block determined by $\delta(d)$ ends before the last position in a_iv_i with $i \in \{0, 1\}$. The character following this block is j, while the one following the block $\delta(b)$ is f. Note that this letter together with the last two characters of the block gives us the sequences gfj and ijf, that will not match after a valid insertion of holes, by Remark 4.

If the block $\delta(d)$ starts at a position greater than 5, it follows that it is preceded by $\delta(a)$. Since $\delta(a)$ will align with a block $\delta(c)$ according to the previous observations, by Remark 4 we conclude that a matching is impossible. □

Theorem 4. *The infinite word $\delta(\Gamma)$ over a 5-letter alphabet is weakly overlap-free after an arbitrary insertion of holes.*

Proof. This follows directly from Propositions 1, 2, and 3. □

Since strong periodicity implies weak periodicity, the theorem answers an open problem of [2] regarding how large an alphabet must be to create an infinite word that is strongly overlap-free despite arbitrary insertions of holes.

Corollary 2. *The infinite word $\delta(\Gamma)$ over a 5-letter alphabet is strongly overlap-free after an arbitrary insertion of holes.*

Please note that the lower bound of five letters presented in [2] stands, since for alphabets of size smaller than five, all infinite words contain factors of length $2p + 1$ that are strongly p-periodic, and therefore weakly p-periodic. Also note that the use of the terms weakly and strongly overlap-free word comes from the concepts of weak- and strong-periodicity (when looking at overlaps from this point of view, the terminology comes naturally).

References

1. Manea, F., Mercaş, R.: Freeness of partial words. Theoretical Computer Science 389, 265–277 (2007)
2. Blanchet-Sadri, F., Mercaş, R., Scott, G.: A generalization of Thue freeness for partial words. Theoretical Computer Science (2008), doi:10.1016/j.tcs.2008.11.006
3. Thue, A.: Uber unendliche Zeichenreihen. Norske Vid. Selsk. Skr. I, Mat. Nat. Kl. Christiana 7, 1–22 (1906)
4. Thue, A.: Uber die gegenseitige Lage gleicher Teile gewisser Zeichenreihen. Norske Vid. Selsk. Skr. I, Mat. Nat. Kl. Christiana 1, 1–67 (1912)
5. Halava, V., Harju, T., Kärki, T.: Overlap-freeness in infinite partial words. Technical Report 888, Turku Centre for Computer Science (2008)
6. Lothaire, M.: Combinatorics on Words. Cambridge University Press, Cambridge (1997)

Partial Projection of Sets Represented by Finite Automata, with Application to State-Space Visualization*

Bernard Boigelot and Jean-François Degbomont

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

Abstract. This work studies automata-based symbolic data structures for representing infinite sets. Such structures are used in particular by verification tools in order to represent the sets of configurations handled during symbolic exploration of infinite state spaces. Our goal is to develop an efficient projection operator for these representations. There are several needs for such an operator during state-space exploration; we focus here on projecting the set of reachable configurations obtained at the end of exploration. An interesting application is the state-space visualization problem, which consists in providing the user with a graphical picture of a relevant fragment of the reachable state space.

For theoretical reasons, the projection of automata-represented sets is inherently costly. The contribution of this paper is to introduce an improved automata-based data structure that makes it possible to reduce in several cases the effective cost of projection. The idea is twofold. First, our structure allows to apply projection to only a part of an automaton, in cases where a full computation is not necessary. Second, the structure is able to store the results of past projection operations, and to reuse them in order to speed up subsequent computations. We show how our structure can be applied to the state-space visualization problem, and discuss some experimental results.

1 Introduction

State-space exploration is a powerful technique for analyzing the properties of computerized systems. It is not restricted to finite models: infinite state spaces can be explored symbolically, with the help of suitable data structures for representing the sets of configurations that have to be handled [Boi98, BJNT00]. Concretely, if a system undergoing symbolic state-space exploration is controlled by n variables defined over the respective domains $D_1, D_2, \ldots, D_n$, then one needs a data structure suited for representing subsets of $D_1 \times D_2 \times \cdots \times D_n$. This structure must be closed under all operations to be performed during exploration.

* This work is supported by the *Interuniversity Attraction Poles* program *MoVES* of the Belgian Federal Science Policy Office, and by the grant 2.4530.02 of the Belgian Fund for Scientific Research (F.R.S.-FNRS).

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 200–211, 2009.

1.1 Automata-Based Representations

A simple approach consists in representing sets as finite automata: given a fixed alphabet Σ, one considers an encoding relation that maps every value in a domain D onto words over Σ. Such a relation thus encodes subsets S of D as languages. Whenever such languages are regular, they can be accepted by finite automata, which then provide symbolic representations of the sets S. The advantages are that automata are easily manipulated algorithmically, and that most usual operations on sets (such as intersection, union, ...) reduce to carrying out the same operations on the languages accepted by automata [Boi98].

Consider for instance the important case of programs relying on integer variables. Using the positional notation, an integer number can be encoded as a finite word over the alphabet $\{0, 1, \ldots, r-1\}$, where $r > 1$ is an arbitrarily chosen *base*. It has been established that, in this setting, every subset of $\mathbb{Z}$ that is definable in Presburger arithmetic, i.e., the first-order theory $\langle \mathbb{Z}, +, \leq \rangle$, is encoded by a regular language, and is thus automata-representable [BHMV94]. Interestingly, Presburger-definable sets are those that can be expressed as combinations of linear constraints and discrete periodicities [Pre29], which matches quite well the requirements of infinite state-space exploration applications [WB95, SKR98].

In order to represent multidimensional sets, i.e., subsets of $D = D_1 \times D_2 \times \cdots \times D_n$, one needs an encoding relation suited for the global domain D. Such a relation can be obtained by combining together encoding relations suited for each individual domain D_i. Several types of combinations are possible. A first strategy consists in encoding a value $(v_1, v_2, \ldots, v_n) \in D$ by concatenating the individual encodings of $v_1, v_2, \ldots, v_n$, expressed over distinct alphabets. This method is more suited for domains such as communication channel contents [BG96] than for integer variables. Indeed, with this scheme, the Presburger-definable subsets of $\mathbb{Z}^n$ are generally not encoded by regular languages[1].

Another approach is to interleave the encodings of the individual values $v_1, v_2, \ldots, v_n$, by reading repeatedly and successively one symbol in words $w_1, w_2, \ldots, w_n$ encoding those values. This requires these words to share the same length, which can always been achieved by appending a suitable number of padding symbols. In the case of integer numbers encoded positionally, padding is not necessary, since every integer admits encodings of any sufficiently large length. It is known that this composite encoding relation maps every Presburger-definable subset of $\mathbb{Z}^n$ onto a regular language [BHMV94, WB95, Boi98].

In this work, we restrict ourselves to multidimensional encoding relations obtained by the interleaving method. Our motivation stems from the case of programs manipulating integer variables, which is probably the most relevant one in actual applications. Nevertheless, the techniques developed in this paper extend naturally to other domains for which interleaved encodings are also applicable.

1.2 Set Projection and State-Space Visualization

This paper studies the *projection* operation, which intuitively consists in discarding a given subset of variables from a multidimensional set. Formally, given

[1] A simple example is given by the set $\{(x_1, x_2) \in \mathbb{Z}^2 \mid x_1 = x_2\}$.

a set $S \subseteq D_1 \times D_2 \times \cdots \times D_n$ and a set $C = \{i_1, i_2, \ldots\} \subseteq \{1, 2, \ldots, n\}$ of components, the projection of S over C is given by the set $S' = \{(u_{i_1}, u_{i_2}, \ldots) \in D_{i_1} \times D_{i_2} \times \cdots \mid \exists (v_1, v_2, \ldots, v_n) \in S : \forall j : u_{i_j} = v_{i_j}\}$.

During symbolic state-space exploration, the projection operation has to be carried out in several forms. The first one, *local projection*, is needed when one needs to compute the image of a subset of configurations by an operation that discards the current value of some variables. For instance, the effect of an assignment instruction such as $x_1 := 2$ amounts to first projecting the current set of configurations onto all variables but x_1, and then inserting the constant 2 in the first component of all tuples in the resulting set. A different application, *global projection*, corresponds to projecting the whole set of reachable configurations obtained at the end of state-space exploration. This makes it possible to reason on the properties of the reachable set without being hampered by the presence of non-relevant variables.

The aim of this work is to develop an efficient implementation of global projection operations. Our main motivation is the *state-space visualization* problem for programs manipulating integer variables, defined as follows. The goal of state-space exploration is to compute the set of reachable configurations of the system under analysis, in the form of a symbolically-represented set of vectors with integer components. The visualization problem then consists in producing a two-dimensional image of the values taken by a pair of specified variables. Such an image can be obtained by projecting the original set over the selected variables, and then enumerating the values in the resulting set, within given bounds. The aim of visualization is to provide the user with a synthetic and global view of the reachable configurations, so as to draw quickly attention towards erroneous behaviors (which can then be the subject of more focused investigations), or modeling errors.

In order for state-space visualization to be helpful during the software development process, it has to be reasonably efficient. Of course, state-space exploration in itself is usually a quite costly procedure, but that has only to be carried out once for a given system. Having obtained a (typically large) symbolic representation of the reachable set, the problem is thus to visualize it as efficiently as possible, with respect to different choices of variables or viewing parameters (in particular, one should be able to move at will the visualization window, as well as change the zoom factor). This requires an efficient implementation of the projection operator.

1.3 Projecting Sets Represented by Automata

In the case of automata-based representations of multidimensional sets, projection is seemingly a simple operation. Indeed, assuming an interleaved encoding scheme, one can locate in linear time the automaton transitions associated with the variables discarded by the projection, and simply relabel them with the empty word. The drawback of this approach is that it gives out non-deterministic automata.

For state-space exploration however, using non-deterministic representations is problematic. First, during exploration, working systematically with deterministic automata makes it possible to minimize them into a canonical form [Hop71]. This makes the representation of sets independent from their construction, which often helps to keep the size of the representations under control. Another problem

is that testing inclusion between sets, which is needed for checking that a fixed point has been reached during exploration, can only be implemented with a reasonable cost on deterministic automata. Furthermore, visualizing a set with respect to a given pair of variables requires to check for each pixel of the display window whether it has to be lit or not. For a given pixel, this amounts to checking whether the projection of the underlying automaton over the selected variables accepts or not an encoding of the coordinates of the corresponding point. With non-deterministic automata, this procedure requires to check a prohibitively large number of paths. Finally, it is worth mentioning that the worst-case exponential cost of the determinization operation is seldom observed in practice; for automata produced by state-space exploration tools, determinization usually remains an efficient operation [BW02].

The implementation of a usable state-space visualization tool is thus faced with the problem of computing as efficiently as possible a deterministic automaton representing the projection of a given set. This problem is inherently difficult. Indeed, one can easily build families of deterministic automata whose determinized projection is exponentially larger. A possible workaround could be to limit the expressiveness of the symbolic representations. Assuming that only Presburger-definable sets have to be represented, a potential strategy could be to exploit the known structure of the automata representing such sets [Lat05, Ler05]. Unfortunately, this approach is not feasible, for a polynomial algorithm for the projection operator would lead to a polynomial decision procedure for Presburger arithmetic, which does not exist [Opp78].

In spite of these theoretical limitations, it is nevertheless possible to reduce the effective cost of projection in some applications. The approach we propose is based on two ideas. First, projection does not always have to be applied to whole automata. In particular, we show that state-space visualization can be speeded up by only projecting subsets of the transition graph of the original automaton (we name this operation *partial projection*). Second, some projection computations can reuse the results of previous computations. For instance, projecting a set over $\{x_1\}$ becomes simpler if the projection of that set over $\{x_1, x_2\}$ is already available.

The contributions of this paper are the definition of an original data structure allowing the computation of partial projections as well as the efficient reuse of the results of past computations. We also show how this data structure can be exploited for implementing state-space visualization, and then discuss some experimental results.

2 Basic Notions

2.1 Automata-Based Representations of Sets

In order to represent subsets of a domain D by finite automata, one needs an *encoding relation* $E \subseteq D \times \Sigma^*$ mapping the elements of D onto finite words over a finite alphabet Σ. A word can only encode one value, hence the relation E must be such that $\forall (v_1, w_1), (v_2, w_2) \in E : \; v_1 \neq v_2 \Rightarrow w_1 \neq w_2$. Moreover, each element of D must be encoded by at least one word.

For a set $S \subseteq D$, we define its *encoding* as the language $E(S) = \{w \in \Sigma^* \mid \exists v \in S : (v, w) \in E\}$. If this language is regular, then any finite automaton that accepts $E(S)$ is a *representation* of S. We denote finite automata by tuples $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ the alphabet, δ a transition relation, with $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ for non-deterministic automata, and $\delta : (Q \times \Sigma) \to Q$ for deterministic ones, $q_0 \in Q$ an initial state, and $F \subseteq Q$ a set of accepting states.

2.2 Multidimensional Domains

A domain D is *multidimensional* if it can be expressed as the Cartesian product $D = D_1 \times D_2 \times \cdots D_n$ of simpler domains D_i, where $n \geq 0$ is the *dimension*. Assuming that encoding relations E_i have been defined for the domains D_i, for $i = 1, 2, \ldots, n$, we build an encoding relation E suited for D in the following way. First, for simplicity's sake, we assume that the relations E_i are defined over the same alphabet, i.e., $E_i \in D_i \times \Sigma^*$. Then, we require each E_i to be such that, for every value $v_i \in D_i$ and sufficiently large integer N, there exists w_i such that $|w_i| = N$ and $(v_i, w_i) \in E_i$. In other words, every value in D_i must admit encodings of any sufficiently large length. Note that any relation can be turned into one that satisfies this requirement, by appending appropriate *padding symbols* to the encodings of values.

In order to encode a multidimensional value $v = (v_1, v_2, \ldots, v_n) \in D$, we first consider individual encodings $w_1, w_2, \ldots, w_n \in \Sigma^*$ of $v_1, v_2, \ldots, v_n$, such that $|w_1| = |w_2| = \cdots = |w_n|$. Then, we build the word $w = a_1 a_2 \ldots a_n\, b_1 b_2 \ldots b_n \ldots$, where $w_i = a_i b_i \ldots$ for each $i = 1, 2, \ldots, n$. In other words, w is constructed by reading repeatedly one symbol in $w_1, w_2, \ldots, w_n$, in fixed order. By mapping every value $v \in D$ onto the words w obtained in this way, we get an encoding relation $E \subseteq D \times \Sigma^*$ suited for D. Note that a value may admit several distinct encodings, and that the length of encodings is restricted to integer multiples of the domain dimension n. Hence, we have $E \subseteq D \times (\Sigma^n)^*$ [Boi98].

2.3 Projection

Let S be a subset of a multidimensional domain $D = D_1 \times D_2 \times \cdots D_n$, and $C \subseteq \{1, 2, \ldots, n\}$ be a set of *components*. The *projection* of S over C is defined as the set $\pi_C(S) = \{(u_{i_1}, u_{i_2}, u_{i_3}, \ldots) \in D_{i_1} \times D_{i_2} \times D_{i_3} \times \cdots \mid \exists (v_1, v_2, \ldots, v_n) \in S : \forall j : u_{i_j} = v_{i_j}\}$, where $C = \{i_1, i_2, i_3, \ldots\}$ with $i_1 < i_2 < i_3, \cdots$. In other words, projecting S over C amounts to removing from every element of S the components that do not belong to C.

A similar operator can also be defined over words that encode multidimensional values. Given a dimension n and a nonempty set of components $C \subseteq \{1, 2, \ldots, n\}$, the projection of a word $a = a_1 a_2 \cdots a_n$ over C, with $\forall i : a_i \in \Sigma$, is given by $\pi_C(a) = a_{i_1} a_{i_2} a_{i_3} \ldots$, where $C = \{i_1, i_2, i_3, \ldots\}$ with $i_1 < i_2 < i_3, \cdots$. For an empty set of components $C = \{\,\}$, we define $\pi_C(a) = \alpha$, where α is a distinguished symbol[2]. Then, the projection of a word $w = w_1 w_2 \cdots w_q$,

[2] The motivation behind this definition is to keep track of the length of a word in all its projections, even those over the empty set of components.

with $|w_i| = n$ for each $i \in \{1, \ldots, q\}$, over C is then defined as $\pi_C(w) = \pi_C(w_1)\pi_C(w_2)\cdots\pi_C(w_q)$. Finally, the projection of a language of encodings $L \subseteq (\Sigma^n)^*$ over C is given by $\pi_C(L) = \{\pi_C(w) \mid w \in L\}$.

Note that the projection operator preserves the regularity of languages: an automaton accepting $\pi_C(L)$ can easily be built from one accepting L. This is done by first unrolling the transition graph of the automaton so that each transition corresponds to one individual component. Then, the transitions associated with the components that do not belong to C are relabeled with the empty word. This procedure gives out a non-deterministic automaton.

Let $S \subseteq D$ be a set represented by an automaton $\mathcal{A}$. The projection $\pi_C(S)$ of S over some set of components C cannot be simply computed by constructing an automaton accepting $\pi_C(L(\mathcal{A}))$, where $L(\mathcal{A})$ denotes the language accepted by $\mathcal{A}$. Indeed, although each word in $\pi_C(L(\mathcal{A}))$ encodes correctly the projection of some element of S, this language may not necessarily contain all such encodings. The solution to this problem depends on the encoding relation used, and consists in first applying the language projection operator, and then performing a domain-specific *saturation* operation that adds to the resulting language the missing encodings of the represented values [Boi98].

2.4 Application to Sets of Integers

Consider the domain $\mathbb{Z}$ of integer numbers. Choosing a *base* $r \in \mathbb{N} \setminus \{0\}$, the positional notation encodes a number $z \in \mathbb{N}$ as a word $d_{p-1}d_{p-2}\ldots d_1d_0$ over the finite alphabet $\{0, 1, \ldots, r-1\}$, such that $z = \sum_{0 \leq i < p} d_i r^i$. This encoding relation generalizes to numbers in $\mathbb{Z}$ by encoding negative numbers by their *r-complement* [WB95]. The length p of encodings is not fixed. This implies that every number has encodings of any sufficiently large length. One can then apply the technique outlined in Section 2.2 so as to obtain a positional encoding relation suited for subsets of $\mathbb{Z}^n$, for any dimension $n \geq 0$.

The resulting automata-based representation for sets of vectors in $\mathbb{Z}^n$ is called the *Number Decision Diagram (NDD)* [WB95, Boi98]. It is known that all subsets of $\mathbb{Z}^n$ that are definable in Presburger arithmetic, i.e., the first-order theory $\langle \mathbb{Z}, +, \leq \rangle$, can be represented by NDDs [Büc62, BHMV94]. In order to project a set represented by an NDD $\mathcal{A}$, one simply projects the language $L(\mathcal{A})$ accepted by $\mathcal{A}$, and then applies the saturation algorithm developed in [BL04]. Automata-based representations of numbers have been extended to sets of vectors with mixed integer and real components, by moving to infinite-word automata [BJW05].

3 State-Space Visualization

3.1 Problem Statement

Let $n \geq 2$ be a dimension, and $S \subseteq \mathbb{Z}^n$ be a set of vectors represented by a NDD $\mathcal{A}$ in a base $r > 1$. In our intended application, S is the set of reachable configurations of a program controlled by n integer variables, and its representation

$\mathcal{A}$ is produced by a symbolic state-space exploration algorithm [Boi98]. In this setting, r is typically equal to 2.

The *visualization* problem consists in extracting from $\mathcal{A}$ a two-dimensional picture of some fragment of S that is relevant to the user. More precisely, the user provides two *components* $i_1, i_2 \in \{1, 2, \ldots, n\}$, with $i_1 \neq i_2$, a *center position* $(x, y) \in \mathbb{Z}^2$, a *zoom factor* $f \in \mathbb{N}$, with $f > 0$, and a *window size* $(W, H) \in \mathbb{N}^2$ with $W > 0$ and $H > 0$. The idea is that each pixel in the visualization window corresponds to a square region of size $f \times f$ in the domain $\mathbb{Z}^2$, with the window centered on the point of coordinates (x, y). The goal of the visualization procedure is to light up the pixels corresponding to regions that contain at least one value in $\pi_{\{i_1,i_2\}}(S)$.

3.2 Visualization and Projection

Visualization can thus be achieved by first computing a NDD representing the set $\pi_{\{i_1,i_2\}}(S)$, and then scanning its accepting paths for values that fit in the visualization window. In order to speed up the latter operation, which has to be carried out for each pixel in the window, a good strategy is to determinize the automaton representing $\pi_{\{i_1,i_2\}}(S)$. Then, whether a value (v_1, v_2) belongs or not to this set can be checked by examining a single automaton path.

Even with a deterministic automaton, the number of paths to be checked can become prohibitive for large zoom factors f, since each pixel covers f^2 distinct points of $\mathbb{Z}^2$. A solution is to restrict the allowable values of f to be equal to powers r^k, with $k \in \mathbb{N}$, of the representation base r, and to align the pixel boundaries on coordinates that are exact multiples of f (this is not problematic in actual applications). With those restrictions, the values covered by a single pixel correspond to a square region of the form $[v_1 r^k, (v_1+1)r^k - 1] \times [v_2 r^k, (v_2+1)r^k - 1]$, with $v_1, v_2 \in \mathbb{Z}$ and $k \in \mathbb{N}$. It is then sufficient to check whether the automaton admits a accepting path that reads an encoding of (v_1, v_2) followed by $2k$ arbitrary symbols.

In a deterministic automaton, following a given prefix is a simple operation. Checking whether, from a given state q, there exists an accepting path of given length l is more problematic, since it may require to examine a large number of paths. Remark that this operation is actually equivalent to projecting the language $L(q)$ accepted from q over the empty set of components, determinizing the resulting automaton, and then checking whether one has $\alpha^l \in \pi_{\{\}}(L(q))$, which can then be done efficiently.

3.3 Avoiding Redundant Computations

Explicitly carrying out a projection followed by a determinization for each considered pixel would however lead to redundant computations. First, some automata states are reached by different prefixes, hence the number of distinct states from which a projection needs to be performed can actually be smaller than the total number of pixels. Second, when the user moves the visualization window, some pixels may correspond to coordinates that have already been checked in previous computations.

Our solution for curbing redundant computations is based on an original data structure, the *Partially Projected Automaton (PPA)*, in which projection and determinization operations can be applied to sub-automata, with their results stored in order to be subsequently reusable. For visualization, the idea is thus to use PPA mainly in order to project efficiently the languages accepted from automaton states over the empty set of components. Interestingly, it turns out that the computation of $\pi_{\{i_1,i_2\}}(S)$, i.e., the projection of the whole reachable set over the pair of selected components, can also benefit from such a data structure.

Indeed, for a regular language $L \subseteq (\Sigma^n)^*$ and a set of components $C \in \{1, 2, \ldots, n\}$, there are several ways of computing a deterministic automaton accepting $\pi_C(L)$ from one accepting L. A first technique is to build a non-deterministic automaton accepting $\pi_C(L)$, and then determinize it at once. An alternative approach is to extract the components $i_1, i_2, \ldots, i_m$ that do not belong to C, i.e., such that $\{i_1, i_2, \ldots, i_m\} = \{1, 2, \ldots, n\} \setminus C$, and to project them out one by one, in some given order, determinizing the resulting automaton after each projection. In the context of symbolic state-space exploration of programs relying on integer variables, we have observed experimentally that the latter solution is more efficient in practice (see Section 5).

With this solution, the results of past projection operations can in some situations be reused in order to speed up new computations. Consider for instance a 5-dimensional domain. Assuming that components are projected out individually in increasing order, the computations of $\pi_{\{2,3\}}(L)$ and $\pi_{\{2,5\}}(L)$ will respectively be decomposed into $\pi_{\{1,2\}}\,\pi_{\{1,2,4\}}\,\pi_{\{2,3,4,5\}}(L)$ and $\pi_{\{1,3\}}\,\pi_{\{1,2,4\}}\,\pi_{\{2,3,4,5\}}(L)$. In this case, the intermediate result $\pi_{\{1,2,4\}}\,\pi_{\{2,3,4,5\}}(L)$ can be reused across both computations.

4 Partially Projected Automata

4.1 Definition

A *Partially Projected Automaton (PPA)* $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, \delta^D, dim, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : (Q \times \Sigma) \to Q$ is a (deterministic) transition relation, $\delta^D : (Q \times 2^{\mathbb{N}}) \to Q$ is a *decomposition relation*, $dim : Q \to \mathbb{N}$ is a *dimension function*, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting states.

In order for a PPA to be well formed, its components have to satisfy some constraints. The language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is defined as being equal to the language accepted by the finite automaton $(Q, \Sigma, \delta, q_0, F)$. This language is supposed to encode some set of vectors from a n-dimensional domain D. This implies that all words in $L(\mathcal{A})$ have a length that is an integer multiple of n. In order to enforce this constraint, the function *dim* associates each state of $\mathcal{A}$ with a *dimension*, which intuitively corresponds to the dimension of the vectors recognized from that state. For all states q visited by the paths of $(Q, \Sigma, \delta, q_0, F)$, one thus has $dim(q) = dim(q_0) = n$.

The purpose of the decomposition relation δ^D is to provide redundant information, corresponding to the results of language projection operations that

have previously been carried out from automaton states. For states $q, q' \in Q$ and a subset of components $C \subseteq \mathbb{N}$ such that $q' = \delta^D(q, C)$, we must have $C \subset \{1, \ldots, dim(q)\}$ and $dim(q') = |C|$. The language accepted from the state q' is then equal to $\pi_C(L(q))$, where $L(q)$ denotes the language accepted from q. The paths of (Q, δ) that can be followed from q' must thus only visit states q'' such that $dim(q'') = dim(q') = |C|$. Each such path σ reads a word w that belongs to $\pi_{(C,|w|)}(L(q))$ iff σ ends in an accepting state.

In summary, a PPA accepting a language encoding a set of n-dimensional vectors can be seen as n distinct finite-state automata, each of them recognizing sets with a specific dimension in $\{1, \ldots, n\}$, linked together by the decomposition relation. Decompositions can be nested, in the sense that from the destination of a decomposition transition, one may reach states from which other decomposition transitions are defined. The advantage of PPA is that they provide a way of storing and reusing the results of previous projection operations carried out from automaton states. These stored results do not have to be kept indefinitely, since decomposition transitions can always be removed from a PPA without affecting its accepted language. Procedures for constructing well-formed PPA and manipulating them are discussed in the next section.

4.2 Algorithms

Let $L \subseteq \Sigma^*$ be a regular language encoding the n-dimensional set $S \subseteq D$. A well-formed PPA $\mathcal{A}$ representing S can simply be constructed by associating a deterministic finite-state automaton $(Q, \Sigma, \delta, q_0, F)$ accepting L with an empty decomposition relation, i.e., by defining $\mathcal{A} = (Q, \Sigma, \delta, \{\,\}, dim, q_0, F)$, with $dim(q) = n$ for all $q \in Q$.

After obtaining such a PPA, projecting the language accepted from one of its states may result in adding decomposition transitions, in order to remember the result of this operation so as to be able to reuse it later. The size of a PPA thus grows with projection operations. At any time, one may choose to curb this growth by removing arbitrary decomposition transitions, as well as the states and transitions that then become unreachable. Different heuristics can be used for selecting the decomposition transitions to be removed: bounding the total memory footprint of the structure, discarding the last recently or the least frequently followed decomposition transition, ... In all cases, removing decomposition transitions has no influence on the language accepted by the PPA.

We now sketch the algorithm computing the projection $\pi_C(L(q))$ of the language accepted from some state q of a PPA $\mathcal{A} = (Q, \Sigma, \delta, \delta^D, dim, q_0, F)$, with respect to a set of components $C \subset \{1, \ldots, dim(q)\}$.

If $\delta^D = \{\,\}$, then the projection is carried out by composing the automaton $(Q, \Sigma, \delta, q, F)$ with a finite-state transducer, constructed with a transition relation that takes the form of a single cycle of length $dim(q)$. Each transition in this cycle thus corresponds to one vector component in $\{1, \ldots, dim(q)\}$. The transitions corresponding to components in C are designed so as to give out a copy of their input symbol, the other transitions producing an empty word (or, in a case of a projection over $\{\,\}$, one occurrence of the distinguished symbol

α). The result takes the form of a non-deterministic automaton $\mathcal{A}'$, that can be determinized into an automaton $\mathcal{A}''$ using the classical subset construction. Finally, a decomposition transition $\delta^D(q, C) = q''$ is added to $\mathcal{A}$, where q'' is the initial state of $\mathcal{A}''$. Note that each state q' of the determinized projected automaton thus corresponds to a subset $Q_{q'}$ of states of $\mathcal{A}$.

If, on the other hand, $\delta^D \neq \{\}$, the construction is similar but it may then become possible to reuse the results of earlier projections. This happens when a state q' of the projected automaton is such that all the states in $Q_{q'}$ admit outgoing decomposition transitions with the same destination q'' and subset C' of components, such that $C \subseteq C'$. In this case, the computation of the projection can be continued by exploring the successors of q'' instead of those of q.

Finally, the automaton obtained after a projection operation are minimized, by merging states that are known to accept identical languages. This is done by partitioning the states of the automaton according to their dimension, and then applying classical finite-state machine minimization [Hop71] to each part.

5 Experimental Results

The data structure and the algorithms outlined in Section 4 have been implemented in a prototype tool for visualizing NDDs. Our evaluation considers random NDDs generated by the same method as the one used in [BW02], and measures the time needed for displaying them in a 512×512 window, changing the zoom factor from 1 to 256, and moving the view from $(0, 0)$ to $(2^{16}+1, 2^{16}+1)$. The results are given in Figure 1. Those results demonstrate that, although the inherent cost of projection is not avoided, reusing previous results is useful, especially for translating efficiently the visualization window.

	NDD size	without PPA			with PPA		
	$\|Q\|$	t_{disp}	tzoom	tmove	t_{disp}	tzoom	tmove
N_1	621	0.01	0.85	2.33	0.01	0.07	0.04
N_2	755	0.03	2.56	7.63	0.03	0.12	0.12
N_3	1938	0.14	15.98	12.06	0.14	0.64	0.28
N_4	13944	0.15	15.26	11.58	0.15	11.44	0.84

Fig. 1. Time cost of visualization operations (in seconds)

	n	$\|Q\|$	$t_{\text{at-once}}$	t_{incr}		n	$\|Q\|$	$t_{\text{at-once}}$	t_{incr}
S_1	4	434	0.02	0.03	S_7	7	50135	240.86	9.30
S_2	5	224	0.04	0.01	S_8	5	4474	1.70	0.50
S_3	4	15272	154.15	0.80	S_9	7	99169	176.29	100.07
S_4	5	915	0.80	0.06	S_{10}	7	132709	243.52	109.94
S_5	4	3446	0.30	0.30	S_{11}	6	63410	71.50	2.90
S_6	3	1279	0.08	0.10	S_{12}	6	66330	> 1000	5.50

Fig. 2. At-once vs incremental projection (in seconds)

We also evaluated experimentally the benefits of projecting and determinizing a NDD incrementally instead of at once, as discussed in Section 3.3. Figure 2 gives the time spent by both methods for projecting the family of sets described in [Lat05] over two components. The incremental approach clearly stands out.

6 Conclusions

In this paper, we have introduced a simple yet powerful data structure for storing and reusing the results of *partial projections* of finite automata, i.e., projection operations carried out from a given state. We have applied our results to the problem of visualizing a part of the set of reachable configurations produced by a state-space exploration tool. In this setting, the advantage of our approach is not to reduce the inherent cost of projection, but rather to avoid performing redundant computations. This makes the procedure efficient when only slight modifications are applied to the value of parameters, such as the zoom factor or the coordinates of the visualization window. A prototype implementation of the proposed method has been developed, showing that the approach provides clear benefits. Although the focus of this paper was on sets on integers represented by finite-word automata, it is worth mentioning that our technique straightforwardly generalizes to mixed integer and real sets represented by weak infinite-word automata [BJW05]. Future work will address the problem of making PPA compatible with efficient representations of automata, such as [Cou04].

Acknowledgments

We would like to thank Louis Latour and Laetitia Smisdom for their contribution to the investigation of the visualization problem.

References

[BG96] Boigelot, B., Godefroid, P.: Symbolic verification of communication protocols with infinite state spaces using QDDs. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 1–12. Springer, Heidelberg (1996)

[BHMV94] Bruyère, V., Hansel, G., Michaux, C., Villemaire, R.: Logic and p-recognizable sets of integers. Bulletin of the Belgian Mathematical Society 1(2), 191–238 (1994)

[BJNT00] Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 403–418. Springer, Heidelberg (2000)

[BJW05] Boigelot, B., Jodogne, S., Wolper, P.: An effective decision procedure for linear arithmetic over the integers and reals. ACM Trans. Comput. Logic 6(3), 614–633 (2005)

[BL04] Boigelot, B., Latour, L.: Counting the solutions of Presburger equations without enumerating them. Theoretical Computer Science 313, 17–29 (2004)

[Boi98] Boigelot, B.: Symbolic Methods for Exploring Infinite State Spaces. PhD thesis, University of Liége, Belgium (1998)

[BW02] Boigelot, B., Wolper, P.: Representing arithmetic constraints with finite automata: An overview. In: Stuckey, P.J. (ed.) ICLP 2002. LNCS, vol. 2401, pp. 1–19. Springer, Heidelberg (2002)

[Büc62] Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proc. International Congress on Logic, Methodoloy and Philosophy of Science, pp. 1–12. Stanford University Press, Stanford (1962)

[Cou04] Couvreur, J.-M.: A BDD-like implementation of an automata package. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 310–311. Springer, Heidelberg (2005)

[Hop71] Hopcroft, J.E.: An $n \log n$ algorithm for minimizing states in a finite automaton. Theory of Machines and Computation, 189–196 (1971)

[Lat05] Latour, L.: Presburger Arithmetic: From Automata to Formulas. PhD thesis, University of Liége, Belgium (2005)

[Ler05] Leroux, J.: A polynomial time Presburger criterion and synthesis for number decision diagrams. In: Proc. 20th LICS, pp. 147–156. IEEE Computer Society, Los Alamitos (2005)

[Opp78] Oppen, D.C.: A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. Journal of Computer and System Sciences 16, 323–332 (1978)

[Pre29] Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du Premier Congrès des Mathématiciens des Pays Slaves, Warsaw, pp. 92–101 (1929)

[SKR98] Shiple, T.R., Kukula, J.H., Ranjan, R.K.: A comparison of presburger engines for EFSM reachability. In: Y. Vardi, M. (ed.) CAV 1998. LNCS, vol. 1427, pp. 280–292. Springer, Heidelberg (1998)

[WB95] Wolper, P., Boigelot, B.: An automata-theoretic approach to Presburger arithmetic constraints. In: Mycroft, A. (ed.) SAS 1995. LNCS, vol. 983, pp. 21–32. Springer, Heidelberg (1995)

Larger Lower Bounds on the OBDD Complexity of Integer Multiplication*

Beate Bollig

LS2 Informatik, TU Dortmund,
44221 Dortmund, Germany

Abstract. Integer multiplication as one of the basic arithmetic functions has been in the focus of several complexity theoretical investigations. Ordered binary decision diagrams (OBDDs) are one of the most common dynamic data structures for Boolean functions. Only recently it has been shown that the OBDD complexity of the most significant bit of integer multiplication is exponential, answering an open question posed by Wegener (2000). In this paper a larger lower bound is presented, using a simpler proof. Moreover, the best known lower bound on the OBDD complexity for the so-called graph of integer multiplication is improved.

1 Introduction and Results

Integer multiplication is certainly one of the most important functions in computer science and a lot of effort has been spent in designing good algorithms and small circuits and in determining its complexity. Ordered binary decision diagrams (OBDDs) are one of the most common dynamic data structures for Boolean functions. Although many exponential lower bounds on the OBDD size of Boolean functions are known and the lower bound methods are simple, it is often a more difficult task to prove large lower bounds for some predefined and interesting functions. Despite the well-known lower bounds on the OBDD size of the so-called middle bit of multiplication ([1], [2]), only recently it has been shown that the OBDD complexity of the most significant bit of multiplication is also exponential [3] answering an open question posed by Wegener [4]. Here, we present a simpler proof that lead to a larger lower bound. As a by-product the known lower bound on the OBDD complexity of the so-called graph of integer multiplication is improved.

1.1 Ordered Binary Decision Diagrams

Boolean circuits, formulae, and binary decision diagrams (BDDs), sometimes called branching programs, are standard representations for Boolean functions. (For a history of results on binary decision diagrams see, e.g., the monograph of Wegener [4]). Besides the complexity theoretical viewpoint people have used

* Ideas presented in this paper were obtained during the Dagstuhl seminar 08381 on computational complexity of discrete problems.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 212–223, 2009.

restricted binary decision diagrams in applications. Bryant [5] has introduced ordered binary decision diagrams (OBDDs) which have become one of the most popular data structures for Boolean functions. Among the many areas of application are verification, model checking, computer-aided design, relational algebra, and symbolic graph algorithms.

Definition 1. *Let $X_n = \{x_1, \ldots, x_n\}$ be a set of Boolean variables. A variable order π on X_n is a permutation on $\{1, \ldots, n\}$ leading to the ordered list $x_{\pi(1)}, \ldots, x_{\pi(n)}$ of the variables.*

In the following a variable order π is sometimes identified with the corresponding order $x_{\pi(1)}, \ldots, x_{\pi(n)}$ of the variables if the meaning in clear from the context.

Definition 2. *A π-OBDD on X_n is a directed acyclic graph $G = (V, E)$ whose sinks are labeled by Boolean constants and whose non sink (or inner) nodes are labeled by Boolean variables from X_n. Each inner node has two outgoing edges one labeled by 0 and the other by 1. The edges between inner nodes have to respect the variable order π, i.e., if an edge leads from an x_i-node to an x_j-node, $\pi^{-1}(i) \leq \pi^{-1}(j)$ (x_i precedes x_j in $x_{\pi(1)}, \ldots, x_{\pi(n)}$). Each node v represents a Boolean function $f_v : \{0,1\}^n \to \{0,1\}$ defined in the following way. In order to evaluate $f_v(b)$, $b \in \{0,1\}^n$, start at v. After reaching an x_i-node choose the outgoing edge with label b_i until a sink is reached. The label of this sink defines $f_v(b)$. The size of the π-OBDD G is equal to the number of its nodes and π-OBDD(f) denotes the size of the minimal π-OBDD representing f.*

The size of the minimal π-OBDD representing a Boolean function f on n variables, i.e., $f \in B_n$, is described by the following structure theorem [6].

Theorem 1. *The number of $x_{\pi(i)}$-nodes of the π-OBDD for f is the number s_i of different subfunctions $f_{|x_{\pi(1)}=a_1,\ldots,x_{\pi(i-1)}=a_{i-1}}$, $a_1, \ldots, a_{i-1} \in \{0,1\}$, essentially depending on $x_{\pi(i)}$ (a function g depends essentially on a variable z if $g_{|z=0} \neq g_{|z=1}$).*

The following simple observation is helpful in order to prove lower bounds. Given an arbitrary variable order π the number of nodes labeled by a variable x in the minimal π-OBDD representing a given function f is not smaller than the number of x-nodes in a minimal π-OBDD representing any subfunction of f.

It is well known that the size of an OBDD representing a function f depends on the chosen variable order. Since in applications the variable order is not given in advance we have the freedom (and the problem) to choose a good or even an optimal order for the representation of f.

Definition 3. *The OBDD size or OBDD complexity of a function f (denoted by OBDD(f)) is the minimum of all π-OBDD(f).*

1.2 Integer Multiplication and Ordered Binary Decision Diagrams

In the last years a new research branch has emerged which is concerned with the theoretical design and analysis of so-called symbolic algorithms for classical

graph problems on OBDD-represented graph instances (see, e.g., [7,8], [9], and [10]). Symbolic algorithms have to solve problems on a given graph instance by efficient functional operations offered by the OBDD data structure. Therefore, at the beginning the OBDD-based algorithms have been justified by analyzing the number of executed OBDD operations (see, e.g., [7,8]). Newer research tries to analyze the over-all runtime of symbolic methods including the analysis of all OBDD sizes occurring during such an algorithm (see, e.g., [10]). In order to investigate the limits of symbolic graph algorithms for the all-pairs shortest paths problem Sawitzki [9] has investigated the graph of integer multiplication.

Definition 4. *The Boolean function* MUL-Graph$_n \in B_{4n}$ *maps two* n*-bit integers* $x = x_{n-1} \dots x_0$ *and* $y = y_{n-1} \dots y_0$*, and a* $2n$*-bit integer* $z = z_{2n-1} \dots z_0$ *to 1 iff the product of* x *and* y *equals* z*, where* x_0, y_0, z_0 *denote the least significant bits.*

Sawitzki [9] has shown that the OBDD size for MUL-Graph$_n$ is at least $\Omega(2^{n/768})$. Recently, in [11] this lower bound has been improved up to $\Omega(2^{n/24})$. Here, we present a further simplification of the lower bound proof and present a lower bound of $\Omega(2^{n/8})$.

Lower bounds for integer multiplication are also motivated by the general interest in the complexity of important arithmetic functions.

Definition 5. *The Boolean function* $\mathrm{MUL}_{i,n} \in B_{2n}$ *maps two* n*-bit integers* $x = x_{n-1} \dots x_0$ *and* $y = y_{n-1} \dots y_0$ *to the* i*th bit of their product , i.e., we write* $\mathrm{MUL}_{i,n}(x, y) = z_i$*, where* $x \cdot y = z_{2n-1} \dots z_0$ *and* x_0, y_0, z_0 *denote the least significant bits.*

The first nontrivial upper bound on the OBDD complexity for the so-called middle bit of integer multiplication $\mathrm{MUL}_{n-1,n}$ has been shown in [2]. In [12] upper bounds on the OBDD size for all functions $\mathrm{MUL}_{i,n}$ have been investigated. Recently, the best known upper bound on the OBDD complexity for the most significant bit of integer multiplication $\mathrm{MUL}_{2n-1,n}$ has been proved in [13].

The first exponential lower bound has also been proved for $\mathrm{MUL}_{n-1,n}$. Bryant [1] has presented a lower bound of $2^{n/8}$. Progress in the analysis of the middle bit of integer multiplication has been achieved by an approach using universal hashing and as a result Woelfel [2] has improved Bryant's lower bound up to $\Omega(2^{n/2})$. In the meantime exponential lower bounds for the middle bit of multiplication have also been proved for more general binary decision diagram models (see, e.g., [14] and [15]). Only recently it has been shown that the OBDD complexity of the most significant bit of multiplication is exponential [3] and a lower bound of $\Omega(2^{n/720})$ has been proved. Here, we present a simpler proof that lead to a lower bound of $\Omega(2^{n/72})$. Since it has been noted before that the lower bound in [3] can be improved up to $\Omega(2^{n/288})$ [16], the merit of the new result is a simplified proof. As a result we gain more insight into the structure of the most significant bit of integer multiplication.

Our results can be summarized as follows.

Theorem 2. $OBDD(\text{MUL-Graph}_n) = \Omega(2^{n/8})$.

Theorem 3. $OBDD(\mathrm{MUL}_{2n-1,n}) = \Omega(2^{n/72})$.

In Section 2 we define some notation and present some basics concerning communication complexity. Moreover, to the best of our knowledge we present the first fooling set of exponential size for a certain Boolean function obtained by the composition of three simpler functions. As a warm-up in Section 3 the known lower bound for the graph of integer multiplication is improved by a reduction from the function equality, the test whether two n-bit numbers are identical, to the graph of integer multiplication. Section 4 contains the main result of the paper, a simpler proof of an exponential lower bound on the OBDD complexity of $\mathrm{MUL}_{2n-1,n}$. The idea is to reduce the Boolean function investigated in Section 2 to the most significant bit of integer multiplication in an appropriate way. Here, also the reduction presented in Section 3 will be helpful.

2 Preliminaries

2.1 Notation

In the rest of the paper we use the following notation.

Let $[x]_r^l$, $n-1 \geq l \geq r \geq 0$, denote the bits $x_l \dots x_r$ of a binary number $x = (x_{n-1}, \dots, x_0)$. For the ease of description we use the notation $[x]_r^l = z$ if $(x_l, \dots, x_r)$ is the binary representation of the integer $z \in \{0, \dots, 2^{l-r+1} - 1\}$. Sometimes, we identify $[x]_r^l$ with z if the meaning is clear from the context. We use the notation $(z)_r^l$ for an integer z to identify the bits at position $l, \dots, r$ in the binary representation of z.

Let $\ell \in \{0, \dots, 2^m - 1\}$, then $\overline{\ell}$ denotes the number $(2^m - 1) - \ell$. For a binary number $x = (x_{n-1}, \dots, x_0)$ we use the notation $\overline{x}$ for the binary number $(\overline{x}_{n-1}, \dots, \overline{x}_0)$.

2.2 Communication Complexity

In order to obtain lower bounds on the size of OBDDs one-way communication complexity has become a standard technique (see Hromkovič [17] and Kushilevitz and Nisan [18] for the theory of communication complexity and the results mentioned below).

The main subject is the analysis of the following (restricted) communication game. Consider a Boolean function $f \in B_n$ which is defined on the variables in $X_n = \{x_1, \dots, x_n\}$, and let $\Pi = (X_A, X_B)$ be a partition of X_n. Assume that Alice has only access to the input variables in X_A and Bob has only access to the input variables in X_B. In a one-way communication protocol, upon a given input x, Alice is allowed to send a single message (depending on the input variables in X_A) to Bob who must then be able to compute the answer $f(x)$. The *one-way communication complexity* of the function f denoted by $C(f)$ is the worst case number of bits of communication which need to be transmitted by such a protocol that computes f. It is easy to see that an OBDD G with respect to a variable order where the variables in X_A are tested before the variables in X_B can be transformed into a communication protocol and $C(f) \leq \lceil \log |G| \rceil$. Therefore, linear lower bounds on the communication complexity of a function

$f : \{0,1\}^{|X_A|} \times \{0,1\}^{|X_B|} \to \{0,1\}$ lead to exponential lower bounds on the size of π-OBDDs where the X_A-variables are before the X_B-variables in π.

One central notion of communication complexity are fooling sets which play an important role for the lower bound proof used later on.

Definition 6. *Let $f : \{0,1\}^{|X_A|} \times \{0,1\}^{|X_B|} \to \{0,1\}$. A set $S \subseteq \{0,1\}^{|X_a|} \times \{0,1\}^{|X_B|}$ is called* fooling set *for f if $f(a,b) = c$ for all $(a,b) \in S$ and some $c \in \{0,1\}$ and if for different pairs $(a_1,b_1),(a_2,b_2) \in S$ at least one of $f(a_1,b_2)$ and $f(a_2,b_1)$ is unequal to c.*

Theorem 4. *If $f : \{0,1\}^{|X_A|} \times \{0,1\}^{|X_B|} \to \{0,1\}$ has a fooling set of size t, the communication complexity of f is bounded below by $\lceil \log t \rceil$.*

Because of our considerations above, the size t of a fooling set for f is a lower bound on the size of OBDDs representing f with respect to a variable order where the variables X_A are tested before the variables X_B. Because of the symmetric definition of fooling sets, t is also a lower bound on the size of OBDDs representing f with respect to a variable order where the variables X_B are tested before the variables X_A. The crucial thing to prove large lower bounds on the OBDD complexity of a function is to obtain for all partitions of the variables large lower bounds on the size of fooling sets for subfunctions of the given function (best case communication complexity).

Now we take a look at known results about the communication complexity of some functions. Let $\mathrm{EQ}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be defined by $\mathrm{EQ}_n(a,b) = 1$ iff the vectors $a = (a_1,\ldots,a_n)$ and $b = (b_1,\ldots,b_n)$ are equal. It is well-known and easy to prove that $C(\mathrm{EQ}_n) = n$. Obviously the same results can be obtained if Alice gets exactly one of the variables a_i and b_i, $1 \le i \le n$. Similar results can be obtained for the functions $\mathrm{GT}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ and $\overline{\mathrm{GT}_n} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, where $\mathrm{GT}_n(a,b) = 1$ iff $[a]_1^n > [b]_1^n$ and $\overline{\mathrm{GT}_n}(a,b) = 1$ iff $[a]_1^n \le [b]_1^n$.

In the rest of the section we investigate the following function $f \in B_{3n}$ which is defined on the variables $a = (a_1,\ldots,a_n)$, $b = (b_1,\ldots,b_n)$, and $c = (c_1,\ldots,c_n)$:

$$f_n(a,b,c) := (\mathrm{EQ}_n(a,\overline{c}) \wedge \overline{\mathrm{GT}_n}(a,b)) \vee \mathrm{GT}_n(a,\overline{c})$$

Our aim is to prove that there exists a fooling set with at least 2^n elements for the function f_n if Alice gets the a- and Bob the b-variables. With other words the communication complexity of f is not smaller than the communication complexity of $\overline{\mathrm{GT}_n}$ and the distribution of the c-variables does not simplify the task.

Proposition 1. *Let $S := \{(\alpha,\alpha,\overline{\alpha}) \mid \alpha \in \{0,1\}^n\}$, where the first component of a triple in S is an assignment to the a-variables, the second to the b-variables, and the third to the c-variables. S is fooling set of size 2^n for the function f_n.*

Proof. Obviously $f(\alpha,\alpha,\overline{\alpha}) = 1$ for $\alpha \in \{0,1\}^n$. Let $(\alpha_1,\alpha_1,\overline{\alpha_1})$ and $(\alpha_2,\alpha_2,\overline{\alpha_2})$ be two different elements in S, w.l.o.g. $[\alpha_1]_0^{n-1} > [\alpha_2]_0^{n-1}$. Let i be the most significant bit position where $[\alpha_1]_i \neq [\alpha_2]_i$.

Case 1: The variable c_i belongs to Bob.

Let $\overline{\alpha}$ be the composition of the partial assignment of $\overline{\alpha_2}$ to Alice's c-variables and the partial assignment of $\overline{\alpha_1}$ to Bob's c-variables.

The function value $f(\alpha_2, \alpha_1, \overline{\alpha})$ is 0 since $\mathrm{GT}_n(\alpha, \alpha_2) = 1$.

Case 2: The variable c_i belongs to Alice.

Let D be the set of all indices j where $[\alpha_1]_j \neq [\alpha_2]_j$.

Case 2.1: All variables c_j, $j \in D$, belong to Alice.

Let $\overline{\alpha}$ be the composition of the partial assignment of $\overline{\alpha_1}$ to Alice's c-variables and the partial assignment of $\overline{\alpha_2}$ to Bob's c-variables.

The function value $f(\alpha_1, \alpha_2, \overline{\alpha})$ is 0 since $\overline{\mathrm{GT}_n}(\alpha_1, \alpha_2) = 0$ and $\mathrm{EQ}_n(\alpha_1, \alpha) = 1$, therefore $\mathrm{GT}_n(\alpha_1, \alpha) = 0$.

Case 2.2 There are variables c_j, $j \in D$, that belong to Bob.

Let $D' \subseteq D$ be the set of indices j, where c_j belong to Bob and $i' := \max\{j \mid j \in D'\}$.

Case 2.2.1 $[\alpha_1]_{i'} = 0$ and $[\alpha_2]_{i'} = 1$

Let $\overline{\alpha}$ be the composition of the partial assignment of $\overline{\alpha_1}$ to Alice's c-variables and the partial assignment of $\overline{\alpha_2}$ to Bob's c-variables.

The function value $f(\alpha_1, \alpha_2, \overline{\alpha})$ is 0 since $\overline{\mathrm{GT}_n}(\alpha_1, \alpha_2) = 0$ and $\mathrm{GT}_n(\alpha_1, \alpha) = 0$.

Case 2.2.2 $[\alpha_1]_{i'} = 1$ and $[\alpha_2]_{i'} = 0$

Let $\overline{\alpha}$ be the composition of the partial assignment of $\overline{\alpha_2}$ to Alice's c-variables and the partial assignment of $\overline{\alpha_1}$ to Bob's c-variables.

The function value $f(\alpha_2, \alpha_1, \overline{\alpha})$ is 0 since $\mathrm{GT}_n(\alpha, \alpha_2) = 1$. □

The same result can be obtained if Alice gets exactly one of the variables a_i and b_i for all $i \in \{0, \ldots, n-1\}$. In this case it is not important whether the investigated c-variables belong to Alice or Bob but whether the considered a- and c-variables or b- and c-variables are tested together.

Using Theorem 4 we obtain the following result.

Corollary 1. *The communication complexity of the function f_n is at least n.*

3 A Larger Lower Bound on the OBDD Complexity of the Graph of Integer Multiplication

In this section we prove Theorem 2 and show that the OBDD size for the representation of MUL-Graph$_n$ is at least $2^{n/8}$. The crucial thing is to choose an appropriate subset of the input variables in order to show that there exists a large fooling set. Besides the larger lower bound the improvement to the result in [11] is that no case inspection is necessary.

Let π be an arbitrary variable order. We consider the set of the variables $S := \{x_{n-1}, \ldots, x_{n/2}, z_{n-1+n/2}, \ldots, z_n\}$. Let T be the set of the first $|T|$ variables according to π, where there are $n/2$ variables from S, and B be the set of the remaining variables. Let $X_{S,T}$ be the x-variables in $S \cap T$, $X_{S,B}$ the x-variables in $S \cap B$. Similar the sets $Z_{S,T}$ and $Z_{S,B}$ are defined. Using simple counting arguments we can prove that there exists a distance parameter d such that there exist at least $n/8$ pairs (x_i, z_{i+d}) in $X_{S,T} \times Z_{S,B} \cup X_{S,B} \times Z_{S,T}$ (for a similar

proof see, e.g., [1]). Let I be the set of indices i, $n/2 \leq i < n$, where x_i belongs to such a pair.

Now we replace some of the variables in the following way:

- y_d is replaced by 1 and the remaining y-variables are replaced by 0,
- the variables x_i, $i \notin I$, are replaced by 0, and
- the variables z_j, $0 \leq j \leq 2n-1$ and $j-d \notin I$, are replaced by 0.

The effect of these replacements is that the corresponding subfunction of MUL-Graph$_n$ is equal to the function EQ$_m$, $m \geq n/8$, where for each pair (x_i, z_{i+d}), $i \in I$, there is exactly one variable in T. Therefore, the OBDD size is at least $2^{n/8}$.

We only want to mention here that d has to be at least $n/8$ since there are at least $n/8$ pairs. In the next section this observation will be helpful in order to improve the lower bound on the OBDD complexity of MUL$_{2n-1,n}$.

4 A Larger Lower Bound on the OBDD Complexity of the Most Significant Bit of Integer Multiplication

In this section we prove Theorem 3 and determine the lower bound of $\Omega(2^{n/72})$ on the size of OBDDs for the representation of the most significant bit of integer multiplication. We start to prove a lower bound of $\Omega(2^{n/96})$ and present afterwards the idea how to improve this lower bound up to $\Omega(2^{n/72})$.

In the following for the sake of simplicity we do not apply floor or ceiling functions to numbers even when they need to be integers whenever this is clear from the context and has no bearing on the essence of the proof.

We start with a (simplified) presentation of our main proof idea. Our aim is to show for an arbitrary variable order π that a π-OBDD for MUL$_{2n-1,n}$ contains in a certain way a π-OBDD for the Boolean function f_n presented in Section 2:

$$f_n(a,b,c) = (\mathrm{EQ}_n(a,\overline{c}) \wedge \overline{\mathrm{GT}_n}(a,b)) \vee \mathrm{GT}_n(a,\overline{c}),$$

where the length of the inputs a, b, and c is $\Theta(n)$ and the a-variables are before the b-variables in π. Therefore, there exists a large fooling set and as a consequence also the size of the π-OBDD for MUL$_{2n-1,n}$ has to be large. The vectors a is a subvector of one of the inputs x and y for MUL$_{2n-1,n}$, the vectors b and c of the other input.

Besides the idea of the lower bound proof presented in Section 3 we use the idea of the following reduction from multiplication to squaring presented by Wegener [19], where squaring computes the square of an m-bit input. For two m-bit numbers u and w the number $z := u \cdot 2^{2m} + w$ is defined. Then

$$z^2 = u^2 \cdot 2^{4m} + u \cdot w 2^{2m} + w^2.$$

Since w^2 and $u \cdot w$ are numbers of length $2m$, the binary representation of the product uw can be found in the binary representation of z^2. (See Figure 1 shows the bit composition of the number z^2.)

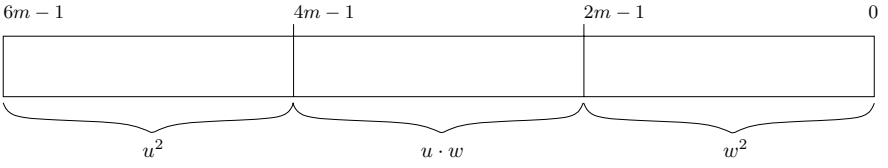


Fig. 1. The bit composition of the number z^2

A key observation is the following one.

$\mathrm{MUL}_{2n-1,n}$ answers the question whether the product of two numbers is at least 2^{2n-1}. For a number $2^{n-1} + \ell 2^{n/2}$, the corresponding smallest number such that the product of the two numbers is at least 2^{2n-1} is $2^n - \ell 2^{n/2+1} + 4\ell^2 - \left\lfloor \frac{4\ell^3}{2^{n/2-1}+\ell} \right\rfloor$. The number $\left\lfloor \frac{4\ell^3}{2^{n/2-1}+\ell} \right\rfloor$ is smaller than ℓ if $\ell \leq 2^{n/4-3/2}$. As a consequence if b_ℓ is the binary representation of ℓ, b_{ℓ^2} is the binary representation of ℓ^2, L the length of b_ℓ, and if there exists j, where $j \geq L-2$, and $[b_{\ell^2}]_j = 1$, there is no difference in the upper half of the binary representations of the numbers $4\ell^2$ and $4\ell^2 - \left\lfloor \frac{4\ell^3}{2^{n/2-1}+\ell} \right\rfloor$. More precisely, in this case if b' is the binary representation of $4\ell^2$ and b'' is the binary representation of $4\ell^2 - \left\lfloor \frac{4\ell^3}{2^{n/2-1}+\ell} \right\rfloor$, then $[b']_{j+1}^{2L+1} = [b'']_{j+1}^{2L+1}$.

Next, we investigate requirements that have to be fulfilled for inputs x and y, where $\mathrm{MUL}_{2n-1,n}(x,y) = 1$. If x represents a number $2^{n-1} + \ell 2^{n/2}$, $1 \leq \ell \leq 2^{n/4-3/2}$, the upper half of y has to represent a number of at least $2^{n/2} - 2\ell$, i.e., $[y]_{n/2}^{n-1} \geq 2^{n/2} - 2\ell$. If the upper half of y represents a number greater than $2^{n/2} - 2\ell$, the function value $\mathrm{MUL}_{2n-1,n}(x,y)$ is 1. Let j be the minimum number of the set $\{i \mid n/2 \leq i < (3/4)n - 2 \text{ and } x_i = 1\}$. If $[\overline{y}]_{i+2}^{n-1} > [x]_{i+1}^{n-1}$, the function value $\mathrm{MUL}_{2n-1,n}$ is 0. If $[\overline{y}]_{i+2}^{n-1} < [x]_{i+1}^{n-1}$, the function value $\mathrm{MUL}_{2n-1,n}$ is 1. If $y_{i+1} = 1$, $[\overline{y}]_{i+2}^{n-1} = [x]_{i+1}^{n-1}$, and $[y]_{n/2}^{i} = 0$, $[y]_0^{n/2-1}$ has to represent a number of at least $4\ell^2 - \left\lfloor \frac{4\ell^3}{2^{n/2-1}+\ell} \right\rfloor$.

In order to use Wegener's observation on squaring mentioned above combined with our lower bound proof presented in Section 3, we only consider integers ℓ where $\ell = u2^{2m} + w$, $u, w < 2^m$ and $m = n/12 - 1$. (Later on we show that m can be enlarged up to $n/9 - 2/3$ which leads to a larger lower bound.) For this reason we replace the variables $x_{n/2+m}, \ldots, x_{n/2+2m-1}$ by 0. (See Figure 2 for the composition of the number x.) Afterwards we replace some of the x-variables and the corresponding y-variables by constants, where y_{i+1} is the corresponding y-variable to x_i, such that a certain part of $u \cdot w$ is equal to a certain part of $2^d \cdot w$ for d appropriately chosen. Furthermore, we choose w in such a way that the assignments to the variables at position $3m+2, \ldots, 6m+1$ are the same in the binary representations of $4\ell^2$ and $4\ell^2 - \left\lfloor \frac{4\ell^3}{2^{n/2-1}+\ell} \right\rfloor$. Furthermore, for different numbers ℓ_1 and ℓ_2 (which means different assignments to the w-variables) the assignments to the variables at position $3m+2, \ldots, (7/2)m+1$ in the binary

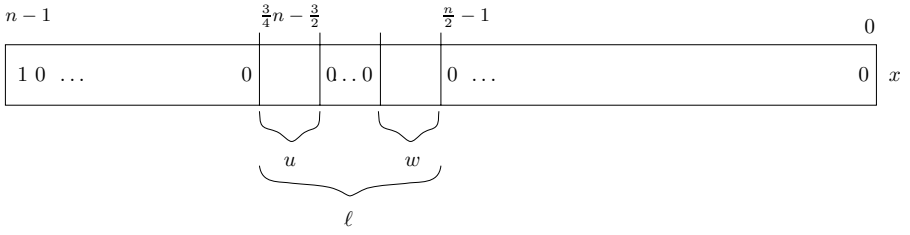


Fig. 2. The composition of the input x

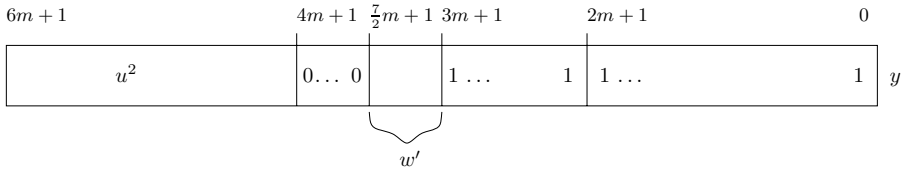


Fig. 3. The effect of the replacements of some of the y-variables, where $u = [u]_0^{m-1}$ (w' has to be at least $(2^d \cdot w)_m^{(3/2)m-1}$)

representations of $4\ell_1^2$ and $4\ell_2^2$ are different. (Figure 3 illustrates some of the replacements of the y-variables.)

Now we make our proof idea more precise. We rename $[x]_{n/2}^{n/2+(n/12)-2}$ by $[w]_0^{m-1}$ and $[x]_{n/2+n/6-2}^{n/2+n/4-4}$ by $[u]_0^{m-1}$. If $\ell = u \cdot 2^{2m} + w$ the product $u \cdot w$ can be found at position $2m+2, \ldots, 4m+1$ in the binary representation of $4\ell^2$. Our goal is to adapt the lower-bound proof for the graph of integer multiplication from Section 3 to the variables $S := \{w_{m/2}, \ldots, w_{m-1}, y_{3m+2}, \ldots, y_{(7/2)m+1}\}$. Let T be the set of the first $|T|$ variables according to π where there are $m/2$ variables from S. There exists a distance parameter d such that there are at least $m/8$ pairs $(w_i, y_{2m+2+i+d})$, where there is exactly one variable of each pair in T. Let I be the set of indices, where w_i belongs to such a pair. We replace the u-variables such that yields $[u]_0^{m-1} = 2^d$; similarly, the variables $y_{4m+2}, \ldots, y_{6m+1}$ are replaced such that $[y]_{4m+2}^{6m+1} = 2^{2d}$).

The variables $x_{n/2+i}$, $i \in I$, are called free x-variables, the variables $y_{n/2+i+1}$ and $y_{2m+2+i+d}$, $i \in I$, free y-variables. The free x-variables will play the role of the a-variables, the free variables $y_{n/2+i+1}$, $i \in I$, the role of the c-, and the remaining free y-variables the role of the b-variables in the reduction from the function f_n mentioned above to $\mathrm{MUL}_{2n-1,n}$. Now we present the reduction. (Figure 4 shows some of the replacements to the inputs x and y of $\mathrm{MUL}_{2n-1,n}$.)

- The variables y_{n-1} and x_{n-1} are set to 1,
- $x_{n/2+m-d-1}$ (which corresponds to w_{m-d-1}) and $y_{n/2+m-d}$ are set to 1,
- $x_{n/2+2m+d}$ (which corresponds to u_d) is set to 1, the corresponding variable $y_{n/2+2m+d+1}$ is set to 0, $y_{4m+2+2d}$ to 1, the variables $y_{(7/2)m+2}, \ldots, y_{4m+1+2d}$ and $y_{4m+3+2d}, \ldots, y_{6m+1}$ to 0 (as a result $[y]_{4m+2}^{6m+1} = 2^{2d}$).

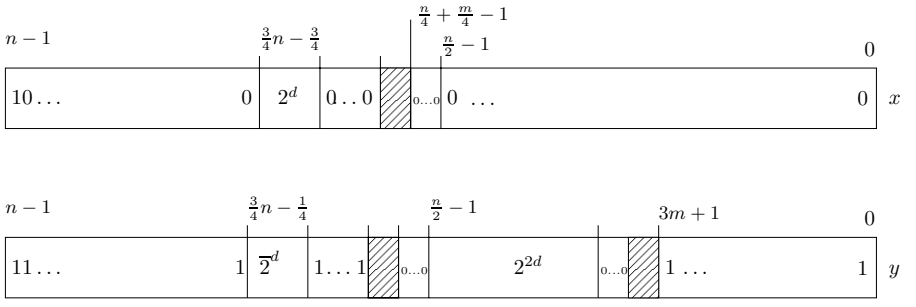


Fig. 4. A (simplified) presentation of replacements to some of the x- and y-variables. The shaded areas contain the free variables (and possibly other variables).

- The variables $y_{n/2}, \ldots, y_{n/2+m-d-1}$ are set to 0.
- Besides the free x-variables the remaining x-variables are replaced by 0.
- Besides the free y-variables the remaining y-variables are replaced by 1.

What is the effect of the replacements?

- The inputs x and y represent numbers that are at least 2^{n-1}, since otherwise the function value $\mathrm{MUL}_{2n-1,n}(x,y)$ is 0.
- Since $w_{m-d-1} = 1$ and $[u]_0^{m-1} = 2^d$, $4\ell^2$ and $4\ell^2 - \left\lfloor \frac{4\ell^3}{2^{n/2-1}+\ell} \right\rfloor$, where $\ell = u \cdot 2^{2m} + w$, do not differ in one of the bits at position $3m+2, \ldots, 6m+1$ of their binary representations.
- Since $x_{n/2+m-d-1} = 1$ and $y_{n/2+m-d} = 1$, $x_{n/2} = \ldots = x_{n/2+m-d-2} = 0$ and $y_{n/2} = \ldots = y_{n/2+m-d-1} = 0$, $[x]_{n/2+m}^{n-2} = [\overline{y}]_{n/2+m}^{n-2}$, $[x]_{n/2+m-d}^{n/2+m-1}$ has to be at least $[\overline{y}]_{n/2+m-d+1}^{n/2+m}$ for inputs x and y, where $\mathrm{MUL}_{2n-1,n}(x,y) = 1$. If $[x]_{n/2+m-d}^{n/2+m-1} > [\overline{y}]_{n/2+m-d+1}^{n/2+m}$, $\mathrm{MUL}_{2n-1,n}(x,y) = 1$.
- Since $[y]_{4m+2}^{6m+1} = 2^{2d} = u^2$ and because of the other replacements, $[y]_{3m+2}^{4m+1}$ has to be at least $(u \cdot w)_m^{2m-1}$ for inputs x and y, where $\mathrm{MUL}_{2n-1,n}(x,y) = 1$, if $[y]_{n/2}^{n-1} = 2^{n/2} - 2\ell$ and $[x]_{n/2}^{n-1} = 2^{n/2-1} + \ell$.

Therefore, the correctness of our reduction follows from our considerations above. Using Proposition 1 we obtain a fooling set of at least $2^{m/8}$ elements. Considering the fact that $m = n/12 - 1$, we get the result that the OBDD complexity of $\mathrm{MUL}_{2n-1,n}$ is at least $\Omega(2^{n/96})$.

Finally, we present the idea how to improve the lower bound on the OBDD complexity of $\mathrm{MUL}_{2n-1,n}$ up to $\Omega(2^{n/72})$. Up to now we have considered numbers ℓ, where $\ell = u \cdot 2^{2m} + w$ and $u, w < 2^m$ with $m = (n/12) - 1$. Using the fact that in our lower bound proof only the upper half of the bits in the binary representation of $u \cdot w$ is important, $u \cdot w$ div $2^{(3/2)m} = 0$, and $u^2 \mod 2^{m/4} = 0$, we can choose $\ell = u \cdot 2^{(5/4)m} + w$ and $u, w < 2^m$. As a result we can enlarge m up to $(n/9) - 2/3$.

5 Concluding Remarks

We only want to mention here that similar to the results presented in [20] the results presented in Section 3 and 4 can be extended to arbitrary oblivious binary decision diagrams of linear length.

The next challenge is to improve the lower bound on the OBDD complexity of $\mathrm{MUL}_{2n-1,n}$. The method presented in this paper seems to be not strong enough to enlarge the lower bound furthermore. Moreover, the complexity of $\mathrm{MUL}_{2n-1,n}$ for more general models than OBDDs is open.

References

1. Bryant, R.E.: On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. IEEE Trans. Computers 40(2), 205–213 (1991)
2. Woelfel, P.: Bounds on the OBDD-size of integer multiplication via universal hashing. J. Comput. Syst. Sci. 71(4), 520–534 (2005)
3. Bollig, B.: On the OBDD complexity of the most significant bit of integer multiplication. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 306–317. Springer, Heidelberg (2008)
4. Wegener, I.: Branching programs and binary decision diagrams: theory and applications. Society for Industrial and Applied Mathematics, Philadelphia (2000)
5. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers 35(8), 677–691 (1986)
6. Sieling, D., Wegener, I.: NC-algorithms for operations on binary decision diagrams. Parallel Processing Letters 3, 3–12 (1993)
7. Gentilini, R., Piazza, C., Policriti, A.: Computing strongly connected components in a linear number of symbolic steps. In: Proc. SODA, pp. 573–582 (2003)
8. Gentilini, R., Piazza, C., Policriti, A.: Symbolic graphs: Linear solutions to connectivity related problems. Algorithmica 50(1), 120–158 (2008)
9. Sawitzki, D.: Lower bounds on the OBDD size of graphs of some popular functions. In: Vojtáš, P., Bieliková, M., Charron-Bost, B., Sýkora, O. (eds.) SOFSEM 2005. LNCS, vol. 3381, pp. 298–309. Springer, Heidelberg (2005)
10. Woelfel, P.: Symbolic topological sorting with OBDDs. J. Discrete Algorithms 4(1), 51–71 (2006)
11. Bollig, B.: A note on the size of OBDDs for the graph of integer multiplication. Inf. Process. Lett. 109(2), 41–43 (2008)
12. Amano, K., Maruoka, A.: Better upper bounds on the QOBDD size of integer multiplication. Discrete Applied Mathematics 155(10), 1224–1232 (2007)
13. Bollig, B., Klump, J.: New results on the most significant bit of integer multiplication. In: Proc. ISAAC, pp. 883–894 (2008)
14. Bollig, B., Waack, S., Woelfel, P.: Parity graph-driven read-once branching programs and an exponential lower bound for integer multiplication. Theor. Comput. Sci. 362(1-3), 86–99 (2006)
15. Sauerhoff, M., Woelfel, P.: Time-space tradeoff lower bounds for integer multiplication and graphs of arithmetic functions. In: Proc. STOC, pp. 186–195 (2003)

16. Bollig, B.: On the OBDD complexity of the most significant bit of integer multiplication (full version). Theoretical Computer Science (invited to special issue)
17. Hromkovič, J.: Communication complexity and parallel computing. Springer, Heidelberg (1997)
18. Kushilevitz, E., Nisan, N.: Communication complexity. Cambridge University Press, New York (1997)
19. Wegener, I.: Optimal lower bounds on the depth of polynomial-size threshold circuits for some arithmetic functions. Inf. Process. Lett. 46(2), 85–87 (1993)
20. Gergov, J.: Time-space tradeoffs for integer multiplication on various types of input oblivious sequential machines. Inf. Process. Lett. 51(5), 265–269 (1994)

Picture Languages Generated by Assembling Tiles

Paola Bonizzoni, Claudio Ferretti, Anthonath Roslin Sagaya Mary, and Giancarlo Mauri

Dipartimento di Informatica Sistemistica e Comunicazione, Università degli Studi di Milano – Bicocca, Viale Sarca 336, 20126 Milano, Italy
{anthonath,bonizzoni,ferretti,mauri}@disco.unimib.it

Abstract. We propose a new formalism for generating picture languages based on an assembly mechanism of tiles that uses rules having a context and a replacement site. More precisely, a picture language will be generated from a finite set of initial pictures by iteratively applying rewriting rules from a given finite set of rules, called a *tiling rule system* (TRuS system). We prove that the TRuS systems have a greater generative capacity than the tiling systems of Giammarresi and Restivo, even in the case of one-letter alphabet picture languages. This is due mainly to the use of the notion of replacement.

1 Introduction

Recently the investigation of two dimensional (picture) languages has moved towards the definition of formal models capable of characterizing special classes of languages that are not included in the family of recognizable languages generated by tiling systems of Giammaresi and Restivo [10]. An example of such models is tile rewriting grammars (TRG) defined in [7] and further investigated in [6]. Indeed, while tiling systems represent an extension to the two dimensional case of regular string grammars, TRG provide an analogue of context-free grammars in the two dimensions, thus showing the capability of this approach of generating interesting picture languages that generalize context-free string languages, including for example Dyck languages. Grammar approaches, besides tiling systems and cellular automata (see [10] for a complete survey) reflect the efforts done towards a generalization of classical formal language theory to the two dimensional case. This research direction is now a rich field of investigation (see [1], [2], [11] and [3] as an example).

In this paper, our investigation of picture languages goes in a different direction, since we propose new operations that are not generalization of classical formal language concepts, but are instead inspired by operations used in modelling DNA self assembly [12]. More precisely, our approach for generating pictures is based on a notion of *tiling rule system*, consisting of an initial finite set of pictures and a set of rules that can be iteratively applied to the initial language to generate a picture language.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 224–235, 2009.

A rule consists of a pair of tiles: a *context site* and a *replacement site* tile. Context site is used to specify where the rule can be applied, while the replacement site is used to change part of the context site. This type of rule generalizes to the 2-dimensional case a typical behavior of rules acting on DNA strings, i.e. a context is needed to allow the applications of rules, while replacement specifies how the context will be modified.

In a tiling rule system, at each step a set of rules is simultaneously applied to a picture from the initial language or assembled in a previous computation step. The effect of the simultaneous application of rules is the replacement and insertion of a row or column, respectively, so allowing the growth of a new picture according to the rule system.

Formally a tiling rule system, TRuS system in short, is a triple (I, R, Σ), where I is an initial finite set of pictures, R is a finite set of tiling rules and Σ is the alphabet of the generated pictures.

Observe that our notion of tiling rule system is different from tiling systems, but also from Wang systems [8], which model DNA self assembly by pure growth and which are proved to be equivalent to tiling systems. In tiling systems by Giammaresi and Restivo (see Section 2), picture languages are defined by a projection function applied to a local language defined by a set of tiles.

We show that TRuS systems have greater generative capacity than the tiling systems, even in the case of systems generating one-letter alphabet picture languages. More precisely, the constructive proof of a TRuS system that simulates a tiling system shows that recognizable languages are generated by rules that act always by growing pictures along their borders. On the contrary the class of languages generated by TRuS systems includes languages that seem to strictly require rules acting on specific positions inside the pictures in order to grow the language. This is for example the case of the language of palindromic columns, that has been proved not being a recognizable language in [7].

The paper is organized as follows. In Section 2 preliminaries on pictures languages are given, while in Section 3 tiling rule systems are introduced. Then Section 4 is devoted to the investigation of the computational power of tiling rule systems and to a comparison with recognizable picture languages.

2 Preliminaries

Let Σ be a finite alphabet, let $\Delta = \{\#, \diamond\}$ be a set containing two *special* boundary symbols and such that $\Sigma \cap \Delta = \emptyset$.

A *picture* p over Σ is a rectangular array of elements in Σ. The *size* of the picture p is a pair (n, m) where n is the number of rows and m the number of columns of p. The i-th row, the j-th column and the element belonging to both of them in picture p will be denoted by $p^r[i]$, $p^c[j]$ and $p[i, j]$ (or p_{ij}) respectively. Moreover, by $p^r[i..i+k]$ ($p^c[i..i+k]$, respectively) we denote the sub-array of p consisting of the rows (columns, respectively) of p from index i to $i+k$.

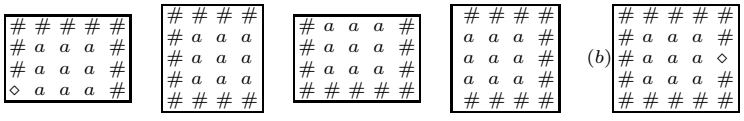
The only picture of size $(0, 0)$ is the *empty picture*, denoted by λ. Then Σ^{++} denotes the set of all nonempty pictures over Σ, and define $\Sigma^{**} = \Sigma^{++} \cup \{\lambda\}$.

The *bordered* version of a picture p of size (n, m) is the array $\hat{p}$ of size $(n + 2, m+2)$ obtained by surrounding p with special symbols in Δ. Since the alphabet Δ consists of two symbols, we call *canonical* pictures those bordered uniquely with the boundary symbol #.

Given a picture p of size (n, m), a *partial bordered* version of p, or simply a *partial picture*, is the picture $\bar{p}$ of size either $(n', m + 2)$ or $(n + 2, m')$ for $n' = n + 1, m' = m + 1$, obtained from p by adding borders partially along the picture (see Example 1).

In the paper by *pseudo-canonical* picture we mean a picture that is completely bordered using also the symbol $\diamond$ or is partially bordered obtained by a picture p(without border). The notation $\hat{p}$ will be used for *pseudo-canonical* or canonical pictures that are obtained from picture p and partially bordered pictures $\bar{p}$. Moreover, observe that the size of a pseudo-canonical picture is the size of the array over the extended alphabet $\Delta \cup \Sigma$.

Example 1. *Pseudo-canonical pictures : partially bordered pictures over one letter alphabet and (b) a completely bordered picture including the $\diamond$ symbol:*



A *sub-picture* $\hat{p}'$ of a (pseudo-canonical) picture $\hat{p}$ is a picture that is a sub-array of $\hat{p}$. Given picture $\hat{p}$ then $B_{h,k}(\hat{p})$ denotes the set of sub-pictures of size (h, k). A *tile* is a $(2, 2)$ picture over the alphabet $\Sigma \cup \Delta$, where tiles containing symbols from alphabet Δ are called *border tiles*. In this paper a tile is denoted by $t = \begin{smallmatrix} a & b \\ c & d \end{smallmatrix}$.

A *picture language* L consists of a subset of Σ^{**}. L is *local* if there exists a finite set Θ of tiles over alphabet $\Sigma \cup \{\#\}$ such that $L = \{p \in \Sigma^{**} | B_{2,2}(\hat{p}) \subseteq \Theta$ where $\hat{p}$ are canonical pictures $\}$. Then L is the local language defined by Θ.

Let us now recall the notion of a tiling system and language generated by such system [10].

Definition 1 (Tiling System). *A tiling system is a 4-tuple $\tau = (\Sigma, \Gamma, \Theta, \pi)$, where Σ and Γ are two finite alphabets, Θ is a finite set of tiles over the alphabet $\Gamma \cup \{\#\}$ and $\pi : \Gamma \to \Sigma$ is a projection.*

Given system τ, the language defined by the system, denoted $L(\tau)$, is the projection by π of the local language defined by Θ.

In the paper we denote by $\mathcal{L}$(TS) the class of languages defined by tiling systems, known as the class of recognizable languages that has been deeply investigated [10].

3 Tiling Rule Systems

In this section, we define the notion of tiling rule and tiling rule system.

A *general tiling rule* r over a set Θ of tiles is defined by a pair t_1, t_2 of tiles in Θ, where t_1 is the *context site* of rule r, while t_2 is the *replacement site* of rule r. Then r is denoted as $r = t_1 \rightarrow t_2$. We distinguish two types of rules: *row* and *column* rules. The context site of a row rule is denoted by a tile $t = \frac{a\ b}{c\ d}$, while the context site of column rules is denoted by a tile $t = {}^{a}_{c}|^{b}_{d}$.

When a row rule r is applied, then t_2 replaces the bottom row domino of tile t_1. Similarly, when a column rule is applied then t_2 replaces the rightmost column domino of tile t_1. A rule acts together with other rules to enlarge a picture: this fact is formalized by the notion of *rule sequence* defined below.

Definition 2 (row rule sequence). *A sequence $S = r_1 \cdot r_2 \cdot \ldots \cdot r_m$ of rules is a* row rule sequence*, in short* r-sequence*, iff for each j, $1 \leq j \leq m$, it holds that* $r_j = \frac{a_j\ a_{j+1}}{b_j\ b_{j+1}} \rightarrow^{a'_j\ a'_{j+1}}_{b'_j\ b'_{j+1}}$.

Given a row rule sequence S in the above definition 2, the application of rules in S defines the pseudo-pictures $p_{(S,r)}$ and $q_{(S,r)}$ consisting of 2 rows and $m+1$ columns called the *context site* and *replacement site* of S respectively, such that $p_{(S,r)}[1,j] = a_j$, $p_{(S,r)}[2,j] = b_j$, while $q_{(S,r)}[1,j] = a'_j$ and $q_{(S,r)}[2,j] = b'_j$ for each column j.

Definition 3 (column rule sequence). *A sequence $S = r_1 \cdot r_2 \cdot \ldots \cdot r_n$ of rules is a* column rule sequence*, in short* c-sequence*, iff for each i, $1 \leq i \leq n$, it holds that* $r_i = {}^{a_i}_{a_{i+1}}|^{b_i}_{b_{i+1}} \rightarrow^{a'_i\ b'_i}_{a'_{i+1}b'_{i+1}}$.

Given a column rule sequence S in the above definition 3, the application of S produces the pseudo-pictures $p_{(S,c)}$ and $q_{(S,c)}$ consisting of $n+1$ rows and 2 columns called the *context site of S* and *replacement site of S* respectively, such that $p_{(S,c)}[i,1] = a_i$, $p_{(S,c)}[i,2] = b_i$, while $q_{(S,c)}[i,1] = a'_i$, $q_{(S,c)}[i,2] = b'_i$ for each row i.

Example 2. *Let* $r_1 = \frac{a\ b}{e\ f} \rightarrow^{a'\ b'}_{e'\ f'}$, $r_2 = \frac{b\ c}{f\ g} \rightarrow^{b'\ c'}_{f'\ g'}$ *and* $r_3 = \frac{c\ d}{g\ h} \rightarrow^{c'\ d'}_{g'\ h'}$ *be row rules. Then $S = r_1 \cdot r_2 \cdot r_3$ is a r-sequence, picture* $p_{(S,r)} = \boxed{\begin{matrix} a & b & c & d \\ e & f & g & h \end{matrix}}$ *is the context site of S, while picture* $q_{(S,r)} = \boxed{\begin{matrix} a' & b' & c' & d' \\ e' & f' & g' & h' \end{matrix}}$ *is the replacement site of S.*

Example 3. *Let* $r_1 = {}^{a}_{b}|^{e}_{f} \rightarrow^{a'\ e'}_{b'\ f'}$, $r_2 = {}^{b}_{c}|^{f}_{g} \rightarrow^{b'\ f'}_{c'\ g'}$, *and* $r_3 = {}^{c}_{d}|^{g}_{h} \rightarrow^{c'\ g'}_{d'\ h'}$ *be column rules. Then $S = r_1 \cdot r_2 \cdot r_3$ is a c-sequence, such that the context site of S is* $p_{(S,c)} = \boxed{\begin{matrix} a & e \\ b & f \\ c & g \\ d & h \end{matrix}}$ *and the replacement site of S is* $q_{(S,c)} = \boxed{\begin{matrix} a' & e' \\ b' & f' \\ c' & g' \\ d' & h' \end{matrix}}$.

Now rule sequences can be applied to canonical or pseudo-canonical pictures, i.e. pictures either bordered or bordered with special symbol $\diamond$ or partially bordered.

Definition 4 (application of row rule sequences). *Let $S = r_1 \cdot r_2 \cdot \ldots \cdot r_{m+1}$ be a r-sequence having context site $p_{(S,r)}$ and replacement site $q^r_{(S,r)}[1] = [a'_1 \ldots a'_{m+2}]$, $q^r_{(S,r)}[2] = [b'_1 \ldots b'_{m+2}]$. Then S can be applied to the picture $\hat{p}$ obtained from pictures $p, \bar{p}$, of sizes (n,m), $(n+1, m+2)$ respectively, at the i-th row of $\hat{p}$, where $1 \leq i \leq n$ iff the following holds:*

1. $\hat{p}^r[i..i+1] = p_{(S,r)}$ *and*
2. $[a'_2...a'_{m+1}]$ *is over* Σ *for row index* i, *with* $1 \leq i \leq n$, *while* $[b'_2...b'_{m+1}]$ *is over* Σ *when* $1 \leq i < n$.

Definition 5 (application of column rule sequences). *Let* $S = r_1 \cdot r_2 \cdot ... \cdot r_{n+1}$ *be a c-sequence having context site* $p_{(S,c)}$ *and replacement site* $q^c_{(S,r)}[1] = [a'_1...a'_{n+2}]$, $q^c_{(S,r)}[2] = [b'_1...b'_{n+2}]$. *Then* S *can be applied to the picture* $\hat{p}$ *obtained from pictures* $p, \bar{p}$ *of sizes* (n, m), $(n+2, m+1)$ *respectively, at the* i*-th column of* $\hat{p}$, *where* $1 \leq i \leq m$ *iff the following holds:*

1. $\hat{p}^c[i..i+1] = p_{(S,c)}$ *and*
2. $[a'_2...a'_{n+1}]$ *is over* Σ *for row index* i, *with* $1 \leq i \leq m$, *while* $[b'_2...b'_{n+1}]$ *is over* Σ *when* $1 \leq i < m$.

Observe that condition 2 in the above definitions 4 and 5 is given to forbid the insertion of border special symbols inside a picture. The effect of the application of S to picture $\hat{p}$ is specified by the replacement site $q_{(S,r)}$ of S as shown in Figure 1 and 2.

Let's formally define picture $\hat{q}$ obtained by the application of rule sequences S to $\hat{p}$.

Definition 6 (row derived picture). *Let* S *be a r-sequence such that* S *can be applied at the* i*-th row of a picture* $\hat{p}$ *of size* $n'' \times m''$ *obtained from the pictures* $p, \bar{p}$ *of sizes* $(n, m), (n+1, m+2)$ *respectively. Then the picture* $\hat{q}$ derived from $\hat{p}$ by S *is such that:*

$\hat{q}^r[1..i] = \hat{p}^r[1..i]$, $\hat{q}^r[i+1..i+2]$ *is equal to the replacement site* $q_{(S,r)}$ *of* S, *and* $\hat{q}^r[i+3..n''+1] = \hat{p}^r[i+2..n'']$.

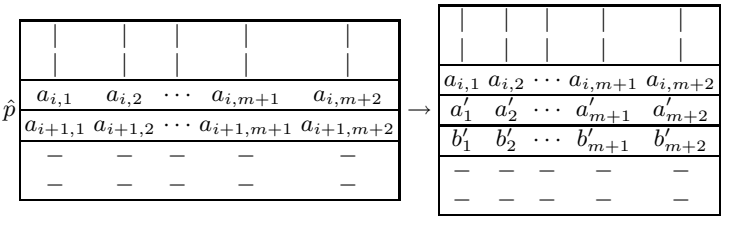


Fig. 1. Application of S to the i-th row of $\hat{p}$

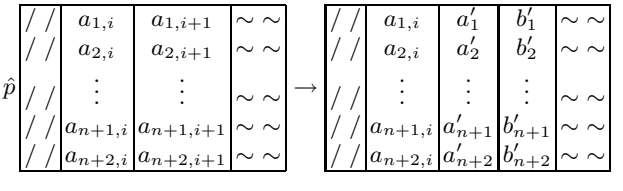


Fig. 2. Application of S to the i-th column of $\hat{p}$

Similarly, we define:

Definition 7 (column derived picture). *Let S be a c-sequence such that S can be applied to a picture $\hat{p}$ of size $n'' \times m''$ obtained from the pictures $p, \bar{p}$ of sizes $(n, m), (n+2, m+1)$ respectively. Then the picture $\hat{q}$* derived from *$\hat{p}$ by S is such that:*

$\hat{q}^c[1..i] = \hat{p}^c[1..i]$, $\hat{q}^c[i+1..i+2]$ is equal to the replacement site $q_{(S,c)}$ of S, and $\hat{q}^c[i+3..m''+1] = \hat{p}^c[i+2..m'']$.

The application of S to $\hat{p}$ to derive picture $\hat{q}$ is denoted by $\hat{p} \rightarrow_S \hat{q}$. The i-iterated application of S over a picture $\hat{p}$ to generate picture $\hat{q}$ is denoted by $\hat{p} \rightarrow_S^i \hat{q}$. A picture $\hat{p}'$ is *derived* from a picture $\hat{p}$, denoted by $\hat{p} \Rightarrow_R \hat{p}'$, iff there exist rule sequences $S_1, \cdots, S_k$ such that $\hat{p} \rightarrow_{S_1} \hat{p}_1 \rightarrow_{S_2} \hat{p}_2 ... \rightarrow_{S_k} \hat{p}'$. Then $\hat{p} \rightarrow_{S_1} \hat{p}_1 \rightarrow_{S_2} \hat{p}_2 ... \rightarrow_{S_k} \hat{p}'$ is called *derivation of $\hat{p}'$ from $\hat{p}$*, while $d = S_1, \cdots, S_k$ is the *derivation sequence* applied to derive $\hat{p}'$ from $\hat{p}$.

A derivation sequence $d = S_1, \cdots, S_k$ is called *unambiguous* iff for each i, with $1 \leq i < k$, the context site of S_{i+1} is the replacement site of S_i. A derivation sequence is *deterministic* if given sequence S_i it can be only followed by sequence S_{i+1}.

Given an initial finite set of pictures and a finite set of rules, then rules can be combined to produce c-sequences or r-sequences that can be applied iteratively to the initial pictures to generate an infinite language of pictures. This process of generating pictures is described by the notion of a tiling rule system and language generated by such type of systems.

Definition 8 (Tiling Rule System (TRuS)). *A tiling rule system, in short TRuS system, is a quadruple $T = (I, R, \Sigma, \Delta)$ where I is a finite set of canonical pictures, called* initial set *and R is a set of rules over alphabet $\Sigma \cup \Delta$, where Δ is a special two symbols alphabet disjoint from Σ.*

Thus let us define the language generated by a TRuS system.

Definition 9 (language). *Given a TRuS system T, then the language generated by T, denoted by $L(T)$ is the set $\{p : \hat{p} \in L\}$, where $L = I \cup \{\hat{p_1} : \hat{p} \Rightarrow_R \hat{p_1}, \hat{p} \in I, \hat{p_1}$ is canonical $\}$. Language L is the* canonical *language generated by the system.*

Then $\mathcal{L}$(TRuS) denotes the class of languages generated by TRuS systems.

Remark 1. *Assume that $p' \in L(T)$, $\hat{p} \rightarrow_{S_1} \hat{p_1} \rightarrow_{S_2} \hat{p_2} \rightarrow ... \rightarrow_{S_{k-1}} \hat{p}_{k-1} \rightarrow_{S_k} \hat{p}'$, where $\hat{p} \in I$. Observe that by definition 9, the intermediate pictures $\hat{p}_i$, with $1 \leq i < k$ are not necessarily canonical pictures, but are pseudo-canonical ones.*

4 Computational Power of TRuS Systems

In this section we investigate the computational power of TRuS system. Now, the class of languages generated by TRuS system properly includes the one of

recognizable languages. Indeed, we first show by Theorem 1 that recognizable languages are generated by TRuS systems.

Then we show that the generative capacity of tiling rule systems is greater than the tiling systems of Giammarresi and Restivo even in the unary case. Indeed, the unary language L consisting of pictures of size $(n, n!)$ is generated by tiling rule systems, while, as stated in [9], language L is not recognizable. A proof of this result is given in Proposition 1.

The strict inclusion also follows by the fact that it has been proved in [7] that the language of palindromic columns is not in the class $\mathcal{L}(\mathrm{TS})$ while it is generated by TRuS systems (the proof of this proposition can be found in [4]). This example of language L has been given to show the more powerful generative capacity of Tiling Rewriting Grammars (TRG) w.r.t. tiling systems.

Theorem 1. *$\mathcal{L}(TS) \subseteq \mathcal{L}(TRuS)$.*

Proof. Let L be a recognizable language and let $\tau = (\Sigma, \Gamma, \Theta, \pi)$ be a tiling system for L where Σ, Γ are finite alphabets, Θ is a set of tiles over $(\Gamma \cup \#)$ and $\pi : \Gamma \longrightarrow \Sigma$ is a projection. Let us define the following binary relations $=_r$ and $=_c$ over pairs of tiles in Θ: $t_1 =_c t_2$ iff $t_1 = \frac{a_1 a_2}{a_3 a_4}, t_2 = \frac{b_1 b_2}{b_3 b_4}$ where $a_2 = b_1, a_4 = b_3$, $t_1 =_r t_2$ iff $t_1 = \frac{b_1 b_2}{b_3 b_4}, t_2 = \frac{c_1 c_2}{c_3 c_4}$ where $b_3 = c_1, b_4 = c_2$.
In the following we define a TRuS -system $T = (I, R, A, \Delta)$ for generating L, where I consists of the empty picture $\begin{smallmatrix}\# & \# \\ \# & \#\end{smallmatrix}$ and alphabet A consists of $\Sigma \cup \Gamma \cup A'$, being $A' = \{[a\ b] : a, b \in \Gamma\}$.

Now, the set R of rules are listed below and are grouped according to the pair of tiles in Θ related by the relations $=_c$ and $=_r$, respectively. Indeed, rules should reproduce the tiling of a local picture and at the same time the projection of the local language. In order to do so, given a pair of tiles t_1, t_2 such that $t_1 =_r t_2$, we build a row rule r having the replacement site given by tile t such that the upper row of t (i.e. $t^r[1]$) will project the upper row domino of tile t_2, while the bottom row of t (i.e. $t^r[2]$) "memorizes" by using symbols in A' the tile t_2 that will have the upper row projected. Similarly, for the context-site of rule r. More precisely, given a pair of tiles where $\frac{ab}{cd} =_r \frac{cd}{ef}$ then we should build a row rule of the form $\frac{\pi(a)\pi(b)}{[a\ c][b\ d]} \rightarrow \frac{\pi(c)\pi(d)}{[c\ e][d\ f]}$. Similarly we build rules for pair of tiles related by the $=_c$ relation. Clearly, when tiles have border symbols then the construction of rules will be specific, as described below.

In the following column rules are grouped into the set $R_{c,U} = R_{c,U,1} \cup R_{c,U,2} \cup R_{c,U,3} \cup R_{c,U,4}$ of column rules that are derived by every pair of border tiles in Θ related by the binary relation $=_c$.

$$R_{c,U,1} = \{\tfrac{\#|\#}{\#|\#} \rightarrow \tfrac{\#\#}{a\ \#} : \tfrac{\#\#}{\#\ a} =_c \tfrac{\#\ \#}{a\ \#}\},\quad R_{r,L,1} = \{\tfrac{\#\#}{\#a} \rightarrow \tfrac{\#\pi(a)}{\#\ \#} : \tfrac{\#\#}{\#a} =_r \tfrac{\#a}{\#\#}\},$$
$$R_{c,U,2} = \{\tfrac{\#|\#}{\#|\#} \rightarrow \tfrac{\#\ \diamond}{a\ [a\ b]} : \tfrac{\#\#}{\#\ a} =_c \tfrac{\#\#}{a\ b}\},\quad R_{r,L,2} = \{\tfrac{\#\#}{\#a} \rightarrow \tfrac{\#\pi(a)}{\diamond\ [a\ b]} : \tfrac{\#\#}{\#a} =_r \tfrac{\#a}{\#b}\},$$
$$R_{c,U,3} = \{\tfrac{\#|\diamond}{a|[a\ b]} \rightarrow \tfrac{\#\ \diamond}{b\ [b\ c]} : \tfrac{\#\#}{a\ b} =_c \tfrac{\#\#}{b\ c}\},\quad R_{r,L,3} = \{\tfrac{\#\pi(a)}{\diamond\ [a\ b]} \rightarrow \tfrac{\#\pi(b)}{\diamond\ [b\ c]} : \tfrac{\#a}{\#b} =_r \tfrac{\#b}{\#c}\},$$
$$R_{c,U,4} = \{\tfrac{\#|\diamond}{a|[a\ b]} \rightarrow \tfrac{\#\#}{b\ \#} : \tfrac{\#\#}{a\ b} =_c \tfrac{\#\#}{b\ \#}\},\quad R_{r,L,4} = \{\tfrac{\#\pi(a)}{\diamond\ [a\ b]} \rightarrow \tfrac{\#\pi(b)}{\#\ \#} : \tfrac{\#a}{\#b} =_r \tfrac{\#b}{\#\#}\},$$

$R_{r,R,1}= \{\frac{\#\#}{a\ \#} \rightarrow \frac{\pi(a)\#}{\#\ \#} : \frac{\#\#}{a\#} =_r \frac{a\#}{\#\#}\}$,$R_{r,M,1}= \{\frac{\#\#}{ab} \rightarrow \frac{\pi(a)\pi(b)}{\#\ \#} : \frac{\#\#}{ab} =_r \frac{a\ b}{\#\#}\}$,

$R_{r,R,2}= \{\frac{\#\#}{a\ \#} \rightarrow \frac{\pi(a)\#}{[a\ b]\ \#} : \frac{\#\#}{a\#} =_r \frac{a\#}{b\#}\}$, $R_{r,M,2}= \{\frac{\#\#}{ab} \rightarrow \frac{\pi(a)\pi(b)}{[a\ c]\ [b\ d]} : \frac{\#\#}{ab} =_r \frac{a\ b}{c\ d}\}$,

$R_{r,R,3}= \{\frac{\pi(a)\#}{[a\ b]\ \#} \rightarrow \frac{\pi(b)\#}{[b\ c]\ \#} : \frac{a\#}{b\#} =_r \frac{b\#}{c\#}\}$, $R_{r,M,3}= \{\frac{\pi(a)\pi(b)}{[a\ c]\ [b\ d]} \rightarrow \frac{\pi(c)\pi(d)}{[c\ e]\ [d\ f]} : \frac{ab}{cd} =_r \frac{c\ d}{e\ f}\}$,

$R_{r,R,4}= \{\frac{\pi(a)\#}{[a\ b]\ \#} \rightarrow \frac{\pi(b)\#}{\#\ \#} : \frac{a\#}{b\#} =_r \frac{b\#}{\#\#}\}$,$R_{r,M,4}= \{\frac{\pi(a)\pi(b)}{[a\ c]\ [b\ d]} \rightarrow \frac{\pi(c)\pi(d)}{\#\ \#} : \frac{ab}{cd} =_r \frac{c\ d}{\#\#}\}$.

Note that rules in $R_{c,U}$ only form c-sequences of length one that can be combined to form derivation sequences that apply to the initial empty picture to reproduce the tiling of the uppermost rows of a picture in L.

Then row rules are grouped into three sets. The set $R_{r,L} = R_{r,L,1} \cup R_{r,L,2} \cup R_{r,L,3} \cup R_{r,L,4}$ groups rules, called *l-type rules*, that are based on pair of left most column (bordered) tiles , while set $R_{r,R} = R_{r,R,1} \cup R_{r,R,2} \cup R_{r,R,3} \cup R_{r,R,4}$ groups rules, called *r-type rules*, that are based on rightmost pairs of (bordered) tiles in Θ. Finally, set $R_{r,M} = R_{r,M,1} \cup R_{r,M,2} \cup R_{r,M,3} \cup R_{r,M,4}$ groups rules, called *intermediate rules or m-type rules*, that are based on tiles that are non bordered on the left and right column dominoes and that form r-sequences starting with a l-type rule and ending with a r-type rule.

Now, assume that $p \in L$ is a picture of size (n, m) that is the projection of a local picture q. Then we show how the picture is generated by the iterations of the above defined rules.

Assume first that $n = m = 1$. Assume that $p = \pi(a)$, for $a \in \Gamma$. Then, there exists a rule r_0, with $r_0 \in R_{c,U,1}$ that applies to the empty picture to generate the pseudo-canonical picture $\hat{q}^r[1..2]$ with the single element a. Then, there exists a r-sequence $r_1 \cdot r_2$ with $r_1 \in R_{r,L,1}$ and $r_2 \in R_{r,R,1}$ such that is applied at the context site given by the pseudo-canonical picture $\hat{q}^r[1..2]$ producing the canonical picture $\hat{p}$.

Assume now that $n = 1$ and $m > 1$. We first prove the generation of the pseudo-canonical picture $\hat{q}^r[1..2]$ (the picture made by the first 2 rows of $\hat{q}$). In the following define tile $t_{i,j}$ as the sub-picture of $\hat{q}$ of size $(2, 2)$ with the first element consisting of $\hat{q}_{i,j}$. Clearly, it holds that $t_{1,j} =_c t_{1,j+1}$ for each j, $1 \leq j < m + 2$ and thus there exists a column rule $r_j \in R_{c,U}$ corresponding to this pair of tiles, where $r_j \in R_{c,U,2}$ if $j = 1$ and $r_j \in R_{c,U,4}$ if $j = m + 1$, otherwise $r_j \in R_{c,U,3}$. It follows that the derivation sequence d consisting of the c-sequences $S_1, S_2, ..., S_{m+1}$, where $S_j = r_j$ will produce the tiling of $\hat{q}^r[1..2]$ as required.

Once $\hat{q}^r[1..2]$ is generated, then it is easy to verify the existence of rules $r_j^0 \in R_{r,M,1}$ for $1 < j < m + 1$, based on the pair of tiles $t_{2,j} =_r t_{3,j}$ that will produce the projection of symbols in tile $t_{2,j}$. Similarly, there is a rule $r_1^0 \in R_{r,L,1}$ based on the pair of tiles $t_{2,1} =_r t_{3,1}$ and rule $r_{m+1}^0 \in R_{r,R,1}$ corresponding to the pair of tiles $t_{2,m+1} =_r t_{3,m+1}$ that will produce the projection of the first and last symbol of the picture $\hat{q}^r[1..2]$.

Then the r-sequence $r_1^0 \cdot r_2^0 \cdot r_j^0 \cdot r_{m+1}^0$ is applied to picture $\hat{q}^r[1..2]$ having the required context site, thus producing the canonical picture $\hat{p}^r[1..3]$.

Assume now that $n > 1$ and $m \geq 1$. We first list the rule sequences that will be used to generate the rows from 1 to $n - 1$ of picture $\hat{p}$. Let us recall that

for each indexes i, j, with $1 \leq i \leq n+1$ and $1 \leq j \leq m+1$ there are tiles $t_{ij} =_r t_{i+1,j}$. Then, we distinguish two cases.

Case 1: assume $i = 1$. Then by construction there exists rule r_j^1 in $R_{r,M,2}$ corresponding to such pair of tiles for $j \neq 1$ and $j \neq m+1$ and rules r_1^1, r_{m+1}^1 respectively in $R_{r,L,2}$ and $R_{r,R,2}$, such that the r-sequence $S_{2,m} = r_1^1 \cdot r_2^1 \cdots r_j^1 \cdots r_{m+1}^1$ will produce the context site picture consisting of $\hat{q}^r[1..2]$ and the replacement site consisting of the two rows $[\#, \pi(q_{11}), \cdots, \pi(q_{1m}), \#]$ and $[\diamond, [q_{11}\ q_{21}], \cdots, [q_{1m}\ q_{2m}], \#]$.

Case 2: assume that $1 < i < n$. Recalling that for each index j, $1 \leq j \leq m+1$ there are tiles $t_{ij} =_r t_{i+1,j}$, by construction there exists a m-type rule r_j^i in $R_{r,M,3}$ corresponding to such pair of tiles for $j \neq 1$ and $j \neq m+1$. Moreover, there exists a l-type rule $r_1^i \in R_{r,L,3}$ and a r-type rule $r_{m+1}^i \in R_{r,R,3}$ corresponding to the above pair of tiles for $j = 1$ and $j = m+1$, respectively.

It is immediate to verify that the r-sequence $S_{3,i,m} = r_1^i \cdot r_2^i \cdots r_j^i \cdots r_{m+1}^i$ produces the context site picture with rows $[\#, \pi(q_{i1}), \cdots, \pi(q_{im}), \#]$, $[\diamond, [q_{i1}\ q_{i+1,1}], \cdots, [q_{im}\ q_{i+1,m}], \#]$ and the replacement site picture with rows $[\#, \pi(q_{i+1,1}), \cdots, \pi(q_{i+1,m}), \#]$, $[\diamond, [q_{i+1,1}\ q_{i+2,1}], \cdots, [q_{i+1,m}\ q_{i+2,m}], \#]$.

Now, let us show the construction of $\hat{p}$ using the above specified rule sequences $S_{2,m}$ and $S_{3,i,m}$.

The first step will consist of the generation of the pseudo-picture $\hat{p}_0 = \hat{q}^r[1..2]$ which has been detailed above for the case $n = 1, m > 1$. Then, as a second step, the r-sequence $S_{2,m}$ is applied to picture $\hat{p}_0$ to generate picture $\hat{p}_1$. The derivation $\hat{p}_1 \rightarrow_{S_{3,1,m}} \rightarrow_{S_{3,2,m}} \cdots \rightarrow_{S_{3,i,m}} \cdots \rightarrow_{S_{3,n-1,m}} \hat{p}_n$ will produce the picture $\hat{p}_n$ such that $\hat{p}_n[1..n] = \hat{p}[1..n]$, that is $\hat{p}_n$ and $\hat{p}$ have the same n rows. By the property stated above picture $\hat{p}_n$ will have last two rows consisting of $[\#, \pi(q_{n-1,1}), \cdots, \pi(q_{n-1,m}), \#]$, $[\diamond, [q_{n-1,1}\ q_{n,1}], \cdots, [q_{n-1,m}\ q_{nm}], \#]$.

Finally, the last step consists of applying the r-sequence $S_{4,m} = r_1^n \cdot r_2^n \cdots r_j^n \cdots r_{m+1}^n$, where rules $r_1^n \in R_{r,L,4}$, $r_{m+1}^n \in R_{r,R,4}$ and $r_j^n \in R_{r,M,4}$ are based on pairs of tiles $t_{nj} =_r t_{n+1,j}$. Applying $S_{4,m}$ to $\hat{p}_n$ will produce the picture $\hat{p}$ as required.

Let us now show that $L(T) \subseteq L$. We prove the equivalent statement that for any canonical picture $\hat{q} \in L(T)$ there exists a picture $\hat{p}$ such that $B_{2,2}(\hat{p}) \subseteq \Theta \bigwedge \hat{q} = \pi(\hat{p})$ ("full property"). Here, for the sake of simplicity, we consider π extended by the mapping $\# \rightarrow \#$.

The proof will focus on the case when $n, m > 1$. It will be by induction on the number of rows, showing that $\hat{q}$ is built by growing a series of pseudo-canonical pictures, where each pseudo-canonical picture q_i with i rows generates $i-1$ rows of the projection of $\hat{p}$ i.e. there exists p_i such that $B_{2,2}(p_i) \subseteq \Theta \wedge q_i^r[1..i-1] = \pi(p_i)$("weak property"). We finally state that the target full property is obtained when eventually producing $\hat{q}$ by applying the r-sequence adding the bottom border.

The base case of induction, and the only possible starting steps in T, is the building of q_i having $i = 3$ rows, obtained by starting from the empty picture in I and by applying rules in two phases:

– a derivation sequence of c-sequences over the empty picture ; $S_1 \rightarrow S_{2,1} \rightarrow S_{2,2} \cdots \rightarrow S_{2,m} \rightarrow S_3$ where rules of S_1 are in $R_{c,U,2}$, rules of $S_{2,k}$ in $R_{c,U,3}$ and rules of S_3 are in $R_{c,U,4}$, producing the pseudo-canonical picture q_0.
– then a r-sequence over picture q_o where combining rules of the r-sequence are from $R_{r,L,2}, R_{r,M,2}, R_{r,R,2}$, in that order.

The definition of rule sets involved with the second phase, and the requirement of combining of rules in row sequences, imply that $q_i^r[1..2]$ is the projection of a two rows partial picture covered by tiles from Θ, thus satisfying the (weak) property being induced.

Over pictures with such 3 rows, or more rows, only rules from $R_{r,\{L,M,R\},3}$ may be applied (before adding bottom border with rules from $R_{r,\{L,M,R\},4}$). This is our induction step: applying a row sequence of rules from $R_{r,\{L,M,R\},3}$ to a picture q_i with i rows, we obtain a picture q_j with $j = i + 1$ rows, where $q_i^r[1..i-1] = q_j^r[1..i-1]$ satisfies the property, while the bottom $q_i^r[i]$ is replaced in q_j by two new rows. $q_j^r[1..j-1]$ satisfies the property because the added set of tiles $B_{2,2}(q_j^r[i-1..j-1])$ on it are the projection of two rows covered by tiles from Θ, as a consequence of the definition of the applied rules.

A similar reasoning goes when instead we may apply a rule sequence from $R_{r,\{L,M,R\},4}$, obtaining a canonical picture $\hat{q}_j$ with $j = i+1$ rows from a picture q_i with i rows. This time, the definition of the rules being applied shows that the added set of tiles $B_{2,2}(\hat{q}_j^r[i-1..j])$, which includes the bottom border, are projection of a picture covered by tiles from Θ, while by induction the same holds for $\hat{q}_j^r[1..i-1]$, therefore completing our proof. □

Proposition 1. *The picture language L consisting of one-letter alphabet pictures of dimension $(n, n!)$ is generated by TRuS systems.*

Proof. We construct a TRuS system $T = (I, R, \Sigma, \Delta)$ generating language L, where $\Sigma = \{a, b, c\}$ is a finite set of symbols, $\Delta = \{\#, \diamond\}$ is a set of border symbols and I contains the canonical pictures having non bordered versions of size $(n, n!)$ in L with $n \leq 3$. We define R consisting of the set of rules defined below, that is $R = R_r \cup R_{c,INIT1} \cup R_{c,INTER} \cup R_{c,END} \cup R_{c,INIT2}$.

$$R_r = \left\{ r_1 = \tfrac{\#a}{\#\#} \rightarrow \tfrac{\#b}{\#\diamond}, \quad r_2 = \tfrac{aa}{\#\#} \rightarrow \tfrac{bb}{\diamond\diamond}, \quad r_3 = \tfrac{a\#}{\#\#} \rightarrow \tfrac{b\#}{\diamond\#} \right.$$

$$R_{c,INIT1} = \begin{cases} r_4 = \tfrac{\#}{\#}\big|\tfrac{\#}{a} \rightarrow \tfrac{\#\diamond}{ac}, & r_5 = \tfrac{\#}{\#}\big|\tfrac{a}{a} \rightarrow \tfrac{ac}{ac}, \quad r_6 = \tfrac{\#}{\#}\big|\tfrac{a}{a} \rightarrow \tfrac{ac}{ab}, \\ r_7 = \tfrac{\#}{\#}\big|\tfrac{a}{b} \rightarrow \tfrac{ab}{ab}, & r_8 = \tfrac{\#}{\#}\big|\tfrac{b}{\diamond} \rightarrow \tfrac{\#a}{\#b}, \end{cases}$$

$$R_{c,INTER} = \begin{cases} r_9 = \tfrac{\#}{a}\big|\tfrac{\diamond}{c} \rightarrow \tfrac{\#\diamond}{ac}, & r_{10} = \tfrac{a}{a}\big|\tfrac{c}{c} \rightarrow \tfrac{ac}{ac}, \quad r_{11} = \tfrac{a}{a}\big|\tfrac{c}{c} \rightarrow \tfrac{ac}{ab}, \\ r_{12} = \tfrac{a}{a}\big|\tfrac{c}{b} \rightarrow \tfrac{ab}{ab}, & r_{13} = \tfrac{a}{a}\big|\tfrac{b}{b} \rightarrow \tfrac{ab}{ab}, \quad r_{14} = \tfrac{a}{\#}\big|\tfrac{b}{\#} \rightarrow \tfrac{ab}{\#\#}, \end{cases}$$

$$R_{c,END} = \begin{cases} r_{15} = \tfrac{\#}{a}\big|\tfrac{\diamond}{c} \rightarrow \tfrac{\#\#}{aa}, & r_{16} = \tfrac{a}{a}\big|\tfrac{c}{b} \rightarrow \tfrac{aa}{aa}, \quad r_{17} = \tfrac{a}{a}\big|\tfrac{b}{b} \rightarrow \tfrac{aa}{aa}, \\ r_{18} = \tfrac{a}{\#}\big|\tfrac{b}{\#} \rightarrow \tfrac{aa}{\#\#}, \end{cases}$$

$$R_{c,INIT2} = \begin{cases} r_{19} = {}^{\#}_{a}|^{\#}_{a} \rightarrow^{\#\diamond}_{ac}, & r_{20} = {}^{a}_{a}|^{a}_{a} \rightarrow^{ac}_{ac}, \quad r_{21} = {}^{a}_{a}|^{a}_{a} \rightarrow^{ac}_{ab}, \\ r_{22} = {}^{a}_{a}|^{a}_{b} \rightarrow^{ab}_{ab}, & r_{23} = {}^{a}_{\#}|^{b}_{\diamond} \rightarrow^{ab}_{\#\#}. \end{cases}$$

By definitions 2, 3 we can see that only rules in the same set X with $X \in \{R_r, R_{c,INIT1}, R_{c,INTER}, R_{c,END}, R_{c,INIT2}\}$ can be combined to form specific α-sequences, with $\alpha \in \{c, r\}$, the rules in distinct sets either cannot be combined or they cannot join rule sequences that can be applied to the picture.

Now, we denote the r-sequence produced by the rules in R_r as $S_{1,j} = r_1.(r_2)^j.r_3$ where for any integer j, r_2 can be iterated j times. Similarly, we denote the only c-sequences produced by the rules in $R_{c,INIT1}$, $R_{c,INTER}, R_{c,END}$ and $R_{c,INIT2}$ by $S_{2,i} = r_4.(r_5)^i.r_6.r_7.r_8$, $S_{3,h,k} = r_9.(r_{10})^h.r_{11}.r_{12}.(r_{13})^k.r_{14}$, $S_{4,l} = r_{15}.r_{16}.(r_{17})^l.r_{18}$ and $S_{5,m} = r_{19}.(r_{20})^m.r_{21}.r_{22}.r_{23}$ respectively.

Now, let us show by induction on $n \geq 1$ that picture $(n, n!)$ is generated by the system i.e. $L \subseteq L(T)$. Since the pictures in L of size $n \leq 3$ are in I (initial language) it follows that $I \subseteq L(T)$. Given the unary picture p of size $(n, n!)$ in $L(T)$ in the following we show that rules in R can generate the unary picture of size $(n+1, (n+1)!)$.

Given the picture $\hat{p} \in I$ such that p is of size $(n, n!)$, it can be easily verified that the unique α-sequence that can be applied to $\hat{p}$ is the r-sequence $S_{1,n!-1} = r_1.(r_2)^{n!-1}.r_3$ i.e. the rule r_2 in the sequence is iterated $n! - 1$ times generating $\hat{p_1}$. Now, given the picture $\hat{p_1}$ only the unique unambiguous derivation d_1 is applicable to $\hat{p_1}$. More precisely:

$d_1 = \hat{p}_1 \rightarrow_{S_{2,n-2}} \hat{p}_2 \rightarrow^{1}_{S_{3,n-3,1}} \rightarrow^{2}_{S_{3,n-4,2}} \cdots \rightarrow^{n-3}_{S_{3,1,n-3}} \rightarrow^{n-2}_{S_{3,0,n-2}} \hat{p}_3 \rightarrow_{S_{4,n-1}} \hat{p}_4$. Once $\hat{p_4}$ is generated then the only possible derivation that can be applied to $\hat{p_4}$ consists of d_2:

$d_2 = \hat{p}_4 \rightarrow_{S_{5,n-2}} \hat{p}_5 \rightarrow^{1}_{S_{3,n-3,1}} \rightarrow^{2}_{S_{3,n-4,2}} \cdots \rightarrow^{n-3}_{S_{3,1,n-3}} \rightarrow^{n-2}_{S_{3,0,n-2}} \hat{p}_6 \rightarrow_{S_{4,n-1}} \hat{p}_7$. Now, d_2 can be still applied to $\hat{p}_7$ and indeed d_2 can be iterated more times.

Observe that the first application of d_1 and each iteration of d_2 replaces a column $[\#, a, a \cdots, b, \diamond]$ with $(n+1)$ new columns. Since there are $n!$ columns with symbol b and special symbol $\diamond$ in the first pseudo-canonical picture $\hat{p_1}$, totally $(n+1)n!$ columns will replace the $n!$ columns of the non-bordered picture p_1, thus generating the picture $p' \in L$ of size $(n+1, (n+1)!)$. To complete the proof, we have to show that $L(T) \subseteq L$, which easily follows by construction of unambiguous and deterministic derivations. □

5 Conclusions and Open Problems

Tiling rule systems provide a new formalism for defining picture languages that is based on rules to assemble tiles. Pictures of the language are generated by iteratively applying rules: they grow a picture of size (n, m) by locating a $(2, m)$ row (or $(n, 2)$ column) context site picture where the bottom row is replaced and and a new row (or column) is added. Now, tiling rule systems generate a class of languages that properly includes the class of recognizable picture languages. Actually, the proof of the inclusion shows that pictures of a recognizable language are assembled by growing them along a border, that is by adding new rows and

columns. On the contrary, pictures of TRuS languages that are not recognizable languages (see Proposition 1), can only be assembled by adding rows or columns properly inside pictures of smaller size. Moreover, it is proved in [4] that any TRuS system is equivalent to one with the initial language consisting of the empty picture.

Several questions concerning the notion of tiling rule system such as closure properties remain to be investigated, as well as the comparison of this new approach with other language classes like Tiling Rewriting Grammars (TRG) formalism [5].

Acknowledgments. Partially supported by MIUR project "Mathematical aspects and emerging applications of automata and formal languages" (2007).

References

1. Anselmo, M., Giammarresi, D., Madonia, M.: New operations and regular expressions for two-dimensional languages over one-letter alphabet. Theoretical Computer Science 340, 408–431 (2005)
2. Anselmo, M., Madonia, M.: Deterministic two-dimensional languages over one-letter alphabet. In: Bozapalidis, S., Rahonis, G. (eds.) CAI 2007. LNCS, vol. 4728, pp. 147–159. Springer, Heidelberg (2007)
3. Bertoni, A., Goldwurm, M., Lonati, V.: On the complexity of unary tiling-recognizable picture languages. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 381–392. Springer, Heidelberg (2007)
4. Bonizzoni, P., Ferretti, C., Mary, A.R.S., Mauri, G.: Picture languages generated by assembling tiles (extended version, 2008) (2008), http://openit.disco.unimib.it/assemblingTRF.pdf
5. Cherubini, A., Crespi Reghizzi, S., Pradella, M.: Regional languages and tiling: A unifying approach to picture grammars. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 253–264. Springer, Heidelberg (2008)
6. Cherubini, A., Crespi Reghizzi, S., Pradella, M., San Pietro, P.: Picture langugages: Tiling system versus Tile rewriting grammars. Theoretical Computer Science 356, 90–103 (2006)
7. Crespi Reghizzi, S., Pradella, M.: Tile rewriting grammars and picture languages. Theoretical Computer Science 340, 257–272 (2005)
8. De Prophetis, L., Varricchio, S.: Recognizability of Rectangular Pictures by Wang Systems. Journal of Automata, Languages and Combinatorics 2, 4, 269–288 (1997)
9. Giammarresi, D.: Two-dimensional languages and recognizable functions. In: Procs. DLT 1993. World Scientific Publishing Co., Singapore (1994)
10. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 215–267. Springer, Heidelberg (1997)
11. Kari, J., Moore, C.: New results on alternating and non-deterministic two-dimensional finite-state automata. In: Ferreira, A., Reichel, H. (eds.) STACS 2001. LNCS, vol. 2010, pp. 396–406. Springer, Heidelberg (2001)
12. Winfree, E.: Algorithmic self-assembly of DNA: theoretical motivations and 2D assembly experiments. Journal of Biomol. Str. and Dynamics 11, 263–270 (2000)

Undecidability of Operation Problems for T0L Languages and Subclasses

Henning Bordihn[1], Markus Holzer[2,⋆], and Martin Kutrib[2]

[1] Institut für Informatik, Universität Potsdam,
August-Bebel-Straße 89, 14482 Potsdam, Germany
henning@cs.uni-potsdam.de
[2] Institut für Informatik, Universität Giessen,
Arndtstraße 2, 35392 Giessen, Germany
{holzer,kutrib}@informatik.uni-giessen.de

Abstract. We investigate the decidability of the operation problem for T0L languages and subclasses: Fix an operation on formal languages. Given languages from the family considered (0L languages, T0L languages, or their propagating variants), is the application of this operation to the given languages still a language that belongs to the same language family? Observe, that all the Lindenmayer language families in question are anti-AFLs, that is, they are not closed under homomorphisms, inverse homomorphisms, intersection with regular languages, union, concatenation, and Kleene closure. Besides these classical operations we also consider intersection and substitution, since the language families under consideration are not closed under these operations, too. We show that for all of the above mentioned language operations, except for the Kleene closure, the corresponding operation problems of 0L and T0L languages and their propagating variants are not even semidecidable.

1 Introduction

Elementary undecidable questions for formal language families appeared first in [1], where it was shown that the family of languages defined by context-free grammars is too wide to admit a decidable theory for language equivalence. The same holds true for the family of 0L (E0L) languages generated by (extended) context independent Lindenmayer systems (L systems for short). These grammar formalism has been introduced in order to describe the development of lower organism [5,6]. One of the main reasons why L systems are interesting is that they form a parallel counterpart to sequential rewriting mechanisms such as context-free grammars. Moreover, they can be considered as finite substitutions over a free monoid, which are iteratively applied to a designated element of the monoid, the so-called axiom of the system. The basic properties of the derivation in L systems in contrast to sequential rewriting mechanisms can be summarized as follows:

⋆ Most of the work was done while the author was at Institut für Informatik, Technische Universität München, Boltzmannstraße 3, 85748 Garching bei München, Germany.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 236–246, 2009.

- In every derivation step, all symbols in the sentential form have to be rewritten (in parallel).
- There is no distinction between terminal and nonterminal symbols. Therefore, we call those systems *pure* L systems which is in line with the theory of pure grammars [7].
- L systems have a word as axiom instead of a symbol as in the case of Chomsky grammar, or instead of a set of words as in the case of pure grammars.

It is known that for 0L and T0L languages, for example, inclusion, finiteness, and regularity are undecidable [3,9]. The family of T0L languages is generated by context independent tabled L systems. Roughly speaking, these are 0L systems with several production sets, which are also called tables, see, e.g., [9]. On the other hand, some related questions such as membership are decidable.

Another important class of decision problems can be stated as follows. Fix a family of languages and operations thereon. Given languages from this family, is it decidable or semidecidable whether or not the application of the operation to the given languages leads out of this family? In the forthcoming this problem is referred to as the *operation problem*. From an implementation point of view, the operation problem is related to the question whether, for example, a parser or acceptor for a given language can be decomposed into several simpler parsers. Advantages of simpler parsers, whose combination according to the operation is equivalent to the given device, are obvious. For example, the total size of the simpler devices could be smaller than the given parser, the verification is easier, etc. So, there is a natural interest in efficient decomposition algorithms. From this point of view, the complexity of the converse question, whether the composition of languages yields a given language, is interesting. The operation problem can be seen as a weaker class of such problems. Of course, the operation problem makes only sense for language operations under which the family under consideration is *not* closed, since the aforementioned problem becomes trivially decidable otherwise.

The operation problem for families of languages generated by L systems has been investigated only for the *union* of 0L and propagating (that is, non-erasing) 0L languages [4]. Other operations as well as tabled L systems have not been considered yet. The aim of the present paper is to investigate the operation problem for the families of 0L and T0L languages and their propagating variants to a large extent. The notation on 0L and T0L systems and their propagating variants is introduced in Section 2. Besides AFL operations, except the Kleene closure, we also consider intersection and substitution by languages from the families in question. For all of the above mentioned operations we prove non-semidecidability. The results on the Boolean operations can be found in Section 3, while the remaining operations are treated in Section 4. Similarly to the approach in [4], we show how to reduce Post's Correspondence Problem (PCP) to the problem in question [8]. Compared to some other proofs in the literature the constructions presented here and the argumentation are more involved. This is due to the fact that we have to deal with pure language families, which do not allow to hide any sentential form during the derivation process, as in context-free

grammars or extended context-independent L systems. Finally, we summarize our results and state some open problems in the last section.

2 Preliminaries and Definitions

The reader is assumed to be familiar with the basic notions of formal language theory as contained, for example, in [9,10]. In the present paper we will use the following notational conventions. An alphabet is any finite set, its elements are called letters or symbols. For an alphabet V let V^+ and V^* denote the free semigroup and free monoid, respectively, generated by V. The unit element of V^* is the empty word denoted by λ. The reversal of a word $w \in V^*$ is denoted by w^R, and for the length of w we write $|w|$. For the number of occurrences of a symbol a in w we use the notation $|w|_a$. Generally, for a singleton set $\{a\}$ we simply write a. We use $\subseteq$ for *inclusions* and $\subset$ for *strict inclusions*. The powerset of a set S is denoted by 2^S.

Let U and V be two alphabets and σ be a mapping from V into 2^{U^*}, that is, $\sigma(a) \subseteq U^*$, for all $a \in V$. The extension of σ to domain V^* defined by $\sigma(\lambda) = \{\lambda\}$ and $\sigma(w_1 \cdot w_2) = \sigma(w_1) \cdot \sigma(w_2)$, for $w_1, w_2 \in V^*$, is called a *substitution*. If $\sigma(a)$ is a finite set for all $a \in V$, then σ is a finite substitution. If $U = V$, then σ is called substitution over V.

Now we give the formal definition of the L systems which will be considered in this paper.

Definition 1

1. *A* T0L *system is a tuple* $G = (V, P_1, P_2, \ldots, P_r, \omega)$*, where* r *is a positive integer,* V *is an alphabet,* $\omega \in V^+$ *is the axiom, and* P_i*, for* $1 \leq i \leq r$*, is a finite subset of* $V \times V^*$ *such that for every* $a \in V$*, there is a word* $v \in V^*$ *with* $(a, v) \in P_i$*. The sets* P_i *are called the tables of* G*.*
2. *A* T0L *system is* propagating *(a* PT0L *system) if all tables of* G *are finite subsets of* $V \times V^+$*.*
3. *An* 0L *system is a* T0L *system with only one table, that is,* $r = 1$*.*
4. *An* 0L *system is* propagating *(a* P0L *system) if the only table of* G *is a finite subset of* $V \times V^+$*.*

The elements of the tables are called rules and define how a symbol of the current sentential form may be rewritten. Such as in the case of phrase structure grammars we usually write $a \to v$ for (a, v) in P. Since in a single step of a T0L system all symbols are rewritten in parallel according to one of its tables, any table of a T0L system can be viewed as a finite substitution over V. More precisely, with every table $P \subseteq V \times V^*$ we associate the finite substitution σ_P defined by $\sigma_P(a) = \{\, v \mid (a, v) \in P \,\}$. Now, we can define the language generated by a T0L system.

Definition 2. *Let* $r \geq 1$ *and* $G = (V, P_1, P_2, \ldots, P_r, \omega)$ *be a* T0L *system. A word* $x \in V^+$ *directly derives a word* $y \in V^*$ *if there is* i *with* $1 \leq i \leq r$*, such that* $y \in \sigma_{P_i}(x)$*. We write* $x \Rightarrow y$ *in this case. The language* $L(G)$ generated *by* G *is defined to be the set* $L(G) = \{\, w \in V^* \mid \omega \Rightarrow^* w \,\}$*, where* $\Rightarrow^*$ *refers to the reflexive, transitive closure of the derivation relation* $\Rightarrow$*.*

For $X \in \{\text{P0L}, \text{0L}, \text{PT0L}, \text{T0L}\}$, a language is said to be an X language if there is an X system generating it. Let V be an alphabet and σ a substitution over V. If, for all $a \in V$, the set $\sigma(a)$ is an X language, then σ is called an X substitution. By definition, every P0L language is also a 0L, PT0L as well as a T0L language, and both every 0L and every PT0L language is also a T0L language.

In the remainder of this section we introduce the necessary notation from computability theory. A problem is called *decidable*, if there is a Turing machine, that will halt on all inputs and, given an encoding of any instance of the question, will compute the correct answer "yes" or "no" for the instance. Otherwise the problem is *undecidable*. The problem is *semidecidable* or *recursively enumerable*, if the Turing machine halts on all instances for which the answer is "yes." Otherwise the problem is *non-semidecidable*. For example, the well-known halting problem is undecidable. But it is easy to see that it is semidecidable.

We will prove the non-semidecidability of operation problems for T0L systems and subclasses thereof by reduction of Post's Correspondence Problem (PCP) to the problem in question. Formally, the definition of PCP reads as follows, see, e.g., [10]: Let n be a positive integer, V be an alphabet containing at least two letters, and

$$I = \{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\}$$

be a finite set of pairs from $V^* \times V^*$. As n and V are implicitly specified when I is given, the set I determines an instance of the PCP. The PCP I has a solution if and only if there is a sequence of integers $i_1, i_2, \ldots, i_k$ with $k \geq 1$, $1 \leq i_j \leq n$, for $1 \leq j \leq k$, such that

$$u_{i_1} u_{i_2} \ldots u_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}.$$

As an example, assume that $V = \{0, 1\}$ and furthermore, let the PCP instance be $I = \{(1, 111), (10111, 10), (10, 0)\}$. A solution to this instance of the PCP is the sequence $2, 1, 1, 3$ obtaining $10111 \cdot 1 \cdot 1 \cdot 10 = 10 \cdot 111 \cdot 111 \cdot 0$. It is well-known that the PCP is undecidable [8]. Simply by enumerating all possibilities it is easy to see that it is still semidecidable. On the other hand, the problem to determine whether a PCP has *no* solution cannot be semidecidable. Otherwise the PCP would be decidable. So, to be more precise, we will prove our results by reduction of the question whether a PCP has no solution to the operation problem.

Our results (for binary operations) read as follows, where $\circ$ refers to the operation in question:

> Let X and Y be in $\{\text{P0L}, \text{0L}, \text{PT0L}, \text{T0L}\}$. Given two X systems G_1 and G_2, it is non-semidecidable whether $L(G_1) \circ L(G_2)$ is a Y language.

More precisely, we show that the semidecidability of the operation problem under consideration would imply the semidecidability of the question whether a PCP has no solution. Because of the aforementioned inclusion structure of L language families it suffices to prove that, given an instance of the PCP, we can construct two P0L systems G_1 and G_2, such that the language $L(G_1) \circ L(G_2)$ is P0L if the PCP has *no* solution for the given instance, and if the PCP does have a

solution, the resulting language is not even a T0L language. The results and proof structures for unary operations are given analogously.

3 Boolean Operations

In this section we consider the operation problem of the family of T0L languages and subclasses with respect to Boolean operations. Since there are regular languages which are not generated by any T0L system, in addition we consider the intersection with regular languages. First we consider the union operation problem with respect to the four classes of L systems defined in the previous section.

Theorem 1. *Let X and Y be in $\{\mathrm{P0L}, \mathrm{0L}, \mathrm{PT0L}, \mathrm{T0L}\}$. Given two X systems G_1 and G_2, it is non-semidecidable whether the union $L(G_1) \cup L(G_2)$ is a Y language.*

Proof. We prove the non-semidecidability by reducing Post's corresponding problem. Let $I = \{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\}$ be a finite set of pairs from $\{a, b\}^* \times \{a, b\}^*$, an instance of the PCP, and let $\Sigma = \{a, b, c\}$. For technical reasons, we assume (x, x) be not contained in I, that is, we exclude instances for which the PCP is trivially decidable. We consider the P0L system

$$G_1 = (\{S, A, B, C, D, \#\} \cup \Sigma, P_1, S),$$

where P_1 is the union of the following sets of rules:

$$\begin{aligned}
R_1 &= \{S \to \#\#, S \to a^3b^3c^3, S \to A\},\\
R_2 &= \{\, A \to xAx \mid x \in \Sigma \,\},\\
R_3 &= \{\, A \to xBy \mid x, y \in \Sigma, x \neq y \,\} \cup \{\, B \to xBy \mid x, y \in \Sigma \,\},\\
R_4 &= \{\, A \to xC \mid x \in \Sigma \,\} \cup \{\, B \to xC \mid x \in \Sigma \,\} \cup \{\, C \to xC \mid x \in \Sigma \,\},\\
R_5 &= \{\, A \to Dx \mid x \in \Sigma \,\} \cup \{\, B \to Dx \mid x \in \Sigma \,\} \cup \{\, D \to Dx \mid x \in \Sigma \,\},\\
R_6 &= \{B \to \#, C \to \#, D \to \#, \# \to \# \,\},
\end{aligned}$$

and

$$R_7 = \{\, x \to x \mid x \in \Sigma \,\}.$$

The derivation process starts necessarily with an application of a rule from R_1. As only upper case letters can be replaced non-identically, the derivation can be continued only by rewriting A. As long as only rules from R_2 are used, strings in the set

$$K_1 = \{\, wAw^R \mid w \in \Sigma^* \,\}$$

are obtained, and any string in K_1 can be generated. After the symbol A is rewritten by some rule in R_3, R_4, or R_5, the sets

$$\begin{aligned}
K_2 &= \{\, wBw' \mid w, w' \in \Sigma^*, |w| = |w'|, w' \neq w^R \,\},\\
K_3 &= \{\, wCw' \mid w, w' \in \Sigma^*, |w| > |w'| \,\},
\end{aligned}$$

and

$$K_4 = \{\, wDw' \mid w, w' \in \Sigma^*, |w| < |w'| \,\}$$

are generated by further applications of R_3, R_4, or R_5, respectively. Finally, with the help of R_6 we obtain

$$K_5 = \{\, w\#w' \mid w, w' \in \Sigma^*, w' \neq w^R \,\}.$$

Since no further sentential forms can be derived, we have

$$L_1 = L(G_1) = \bigcup_{i=1}^{5} K_i \cup \{\#\#, a^3b^3c^3, S\}.$$

Next, we construct a P0L system G_2 dependent on the given instance of the PCP:

$$G_2 = (\{S, \#, a, b\}, P_2, S),$$

with

$$P_2 = \{\, S \to u_i\#v_i^R \mid 1 \leq i \leq n \,\} \cup \{\# \to u_i\#v_i^R \mid 1 \leq i \leq n \,\} \cup \{a \to a, b \to b\},$$

and set $L_2 = L(G_2)$.

Now we consider the language $L_1 \cup L_2$ in more detail, and distinguish two cases, namely whether the PCP used in the construction of the L systems G_2 has a solution or not.

Case 1. The PCP has no solution for the instance I. Then $L_2 \subset L_1$, hence $L_1 \cup L_2 = L_1$ is a P0L language.

Case 2. If the PCP has a solution for the instance I, then we claim that $L_1 \cup L_2$ cannot be generated by any T0L system. Assume the contrary and let $G = (V, P, \omega)$ be some T0L system with $L(G) = L_1 \cup L_2$.

First, we observe that $\#\#$ is the only word in $L_1 \cup L_2$ which is of the form zz, for some word z. Therefore, in any table $\# \to \#$ is the only rule in P for the symbol $\#$. (The rule $\# \to \lambda$ would imply that λ belongs to $L_1 \cup L_2$, a contradiction.)

As $a^3b^3c^3$ is contained in $L_1 \cup L_2$, in any table for any $x \in \Sigma$, the rules $x \to v$ are so that $|v|_X = 0$, for all $X \in \{S, A, B, C, D, \#\}$. Otherwise a word with at least three occurrences of X could be derived, which contradicts the structure of the words in $L_1 \cup L_2$. Next, $a^3b^3c^3$ is the only word in $L_1 \cup L_2$ over the alphabet Σ. In conclusion, $x \to x$ is the only possible rule, for any $x \in \Sigma$.

Now, let $i_1, i_2, \ldots, i_k$ be a solution of the PCP for the instance I, and let $y = u_{i_1}u_{i_2}\ldots u_{i_k}$. Then $L_0 = \{\, y^m\#(y^R)^m \mid m \geq 1 \,\} \subseteq L_2$. Therefore, for all long enough $v \in L_0$, there are words z, z', and a derivation $\omega \Rightarrow^* zTz' \Rightarrow v$ (recall that $\# \to \#$ is the only rule in P for the symbol $\#$) according to G such that $zTz' \in L_1$ with $T \in \{S, A, B, C, D\}$. Clearly, $T \neq B$ since otherwise $zBz' \in L_1$ implies $|z| = |z'|$ and $z' \neq z^R$. So, $zBz' \Rightarrow v$ is impossible.
Let $T \in \{A, C, D\}$. As $czTz'c$ is an element of L_1 either, also the derivation

$$\omega \Rightarrow^* czTz'c \Rightarrow cy^m\#(y^R)^mc$$

is possible according to G, but $cy^m\#(y^R)^mc$ belongs neither to L_1 nor to L_2, which is a contradiction to our assumption $L(G) = L_1 \cup L_2$. Hence, $L_1 \cup L_2$ is not generated by any T0L system.

The systems G_1 and G_2 are P0L systems and, thus, of any type from $\{\text{P0L}, \text{0L}, \text{PT0L}, \text{T0L}\}$. We have shown that the union $L(G_1) \cup L(G_2)$ is of any type from $\{\text{P0L}, \text{0L}, \text{PT0L}, \text{T0L}\}$, if and only if the PCP has no solution. We conclude that it is non-semidecidable whether the union $L(G_1) \cup L(G_2)$ is a language of any type from $\{\text{P0L}, \text{0L}, \text{PT0L}, \text{T0L}\}$. This proves the theorem. □

We continue our investigation with the intersection operation and base our construction on that of Theorem 1, thus reducing the intersection problem to the undecidability problem of the union problem.

Theorem 2. *Let X and Y be in $\{\text{P0L}, \text{0L}, \text{PT0L}, \text{T0L}\}$. Given two X systems H_1 and H_2, it is non-semidecidable whether the intersection $L(H_1) \cap L(H_2)$ is a Y language.*

Proof. Let $I = \{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\}$ be a finite set of pairs from $\{a, b\}^* \times \{a, b\}^*$, an instance of the PCP, and let G_1 and G_2 be the P0L systems as in the proof of Theorem 1. Let P_1 be the sole table of G_1. We construct the P0L system

$$H_1 = (\{S, A, B, C, D, E, \#, a, b, c\}, P_1', S),$$

where $P_1' = P_1 \cup R_8$ with

$$R_8 = \{\, S \to u_i E v_i^R \mid 1 \le i \le n \,\} \cup \{\, E \to u_i E v_i^R \mid 1 \le i \le n \,\} \cup \{E \to \#\}.$$

Then

$$\begin{aligned} L(H_1) = {} & L(G_1) \\ & \cup \{\, u_{i_1} u_{i_2} \ldots u_{i_j} \# v_{i_j}^R \ldots v_{i_2}^R v_{i_1}^R \mid 1 \le j, 1 \le i_1, i_2, \ldots, i_j \le n \,\} \\ & \cup \{\, u_{i_1} u_{i_2} \ldots u_{i_j} E v_{i_j}^R \ldots v_{i_2}^R v_{i_1}^R \mid 1 \le j, 1 \le i_1, i_2, \ldots, i_j \le n \,\}. \end{aligned}$$

Since

$$L(G_2) = \{S\} \cup \{\, u_{i_1} u_{i_2} \ldots u_{i_j} \# v_{i_j}^R \ldots v_{i_2}^R v_{i_1}^R \mid 1 \le j, 1 \le i_1, i_2, \ldots, i_j \le n \,\},$$

where G_2 is the P0L system from Theorem 1, we can alternatively write language $L(H_1)$ as follows, taking into account that $S \in L(G_1)$:

$$\begin{aligned} L(H_1) = {} & L(G_1) \cup L(G_2) \\ & \cup \{\, u_{i_1} u_{i_2} \ldots u_{i_j} E v_{i_j}^R \ldots v_{i_2}^R v_{i_1}^R \mid 1 \le j, 1 \le i_1, i_2, \ldots, i_j \le n \,\}. \end{aligned}$$

Further, let H_2 be a P0L system generating the set $\{S, A, B, C, D, \#, a, b, c\}^+$. As $L(H_1) \cap L(H_2) = L(G_1) \cup L(G_2)$, and it is non-semidecidable whether $L(G_1) \cup L(G_2)$ is a language of any type from $\{\text{P0L}, \text{0L}, \text{PT0L}, \text{T0L}\}$, the proof is complete. □

It is known that there are regular languages which are not generated by any T0L system [9]. However since semi-groups like $\{S, A, B, C, D, \#, a, b, c\}^+$ belong to all language families in question, we immediately obtain the following corollary.

Corollary 1. *Let X and Y be in $\{\mathrm{P0L}, \mathrm{0L}, \mathrm{PT0L}, \mathrm{T0L}\}$. Given an X system G and a regular language R, it is non-semidecidable whether the intersection $L(G) \cap R$ is a Y language.* □

Whether the only remaining Boolean operation, the complementation, also yields a non-semidecidable operation problem for the four L language families is left open.

4 Non-erasing Homomorphism, Substitution, and Concatenation

Here we consider the operation problem for non-erasing homomorphism, substitution, and finally for concatenation. We start with non-erasing homomorphisms. Observe, that the stated theorem even holds in case of letter-to-letter homomorphisms.

Theorem 3. *Let X and Y be in $\{\mathrm{P0L}, \mathrm{0L}, \mathrm{PT0L}, \mathrm{T0L}\}$. Given an X system G and a (non-erasing) homomorphism h, it is non-semidecidable whether $h(L(G))$ is a Y language.*

Proof. Consider the P0L system H_1 over $V = \{S, A, B, C, D, E, \#, a, b, c\}$ of the proof of Theorem 2, and let $h : V^* \to V^*$ be the homomorphism defined by $h(x) = x$, for $x \in V \setminus \{E\}$ and $h(E) = \#$. Then $h(L(H_1)) = L(G_1) \cup L(G_2)$, where G_1 and G_2 are the P0L systems of the proof of Theorem 1. Since it is non-semidecidable whether $L(G_1) \cup L(G_2)$ is a language of any type from $\{\mathrm{P0L}, \mathrm{0L}, \mathrm{PT0L}, \mathrm{T0L}\}$, the assertion follows. □

For inverse homomorphisms we obtain the following result:

Theorem 4. *Let X and Y be in $\{\mathrm{P0L}, \mathrm{0L}, \mathrm{PT0L}, \mathrm{T0L}\}$. Given an X system G, a (non-erasing) homomorphism h, it is non-semidecidable whether $h^{-1}(L(G))$ is a Y language.*

Proof. We argue as in the previous proof, again by considering the P0L system H_1 over $V = \{S, A, B, C, D, E, \#, a, b, c\}$ of the proof of Theorem 2. Then we define the homomorphism $h : (V \setminus \{E\})^* \to V^*$ by $h(x) = x$, for $x \in V \setminus \{E\}$ and note that $h^{-1}(L(H_1)) = L(G_1) \cup L(G_2)$, where G_1 and G_2 are the P0L systems of the proof of Theorem 1. Thus, the non-semidecidability of the inversion homomorphism operation problem for the languages families in question is immediate. □

For substitutions, whose languages are certain L languages, we find a similar result.

Theorem 5. *Let X, Y and Z be in $\{\mathrm{P0L}, \mathrm{0L}, \mathrm{PT0L}, \mathrm{T0L}\}$. Given an X system G and a Z substitution σ, it is non-semidecidable whether $\sigma(L(G))$ is a Y language.*

Proof. Consider the P0L language $\{a, b\}$ and the P0L substitution σ defined by $\sigma(a) = L(G_1)$ and $\sigma(b) = L(G_2)$, where G_1 and G_2 are the P0L systems of the proof of Theorem 1. Then the undecidability of the union problem reduces to the substitution problem, and the assertion follows. □

In the remainder of this section we consider the concatenation.

Theorem 6. *Let X and Y be in $\{\mathrm{P0L}, \mathrm{0L}, \mathrm{PT0L}, \mathrm{T0L}\}$. Given two X systems H_0 and H_1, it is non-semidecidable whether the concatenation $L(H_0)\cdot L(H_1)$ is a Y language.*

Proof. Given a PCP instance $I = \{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\}$ over the alphabet $\{a, b\}$. Let $H_0 = (\{E, a, b, \#, \$\}, P_0, \#\#E\$\$)$, where

$$P_0 = \{\, E \to u_i E v_i^R \mid 1 \le i \le n \,\} \cup \{a \to a, b \to b, \# \to \#, \$ \to \$\},$$

such that

$$L(H_0) = \{\#\#\}\cdot\{\, u_{i_1}u_{i_2}\ldots u_{i_j}Ev_{i_j}^R\ldots v_{i_2}^R v_{i_1}^R \mid 1 \le j, 1 \le i_1, i_2, \ldots, i_j \le n \,\}\cdot\{\$\$\}.$$

Furthermore, let H_1 be taken from the proof of Theorem 2. This completes the description of the P0L systems.

Next we consider the language $L(H_0) \cdot L(H_1)$ in more detail. We distinguish two cases, namely whether the PCP used in the construction of the L systems has a solution or not:

Case 1. The PCP has no solution for the instance I. Then the rule $E \to \#$ can be omitted from the set of rules of H_1 without changing the generated language $L(H_1)$.

We construct a P0L system H from H_0 and H_1 as follows:

$$H = (\{S, A, B, C, D, E, a, b, c, \#, \$\}, P, \#\#E\$\$S),$$

where $P = P_1' \cup P_0 \cup \{\, X \to X \mid X \in \{S, A, B, C, D, E\} \,\}$. It is easy to see that $L(H) = L(H_0) \cdot L(H_1)$.

Case 2. The PCP has a solution for the given instance, say $j_1 j_2 \ldots j_k$. Let

$$w = (u_{j_1}u_{j_2}\ldots u_{j_k})^m$$

with m sufficiently large. Then any word in $L(H_0)\cdot w\#w$ is in $L(H_0)\cdot L(H_1)$. Assume there is a T0L system H' with $L(H') = L(H_0) \cdot L(H_1)$. As every word in $L(H')$ begins with $\#\#$, in any table the only rule replacing $\#$ is $\# \to \#$.

Next, by arguments similar to the ones given in the proof of Theorem 1 one shows that in any table, $x \to x$ are the only rules for the symbols $x \in \{a, b, c\}$. Since w is long, we conclude that in order to derive the suffix $w\#w$ the symbol $\#$ has to be derived from symbol E. But then the corresponding rule can be applied to the left occurrence of E in any word of the language, yielding a word of the form $\#\#u\#v\$\$y$ which does not belong to $L(H_0) \cdot L(H_1)$. Hence, the words in $L(H_0)\cdot w\#w$ cannot be generated by H', a contradiction. In conclusion, $L(H_0) \cdot L(H_1)$ is not a T0L language.

This proves the assertion. □

5 Conclusions

We have investigated the operation problem for T0L languages and subclasses, that is, we fix an operation on formal languages, and consider the question: Given languages from the language family under consideration, is it decidable or at least semidecidable whether or not the application of the operation to the languages leads out of this family? For the AFL operations, except Kleene closure, we have shown that the considered problem is non-semidecidable, even already for P0L systems. Since we were dealing with pure language families some of our constructions and arguments were slightly more involved than usual.

Nevertheless, some interesting problems remain open:

1. Where are the borderlines between decidability, undecidability, and non-semidecidability of the operation problems for T0L languages and subclasses exactly? Are there some nontrivial operations for which the problem is decidable? Can we characterize these cases?
2. What about the operation problems for deterministic T0L languages and subclasses? A T0L system $G = (V, P_1, P_2, \ldots, P_r, \omega)$ is *deterministic* if all tables $P_i \subseteq V \times V^*$ with $1 \leq i \leq r$ are such that $\sigma_{P_i}(a) = \{ v \mid (a, v) \in P_i \}$ is a singleton set for every letter $a \in V$. In particular, for the most simple L systems, namely propagating D0L systems, which are nothing other than iterated non-erasing homomorphisms, decidability of the union problem has been shown [4]. But the decidability status of the operation problem with respect to other operations is unknown.
3. Determine the decidability status of the operation problem with respect to complementation and Kleene closure for the L systems in question.

We have seen that all studied question are non-semidecidable, but what is their exact status of unsolvability? To this end, one has to consider the *arithmetic hierarchy*, which is defined as follows:

$$\Sigma_1 = \{ L \mid L \text{ is recursively enumerable} \},$$
$$\Sigma_{n+1} = \{ L \mid L \text{ is recursive enumerable in some } A \in \Sigma_n \},$$

and Π_n is the class of all complements of languages in Σ_n, that is, define $\Pi_n = \{ L \mid \overline{L} \in \Sigma_n \}$, for $n \geq 1$. Here, a language L is said to be recursively enumerable in some B if there is a Turing machine with oracle B that semi-decides L. Alternatively, a more revealing characterization of the arithmetic hierarchy can be given in terms of alternation of quantifiers. More precisely, a language L is in Σ_n, for $n \geq 1$, if and only if there exists a *decidable* $(n+1)$-ary predicate R such that

$$L = \{ w \mid \exists y_1 \forall y_2 \exists y_3 \ldots Q y_n \, R(w, y_1, y_2, \ldots, y_n) \},$$

where $Q = \exists$ if n is odd and $Q = \forall$ if n is even. Thus, the non-semidecidability implies that the problems in question are at least on level Π_1 of the arithmetic hierarchy, but what is their precise level? As the reader may easily verify, the upper bound on the operation problems under consideration is Σ_2, although

we have to leave open whether the problems are complete for this class, but we conjecture it to be so. This conjecture is based on the fact that recently it was shown that for certain operations, under which the families of linear and deterministic context-free languages are not closed, the corresponding operation problems are Σ_2-complete [2].

References

1. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung 14, 143–172 (1961)
2. Bordihn, H., Holzer, M., Kutrib, M.: Unsolvability levels of operation problems for subclasses of context-free languages. Int. J. Found. Comp. Sci. 16, 423–440 (2005)
3. Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. Springer, Heidelberg (1989)
4. Dassow, J., Păun, G., Salomaa, A.: On the union of 0L languages. Inform. Process. Letters 47, 59–63 (1993)
5. Lindenmayer, A.: Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. J. Theor. Biol. 18, 280–299 (1968)
6. Lindenmayer, A.: Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. J. Theor. Biol. 18, 300–315 (1968)
7. Maurer, H.A., Salomaa, A., Wood, D.: Pure grammars. Inform. Control 44, 47–72 (1980)
8. Post, E.L.: A variant of a recursively unsolvable problem. Bull. AMS 52, 264–268 (1946)
9. Rozenberg, G., Salomaa, A.: The Mathematical Theory of L Systems. Academic Press, London (1980)
10. Salomaa, A.: Formal Languages. Academic Press, London (1973)

Decision Problems for Convex Languages

Janusz Brzozowski, Jeffrey Shallit, and Zhi Xu

David R. Cheriton School of Computer Science,
University of Waterloo, Waterloo, ON, Canada N2L 3G1
{brzozo,shallit,z5xu}@uwaterloo.ca

Abstract. We examine decision problems for various classes of convex languages, previously studied by Ang and Brzozowski under the name "continuous languages". We can decide whether a language L is prefix-, suffix-, factor-, or subword-convex in polynomial time if L is represented by a DFA, but the problem is PSPACE-hard if L is represented by an NFA. If a regular language is not convex, we prove tight upper bounds on the length of the shortest words demonstrating this fact, in terms of the number of states of an accepting DFA. Similar results are proved for some subclasses of convex languages: the prefix-, suffix-, factor-, and subword-closed languages, and the prefix-, suffix-, factor-, and subword-free languages.

1 Introduction

A word x is a *factor* of a word w if $w = uxv$ for some words u and v. A word x is a *subword* of w if x is a subsequence of w. Thierrin [1] introduced convex languages with respect to the subword relation, and Ang and Brzozowski [2] generalized this concept to arbitrary relations.

A language L is *prefix-convex* if $u, w \in L$ with u a prefix of w implies that any word v must also be in L if u is a prefix of v and v is a prefix of w. L is *prefix-free* if $w \in L$ implies that no proper prefix of w is in L. L *prefix-closed* if $w \in L$ implies that every prefix of w is also in L.

Similar definitions hold for suffix-, factor-, and subword-convex languages, and suffix-, factor-, and subword-free and closed languages. Prefix-free languages (prefix codes) were studied by Berstel and Perrin [3]. Han has recently considered X-free languages for various values of X, such as prefix, suffix, factor and subword [4]. A factor-closed language is often called *factorial*.

We consider the computational complexity of testing whether a given language is prefix-convex, suffix-convex, etc., prefix-closed, suffix-closed, etc., for a total of 12 different problems. The computational complexity of these decision problems depends on how the language is represented. If it is specified by a DFA, the decision problem is solvable in polynomial time. If it is represented as a regular expression or an NFA, the decision problem is PSPACE-complete. We also consider the following question: given that a language is *not* prefix-convex, suffix-convex, etc., what is a good upper bound on the length of the shortest words (*witnesses*) demonstrating this fact?

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 247–258, 2009.

In Section 2 we study the complexity of testing for convexity for languages represented by DFA's, and include testing for closure and freeness as special cases. In Section 3 we exhibit shortest witnesses to the lack of convexity. Convex languages specified by NFA's and context-free grammars are briefly studied in Section 4. Section 5 concludes the paper. Owing to the space constraints, we have had to omit many results and proofs; they can be found in the full version of our paper [5].

2 Decision Problems for Languages Specified by DFA's

We will show that, if a regular language L is represented by a DFA M with n states, it is possible to test the property of prefix-, suffix-, factor-, and subword-convexity efficiently, in fact, in $O(n^3)$ time.

Let $\trianglelefteq$ be one of the four relations *prefix, suffix, factor,* or *subword*. The basic idea is as follows: L is *not* $\trianglelefteq$-convex if and only if there exist words $u, w \in L$, $v \notin L$, such that $u \trianglelefteq v \trianglelefteq w$. Given M, we create an NFA-ϵ M' with $O(n^3)$ states and transitions that accepts the language $\{w \in L(M) \ : \ \text{there exist } u \in L(M), v \notin L(M) \text{ such that } u \trianglelefteq v \trianglelefteq w\}$. Then $L(M') = \emptyset$ if and only if $L(M)$ is $\trianglelefteq$-convex. We can test the emptiness of $L(M')$ using depth-first search in time linear in the size of M'. This gives an $O(n^3)$ algorithm for testing the $\trianglelefteq$-convexity.

Since the constructions for all four properties are similar, we handle the hardest case (factor-convexity) in detail, and refer the reader to [5] for the rest.

Factor-convex languages. Suppose $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA accepting the language $L = L(M)$, and suppose M has n states. We construct an NFA-ϵ M' such that $L(M')$ is the set of words $w \in \Sigma^*$ such that there exist $u, v \in \Sigma^*$ such that u is a factor of v, v is a factor of w, and $u, w \in L, v \notin L$. Clearly $L(M') = \emptyset$ if and only if $L(M)$ is factor-convex.

States of M' are quadruples, where components 1, 2, and 3 keep track of where M is upon processing w, v, and u (respectively). The last component is a flag indicating the present *mode* of the simulation process. Formally, $M' = (Q', \Sigma, \delta', q_0', F')$, where $Q' = Q \times Q \times Q \times \{1, 2, 3, 4, 5\}$, $q_0' = [q_0, q_0, q_0, 1]$, $F' = F \times (Q - F) \times F \times \{5\}$, and

1. $\delta'([p, q_0, q_0, 1], a) = \{[\delta(p, a), q_0, q_0, 1]\}$, for all $p \in Q, a \in \Sigma$;
2. $\delta'([p, q_0, q_0, 1], \epsilon) = \{[p, q_0, q_0, 2]\}$, for all $p \in Q$;
3. $\delta'([p, q, q_0, 2], a) = \{[\delta(p, a), \delta(q, a), q_0, 2]\}$, for all $p, q \in Q, a \in \Sigma$;
4. $\delta'([p, q, q_0, 2], \epsilon) = \{[p, q, q_0, 3]\}$, for all $p, q \in Q$;
5. $\delta'([p, q, r, 3], a) = \{[\delta(p, a), \delta(q, a), \delta(r, a), 3]\}$, for all $p, q, r \in Q, a \in \Sigma$;
6. $\delta'([p, q, r, 3], \epsilon) = \{[p, q, r, 4]\}$, for all $p, q, r \in Q$;
7. $\delta'([p, q, r, 4], a) = \{[\delta(p, a), \delta(q, a), r, 4]\}$, for all $p, q, r \in Q, a \in \Sigma$;
8. $\delta'([p, q, r, 4], \epsilon) = \{[p, q, r, 5]\}$, for all $p, q, r \in Q$;
9. $\delta'([p, q, r, 5], a) = \{[\delta(p, a), q, r, 5]\}$, for all $p, q, r \in Q, a \in \Sigma$.

One can verify that the contruction is correct, and that the NFA-ϵ M' has $3n^3 + n^2 + n$ states and $(3|\Sigma| + 2)n^3 + (|\Sigma| + 1)(n^2 + n)$ transitions, where $|\Sigma|$ is the cardinality of Σ [5]. In other words, the following theorem holds:

Theorem 1. *If M is a DFA with n states, there exists an NFA-ϵ M' with $O(n^3)$ states and transitions such that M' accepts the language $L(M') = \{w \in \Sigma^* : \text{there exist } u, v \in \Sigma^* \text{ such that } u \text{ is a factor of } v, v \text{ is a factor of } w, \text{and } u, w \in L, v \notin L\}$.*

Corollary 1. *We can decide if a given regular language L accepted by a DFA with n states is factor-convex in $O(n^3)$ time.*

Factor-closed languages. The language L is *not* factor-closed if and only if there exist words v, w such that v is a factor of w, and $w \in L$, while $v \notin L$. Given a DFA M accepting L, we construct an NFA-ϵ M' such that $L(M') = \{w \in \Sigma^* : \text{there exists } v \in \Sigma^* \text{ such that } v \text{ is a factor of } w, \text{and } w \in L, v \notin L\}$. Then $L(M') = \emptyset$ if and only if $L(M)$ is factor-closed. The size of M' is $O(n^2)$.

States of M' are triples, where components 1 and 2 keep track of where M is upon processing w and v (respectively). The last component is a flag as before. Formally, $M' = (Q', \Sigma, \delta', q_0', F')$, where $Q' = Q \times Q \times \{1, 2, 3\}$; $q_0' = [q_0, q_0, 1]$; $F' = F \times (Q - F) \times \{3\}$; and

1. $\delta'([p, q_0, 1], a) = \{[\delta(p, a), q_0, 1]\}$ for $p \in Q$, $a \in \Sigma$.
2. $\delta'([p, q_0, 1], \epsilon) = \{[p, q_0, 2]\}$, for all $p \in Q$;
3. $\delta'([p, q, 2], a) = \{[\delta(p, a), \delta(q, a), 2]\}$, for all $p, q \in Q$;
4. $\delta'([p, q, 2], \epsilon) = \{[p, q, 3]\}$, for all $p, q \in Q$;
5. $\delta'([p, q, 3], a) = \{[\delta(p, a), q, 3]\}$, for $p, q \in Q$, $a \in \Sigma$.

M' has $2n^2 + n$ states and $(2|\Sigma| + 1)n^2 + (|\Sigma| + 1)$ transitions. Thus we have Theorem 2. (This result was previously obtained by Béal et al. [6, Prop. 5.1, p. 13] through a slightly different approach.)

Theorem 2. *We can decide if a given regular language L accepted by a DFA with n states is factor-closed in $O(n^2)$ time.*

The converse of the relation "u is a factor of v" is "v contains u as a factor". This relation and similar converse relations derived from the prefix, suffix, and subword relations, lead to "converse-closed languages" [2]. Subword-closed and converse-subword-closed languages were characterized by Thierrin [1]. It has been shown by de Luca and Varricchio [7] that a language L is factor-closed (factorial, in their terminology) if and only if it is a complement of an ideal, that is, if and only if $L = \overline{\Sigma^* K \Sigma^*}$ for some $K \subseteq \Sigma^*$. Ang and Brzozowski [2] noted that a language is an ideal if and only if it is converse-factor-closed, that is, if, for every $u \in L$, each word of the form $v = xuy$ is also in L. Thus, to test whether L is converse-factor-closed, we must check that there is no pair (u, v) such that $u \in L$, $v \notin L$, and u is a factor of v. This is equivalent to testing whether $\overline{L}$ is factor-closed. Then the following is an immediate consequence of Theorem 1:

Corollary 2. *We can decide if a given regular language L accepted by a DFA with n states is an ideal in $O(n^2)$ time.*

Factor-free languages. Factor-free (also known as infix-free) languages have been studied recently by Han et al. [8], who gave efficient algorithms for determining if the language accepted by an NFA is prefix-, suffix-, or factor-free. We can decide whether a DFA language is factor-free in $O(n^2)$ time with the automaton we used for testing factor-closure, except that the set of accepting states is now $F' = F \times F \times \{3\}$.

3 Minimal Witnesses

Let $\trianglelefteq$ represent one of the four relations: factor, prefix, suffix, or subword. A necessary and sufficient condition that a language L be *not* $\trianglelefteq$-convex is the existence of a triple (u, v, w) of words, where $u, w \in L$, $v \notin L$, $u \trianglelefteq v$, and $v \trianglelefteq w$. We call such a triple a *witness* to the lack of $\trianglelefteq$-convexity. A witness (u, v, w) is *minimal* if every other witness (u', v', w') satisfies $|w| < |w'|$, or $|w| = |w'|$ and $|v| < |v'|$, or $|w| = |w'|$, $|v| = |v'|$, and $|u| < |u'|$. The *size* of a witness is $|w|$.

Similarly, if L is not $\trianglelefteq$-closed, then (v, w) is a *witness* if $w \in L$, $v \notin L$, and $v \trianglelefteq w$. A witness (v, w) is *minimal* if there exists no witness (v', w') such that $|w'| < |w|$, or $|w'| = |w|$ and $|v'| < |v|$. The *size* is again $|w|$. For $\trianglelefteq$-freeness, *witness, minimal witness,* and *size* are defined as for $\trianglelefteq$-closure, except that both words are in L.

Suppose we are given a regular language L specified by an n-state DFA M, and we know that L is not $\trianglelefteq$-convex (respectively, $\trianglelefteq$-closed or $\trianglelefteq$-free). A natural question then is, what is a good upper bound on the size of the shortest witness that demonstrates the lack of this property?

3.1 Factor-Convexity

From Theorem 1, we deduce Corollary 3, which gives an $O(n^3)$ upper bound for the length of a witness to the lack of factor-convexity. This bound is best possible, as is shown in Theorem 3, whose proof appears in Section 3.3.

Corollary 3. *Suppose L is accepted by a DFA with n states and L is not factor-convex. Then there exists a witness (u, v, w) such that $|w| \leq 3n^3 + n^2 + n - 1$.*

Theorem 3. *There is a class of non-factor-convex regular languages L_n, accepted by DFA's with $O(n)$ states, such the size of the minimal witness is $\Omega(n^3)$.*

Factor-closure. Theorem 2 gives us a $O(n^2)$ upper bound on the length of a witness to the failure of the factor-closed property:

Corollary 4. *If L is accepted by a DFA with n states and L is not factor-closed, then there exists a witness (v, w) such that $|w| \leq 2n^2 + n - 1$.*

This $O(n^2)$ upper bound is best possible. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, where $Q = \{q_0, q_1, \cdots, q_n, q_{n+1}, p_0, p_1, \cdots, p_n, p_{n+1}\}$, $\Sigma = \{\mathtt{0}, \mathtt{1}\}$, and $F = Q \setminus \{q_{n+1}\}$. The transition function is $\delta(q_0, \mathtt{0}) = q_0$, $\delta(q_0, \mathtt{1}) = q_1$, $\delta(q_{n+1}, \mathtt{0}) = q_{n+1}$, $\delta(q_{n+1}, \mathtt{1}) = q_{n+1}$,

$$\delta(q_i, \mathtt{0}) = \begin{cases} q_{i+1}, & \text{if } 0 < i < n; \\ q_1, & \text{if } 0 < i = n, \end{cases}$$

$$\delta(q_i, \mathtt{1}) = \begin{cases} q_1, & \text{if } 0 < i < n-1; \\ p_0, & \text{if } 0 < i = n-1; \\ q_{n+1}, & \text{if } 0 < i = n; \end{cases}$$

$$\delta(p_j, \mathtt{0}) = \begin{cases} p_{j+1}, & \text{if } 0 \leq j < n; \\ q_0, & \text{if } 0 \leq j = n; \end{cases}$$

$$\delta(p_j, \mathtt{1}) = \begin{cases} q_{n+1}, & \text{if } 0 \leq j < n; \\ p_{n+1}, & \text{if } 0 \leq j = n; \end{cases}$$

and $\delta(p_{n+1}, \mathtt{0}) = q_{n+1}$, $\delta(p_{n+1}, \mathtt{1}) = q_{n+1}$. The DFA M has $2n+4$ states. The following theorem holds [5]:

Theorem 4. *For the DFA M above, let $L = L(M)$. For any witness (u, v) to the lack of factor-closure we have $|v| \geq (n+1)^2 - 1$, and this bound is achievable.*

Factor-freeness. From the remarks at the end of Section 2, we get

Corollary 5. *If L is accepted by a DFA with n states and L is not factor-free, then there exists a witness (v, w) such that $|w| \leq 2n^2 + n - 1$.*

Up to a constant, Corollary 5 is best possible, as the following theorem shows.

Theorem 5. *There is a class of languages accepted by DFA's with $O(n)$ states, such that the smallest witness to the lack of factor-freeness is of size $\Omega(n^2)$.*

Proof. Let $L = \mathtt{bb}(\mathtt{a}^n)^+\mathtt{b} \cup \mathtt{b}(\mathtt{a}^{n+1})^+\mathtt{b}$. This language can be accepted by a DFA with $2n+6$ states. However, the shortest witness to lack of factor-freeness is $(\mathtt{ba}^{n(n+1)}\mathtt{b}, \mathtt{bba}^{n(n+1)}\mathtt{b})$, which has size $n^2 + n + 3$. □

3.2 Prefix-Convexity

For prefix-convexity, we have the following theorem.

Theorem 6. *Let M be a DFA with n states. If $L(M)$ is not prefix-convex, there is a witness (u, v, w) with $|w| \leq 2n - 1$. Furthermore, this bound is best possible, as for all $n \geq 2$, there exists a unary DFA with n states that achieves this bound.*

Proof. If $L(M)$ is not prefix-convex, then such a witness (u, v, w) exists. Without loss of generality, assume that (u, v, w) is minimal. Now write $w = uyz$, where $v = uy$ and $w = vz$.

Let $\delta(q_0, u) = p$, $\delta(p, y) = q$, and $\delta(q, z) = r$. Let P be the path from q_0 to r traversed by uvw, and let P_1 be the states from q_0 to p (not including p), P_2 be the states from p to q (not including q), and P_3 be the states from q to r (not including r); see Figure 1. Since (u, v, w) is minimal, we know that every state of

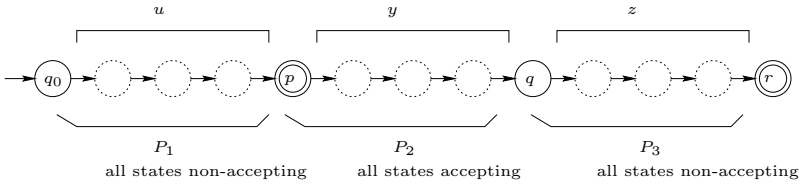


Fig. 1. The acceptance path for w

P_3 is rejecting, since we could have found a shorter w if there were an accepting state among them. Similarly, every state of P_2 must be accepting, for, if there were a rejecting state among them, we could have found a shorter y and hence a shorter v. Finally, every state of P_1 must be rejecting, since, if there were an accepting state, we could have found a shorter u.

Let $r_i = |P_i|$ for $i = 1, 2, 3$. There are no repeated states in P_3, for if there were, we could cut out the loop to get a shorter w; the same holds for P_2 and P_1. Thus $r_i \leq n - 1$ for $i = 1, 2, 3$. Now P_1 and P_2 are disjoint, since all the states of P_1 are rejecting, while all the states of P_2 are accepting. Similarly, the states of P_3 are disjoint from P_2. So $r_1 + r_2 \leq n$ and $r_2 + r_3 \leq n$. It follows that $r_1 + r_2 + r_3 \leq 2n - r_3$. Since $r_3 \geq 1$, it follows that $|w| \leq 2n - 1$.

To see that $2n - 1$ is optimal, consider the DFA of n states accepting the unary language $L = \mathtt{a}^{n-1}(\mathtt{a}^n)^*$. Then L is not prefix-convex, and the shortest witness is $(\mathtt{a}^{n-1}, \mathtt{a}^n, \mathtt{a}^{2n-1})$. □

Prefix-closure. For prefix-closed languages we can get an even better bound.

Theorem 7. *Let M be an n-state DFA, and suppose $L = L(M)$ is not prefix-closed. Then the minimal witness (v, w) showing L is not prefix-closed has $|w| \leq n$, and this is best possible.*

Proof. Assume that (v, w) is a minimal witness. Consider the path P from q_0 to $q = \delta(q_0, w)$, passing through $p = \delta(q_0, v)$. Let P_1 denote the part of the path P from q_0 to p (not including p) and P_2, the part of the path from p to q (not including q). Then all the states traversed in P_2 must be rejecting; otherwise, we would get a shorter w. Similarly, all the states traversed in P_1 must be accepting, because otherwise we could get a shorter v. Neither P_1 nor P_2 contains a repeated state, because if they did, we could "cut out the loop" to get a shorter v or w. Furthermore, the states in P_1 are disjoint from P_2. So the total number of states in the path to w (not counting q) is at most n. Thus $|w| \leq n$.

The result is best possible, as the example of the unary language $L = (\mathtt{a}^n)^*$ shows. This language is not prefix-closed, can be accepted by a DFA with n states, and the smallest witness is $(\mathtt{a}, \mathtt{a}^n)$. □

Prefix-freeness. For the prefix-free property we have:

Theorem 8. *If L is accepted by a DFA with n states and is not prefix-free, then there exists a witness (v, w) with $|w| \leq 2n - 1$. The bound is best possible.*

Proof. The proof is similar to that of Theorem 6. The bound is achieved by a unary DFA accepting $\mathtt{a}^{n-1}(\mathtt{a}^n)^*$. □

3.3 Suffix-Convexity

For the suffix-convex property, the cubic upper bound implied by Corollary 3 is best possible, up to a constant factor.

Theorem 9. *There is a class of non-suffix-convex regular languages L_n, accepted by DFA's with $O(n)$ states, such the size of the minimal witness is $\Omega(n^3)$.*

Proof. Let $L = \mathtt{bbb}(\mathtt{a}^{n-1})^+ \cup \mathtt{bb}(\mathtt{a} + \mathtt{aa} + \cdots + \mathtt{a}^{n-1})(\mathtt{a}^n)^* \cup \mathtt{b}(\mathtt{a}^{n+1})^+$. Then L can be accepted by a DFA with $3n + 5$ states, as illustrated in Figure 2.

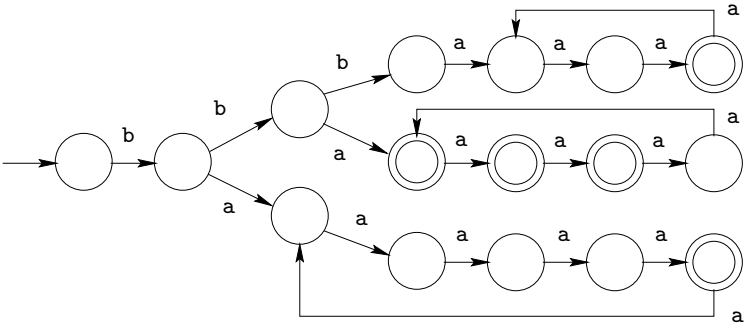


Fig. 2. Example of the construction in Theorem 9 for $n = 4$. All unspecified transitions go to a rejecting "dead state" d that cycles on all inputs.

It can be verified that L is not suffix-convex and the shortest witness is $(\mathtt{ba}^i, \mathtt{bba}^i, \mathtt{bbba}^i)$, where $i = \mathrm{lcm}(n-1, n, n+1) \geq (n-1)n(n+1)/2$. □

A similar technique can be used for non-factor-convex languages. This allows us to prove Theorem 3 in the same way Theorem 9, except we use the language $L\mathtt{b}$ instead.

Suffix-closure. Obviously, a witness to the failure of suffix-closure is also a witness to the failure of factor-closure. So the proof of Theorem 4 shows that the bound $(n+1)^2 - 1$ also holds for suffix-closed languages.

Ang and Brzozowski pointed out [2] that a language L is factor-closed if and only if L is both prefix-closed and suffix-closed. The next result [5] shows that a long minimal witness for factor-closure must also be a witness for suffix-closure.

Proposition 1. *Let M be a DFA of n states, and $L = L(M)$. Let v be the shortest word such that there is $u \notin L, v \in L, |v| > n$ and u is a factor of v. Then u is a suffix of v.*

Suffix-freeness

Theorem 10. *There exists a class of languages accepted by DFA's with $O(n)$ states, such that the smallest witness to the lack of suffix-freeness is of size $\Omega(n^2)$.*

Proof. Let $L = \mathtt{bb}(\mathtt{a}^n)^+ \cup \mathtt{b}(\mathtt{a}^{n+1})^+$. This language is accepted by a DFA with $2n+5$ states. However, the shortest witness to the lack of suffix-freeness $(\mathtt{ba}^{n(n+1)}, \mathtt{bba}^{n(n+1)})$ has size n^2+n+2. □

3.4 Subword-Convexity

We now turn to subword properties. First, we recall some facts about the pumping lemma. If $w = a_1 \cdots a_m$ with $a_i \in \Sigma$ for $1 \le i \le m$, we write $w[i,j]$ for the factor $a_i \cdots a_j$. Assume that $M = (Q, \Sigma, \delta, q_0, F)$ is an n-state DFA, $m \ge n$, let $q \in Q$, and consider the state sequence $S(q,w) = (\delta(q, w[1,0]), \ldots, \delta(q, w[1,m]))$. We know that some state in $S(q,w)$ must appear more than once, because there are only n distinct states in M. Let $\delta(q, w[1,i])$ be the first state that appears more than once in S, and let $x = w[1,i]$. Moreover, let $\delta(q, w[1,j])$ be the first state in $S(q,w)$ equal to $\delta(q, w[1,i])$, and let $y = w[i+1,j]$. Finally, let $z = w[j+1,m]$. Then $w = xyz$, where $|xy| \le n$, $|y| > 0$, and $|z| \ge m-n$, and $\delta(q,x) = \delta(q,xy)$. By the pumping lemma, $xy^*z \subseteq L$. By the definition of x and y, all the states in the sequence $S(q, w[1,j-1])$ are distinct. For a word w with $|w| = m \ge n$, we refer to the factorization $w = xyz$ as the *canonical factorization of w with respect to q*.

Subword-closure. Here $v \trianglelefteq w$ means v is a subword of w. If $L = L(M)$ is not subword-closed, then (v,w) is a *witness* if $w \in L$, $v \notin L$, and $v \trianglelefteq w$.

Lemma 1. *Let M be a DFA with $n \ge 2$ states such that $L(M)$ is not subword-closed. For any witness (v,w), there exists a witness (v',w') with $|w'| \le n$ and $w' \trianglelefteq w$.*

Proof. We will show that, for any witness (v,w) with $|w| \ge n+1$, we can find a witness (v',w') with $|w'| < |w|$ and $w' \trianglelefteq w$. The lemma then follows.

Suppose that (v,w) is a minimal witness, and $|w| = m \ge n+1$. Then the canonical factorization of w is $w = xyz$, where $|xy| \le n$, $|y| > 0$, and $|z| \ge m-n > 0$.

If there is a z' such that $z' \trianglelefteq z$ and $xyz' \notin L$, then $xz' \notin L$, since xyz' and xz' lead to the same state in M. Then (xz', xz) is a witness with $|xz| < |w|$ and $xz \trianglelefteq w$. Thus we can assume that

$$z' \trianglelefteq z \text{ implies } xyz' \in L. \tag{1}$$

Since $v \trianglelefteq w = xyz$, we can write $v = v_x v_y v_z$, where $v_x \trianglelefteq x$, $v_y \trianglelefteq y$, and $v_z \trianglelefteq z$. Clearly, $v \trianglelefteq xyv_z$. If $v_z \ne z$, then by (1), we have $xyv_z \in L$, and (v, xyv_z) is a witness with $|xyv_z| < |w|$ and $xyv_z \trianglelefteq w$. Thus we may assume that our witness has the form $(v_x v_y z, xyz)$.

In the particular case that $z' = \epsilon$, (1) implies that $xy \in L$. If $y' \trianglelefteq y$ and $xy' \notin L$, then (xy', xy) is a witness with $|xy| < |w|$ and $xy \trianglelefteq w$. Thus $y' \trianglelefteq y$ implies $xy' \in L$.

Finally, if $x' \trianglelefteq x$ and $x' \notin L$, then (x', x) is a witness with $|x| < |w|$ and $x \trianglelefteq w$. Thus $x' \trianglelefteq x$ implies $x' \in L$. Altogether, we may assume that all the states along the path spelling w in M are accepting. We know that the states in $S = (\delta(q_0, w[1,0]), \ldots, \delta(q_0, w[1, |xy| - 1]))$ are all distinct. Also, the states in $S' = (\delta(q_0, v_x v_y z[1,1]), \ldots, \delta(q_0, v_x v_y z[1, |z| - 1]))$ are all accepting and distinct; otherwise, v would not be shortest.

We now claim that no state can be in both S and S'. For suppose that $\delta(q_0, w[1,i]) = \delta(q_0, v_x v_y z[1,k])$, for some $0 \leq i \leq |x|$, $0 < k < |z|$. Then $(w[1,i]z[k+1,|z|], xz)$ is a witness with $|xz| < |w|$ and $xz \trianglelefteq w$, since $w[1,i] = x[1,i]$, and $x[1,i]z[k+1,|z|] \trianglelefteq xz$. Next, if $\delta(q_0, xy[1,j]) = \delta(q_0, v_x v_y z[1,k])$, for some $0 < j < |y|$, $0 < k < |z|$, then $(xy[1,j]z[k+1,|z|], xyz[k+1,|z|])$ is a witness with $|xyz[k+1,|z|]| < |w|$ and $xyz[k+1,|z|] \trianglelefteq w$, since $xy[1,j]z[k+1,|z|] \trianglelefteq xyz[k+1,|z|]$, and $xyz[k+1,|z|] \in L$ by (1).

Under these conditions M must have $|xy| + (|z| - 1) = |xyz| - 1$ distinct accepting states, and at least one rejecting state. Hence $|xyz| = |w| \leq n$ and we have found a witness with the required properties. □

Corollary 6. *Let M be a DFA with $n \geq 2$ states. If $L(M)$ is not subword-closed, there exists a witness (v, w) with $|w| \leq n$. Furthermore, this is the best possible bound, as there exists a unary DFA with n states that achieves this bound.*

For $n = 1$, L is either $\emptyset$ or Σ^*, and these languages are subword-closed.

Subword-freeness

Lemma 2. *Let M be a DFA with $n \geq 2$ states such that $L(M)$ is not subword-free. For any witness (u, w), there exists a witness (u', w') with $|w'| \leq 2n - 1$, and $w' \trianglelefteq w$.*

Corollary 7. *Let M be a DFA with $n \geq 2$ states. If $L(M)$ is not subword-free, there exists a witness (u, w) with $|w| \leq 2n - 1$. This is the best possible bound, as there exists a unary DFA with $2n - 1$ states that achieves this bound.*

Subword-Convexity

Lemma 3. *Let M be a DFA with $n \geq 2$ states such that $L(M)$ is not subword-convex. For any witness (u, v, w), there exists a witness (u', v', w') with $w' \trianglelefteq w$, and $|w'| \leq 3n - 2$.*

Proof. We will show that, for any witness (u, v, w) with $|w| \geq 3n - 1$, we can find a witness (u', v', w') with $|w'| < |w|$ and $w' \trianglelefteq w$. The lemma then follows.

We assume without loss of generality that v is a shortest possible word corresponding to the given w, and u is a shortest word corresponding to v and w.

First, consider the witness (u, v) to the lack of subword-closure of the language $\overline{L}$. By Lemma 1, there exists a witness (u', v') to the failure of subword-closure of $\overline{L}$ such that $v' \trianglelefteq v$ and $|v'| \leq n$. Therefore we can assume that we have a witness (u, v, w) to the failure of subword-convexity such that $|v| \leq n$.

Suppose that (u, v, w) is a minimal witness, and $|w| \geq 3n-1$. Then the canonical factorization of w is $w = x_1y_1z_1$, where $|x_1y_1| \leq n$, $|y_1| > 0$, and $|z_1| \geq 2n-1 \geq n > 0$. Consider the states

$$p_0 = \delta(q_0, x_1y_1),\ p_1 = \delta(q_0, x_1y_1z_1[1,1]),\ \cdots,\ p_{|z_1|} = \delta(q_0, x_1y_1z_1).$$

Since $|z_1| \geq n$, there must be at least one pair (p_i, p_j) of states such that $p_i = p_j$. If p_0 is the state that is repeated, let i be the greatest index such that $p_0 = p_i$, and let $x_2 = \epsilon$, $y_2 = z_1[1,i]$, and $z_2 = z_1[i+1, |z_1|]$. If p_i is the first state that is repeated, let j be the greatest index such that $p_i = p_j$, and let $x_2 = z_1[1,i]$, $y_2 = z_1[i+1, j]$, and $z_2 = z_1[j+1, |z_1|]$. If $\delta(q_0, x_1y_1x_2y_2), \delta(q_0, x_1y_1x_2y_2z_2[1,1]), \ldots, \delta(q_0, x_1y_1x_2y_2z_2)$ has no repeated states, we stop. Otherwise, we apply the same procedure to z_2, and so on. In any case, eventually we reach a z_k for which no repeated states exist. Then we have the factorization $w = x_1y_1x_2y_2\cdots x_ky_kz_k$, where $x_1y_1^*x_2y_2^*\cdots x_ky_k^*z_k \subseteq L$, $|x_2\cdots x_kz_k| < n$ (otherwise, there would be repeated states), $|y_i| > 0$, for $i = 1, \ldots, k$, and $k \geq 2$.

For any $y_2' \trianglelefteq y_2, \cdots, y_k' \trianglelefteq y_k$, we have $x_1y_1x_2y_2'\cdots x_ky_k'z_k \in L$. Otherwise, the triple $(x_1x_2\cdots x_kz_k,\ x_1x_2y_2'\cdots x_ky_k'z_k,\ x_1x_2y_2\cdots x_ky_kz_k)$ is a witness with $|x_1x_2y_2\cdots x_ky_kz_k| < |w|$, and $x_1x_2y_2\cdots x_ky_kz_k \trianglelefteq w$.

Since $v \trianglelefteq w$, we can now write $v = v_{x_1}v_{y_1}v_{x_2}v_{y_2}\cdots v_{x_k}v_{y_k}v_{z_k}$, where $v_{x_1} \trianglelefteq x_1$, etc. If there is a y_i with $i \geq 2$, such that $v_{y_i} = \epsilon$, then we can replace that y_i by ϵ in w and obtain a smaller witness. Hence each v_{y_i} must be nonempty. By the same argument, if there is a letter in y_i, for $i \geq 2$, that is not used in v_{y_i}, then that letter can be removed, yielding a smaller witness. Therefore $y_i = v_{y_i}$ for $i = 2, \ldots, k$. We claim that $|y_2\cdots y_k| < |v|$; otherwise $v = v_{y_2}\cdots v_{y_k} = y_2\cdots y_k$ and $(u, v, x_1x_2y_2\cdots x_ky_kz_k)$ is a witness with $|x_1x_2y_2\cdots x_ky_kz_k| < |w|$. Thus $|y_2\cdots y_k| < |v| \leq n$, and $|w| = |x_1y_1| + |x_2\cdots x_kz_k| + |y_2\cdots y_k| \leq n + (n-1) + (n-1) = 3n-2$. □

Corollary 8. *Let M be a DFA with $n \geq 2$ states. If $L(M)$ is not subword-convex, there exists a witness (u, v, w) with $|w| \leq 3n-2$.*

We do not know whether $3n-2$ is the best bound. The unary language $\mathtt{a}^{n-1}(\mathtt{a}^n)^*$ is accepted by a DFA with n states and has a minimal witness $(\mathtt{a}^{n-1}, \mathtt{a}^n, \mathtt{a}^{2n-1})$, showing that $2n-1$ is achievable.

4 Languages Specified by Other Means

4.1 Languages Specified by NFA's

Some of our decision problems become PSPACE-complete if M is represented by an NFA. Our fundamental tool is the following classical lemma [9]:

Lemma 4. *Let T be a one-tape deterministic Turing machine and $p(n)$ a polynomial such that T never uses more than $p(|x|)$ space on input x. Then there is a finite alphabet Δ and a polynomial $q(n)$ such that we can construct a regular expression r_x in $q(|x|)$ steps, such that $L(r_x) = \Delta^*$ if T doesn't accept x, and*

$L(r_x) = \Delta^* - \{w\}$ for some nonempty w (depending on x) otherwise. Similarly, we can construct an NFA M_x in $q(|x|)$ steps, such that $L(M_x) = \Delta^*$ if T doesn't accept x, and $L(M_x) = \Delta^* - \{w\}$ for some nonempty w (depending on x) otherwise.

Theorem 11. *The problem of deciding whether a given regular language L, represented by an NFA or regular expression, is prefix-convex (resp., suffix-, factor-, subword-convex), or prefix-closed (resp., suffix-, factor-, subword-closed) is PSPACE-complete.*

For the properties of prefix-, suffix, and factor-closed properties, this result was essentially already proved by Hunt and Rosenkrantz [10, Thm. 3.4].

The situation is different for deciding the property of prefix-freeness, suffix-freeness, etc., for languages represented by NFA's, as the following theorem shows. This was proved by Han et al. [8] through a different approach.

Theorem 12. *Let M be an NFA with n states and t transitions. Then we can decide in $O(n^2 + t^2)$ time whether $L(M)$ is prefix-free (resp., suffix-free, factor-free, subword-free).*

Minimal Witnesses for NFA's. We have already seen that the length of the minimal witness for the lack of convexity or closure is polynomial in the size of the DFA. For the case of NFA's, however, this bound no longer holds.

Theorem 13. *There is a class of NFA's with $O(n)$ states such that the shortest witness to the lack prefix-convexity (resp., suffix-, factor-, subword-convexity) or prefix-closure (resp., suffix-, factor-, subword-closure) is of length $2^{\Omega(n)}$.*

Theorem 14. *There exists a class of languages, accepted by NFA's with $O(n)$ states and $O(n)$ transitions, such that the minimal witness for the lack of prefix-freeness is of length $\Omega(n^2)$.*

For the lack of subword-freeness, we cannot improve the bound we obtained for DFA's in Corollary 7, as the proof we presented there also works for NFA's.

4.2 Languages Specified by Context-Free Grammars

If L is represented by a context-free grammar, then the decision problems corresponding to convex and closed languages become undecidable. This follows easily from a well-known result that the set of invalid computations of a Turing machine is a CFL [11, Lemma 8.7, p. 203]. Similarly, the decision problems corresponding to the properties of prefix-free, suffix-free, and factor-free become undecidable for CFL's, as shown by Jürgensen and Konstantinidis [12, Thm. 9.5, p. 581]. However, testing subword-freeness is still decidable for CFL's:

Theorem 15. *There is an algorithm that, given a context-free grammar G, will decide if $L(G)$ is subword-free.*

5 Conclusions

We have shown that we can decide in $O(n^3)$ time whether a language specified by a DFA is prefix-, suffix-, factor-, or subword-convex, and that the corresponding closure and freeness properties can be tested in $O(n^2)$ time. If L is specified by an NFA or a regular expression, these problems are PSPACE-complete.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada.

References

1. Thierrin, G.: Convex languages. In: Nivat, M. (ed.) Automata, Languages, and Programming, pp. 481–492. North-Holland, Amsterdam (1973)
2. Ang, T., Brzozowski, J.: Continuous languages. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) Proc. 12th International Conference on Automata and Formal Languages, pp. 74–85. Computer and Automation Research Institute, Hungarian Academy of Sciences (2008)
3. Berstel, J., Perrin, D.: Theory of Codes. Academic Press, New York (1985)
4. Han, Y.S.: Decision algorithms for subfamilies of regular languages using state-pair graphs. Bull. European Assoc. Theor. Comput. Sci. (93), 118–133 (2007)
5. Brzozowski, J.A., Shallit, J., Xu, Z.: Decision problems for convex languages (preprint, 2008), http://arxiv.org/abs/0808.1928
6. Béal, M.P., Crochemore, M., Mignosi, F., Restivo, A., Sciortino, M.: Computing forbidden words of regular languages. Fund. Inform. 56, 121–135 (2003)
7. de Luca, A., Varricchio, S.: Some combinatorial properties of factorial languages. In: Capocelli, R. (ed.) Sequences, pp. 258–266. Springer, Heidelberg (1990)
8. Han, Y.S., Wang, Y., Wood, D.: Infix-free regular expressions and languages. Internat. J. Found. Comp. Sci. 17, 379–393 (2006)
9. Aho, A., Hopcroft, J., Ullman, J.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
10. Hunt III, H.B., Rosenkrantz, D.J.: Computational parallels between the regular and context-free languages. SIAM J. Comput. 7, 99–114 (1978)
11. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
12. Jürgensen, H., Konstantinidis, S.: Codes. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 511–607. Springer, Heidelberg (1997)

On a Family of Morphic Images of Arnoux-Rauzy Words

Michelangelo Bucci and Alessandro De Luca

Dipartimento di Matematica e Applicazioni "R. Caccioppoli"
Università degli Studi di Napoli Federico II, via Cintia, Monte S. Angelo
80126 Napoli, Italy
{micbucci,alessandro.deluca}@unina.it

Abstract. In this paper we prove the following result. Let s be an infinite word on a finite alphabet, and $N \geq 0$ be an integer. Suppose that all left special factors of s longer than N are prefixes of s, and that s has at most one right special factor of each length greater than N. Then s is a morphic image, under an injective morphism, of a suitable standard Arnoux-Rauzy word.

1 Introduction

Factor complexity is a common theme in the combinatorial analysis of finite and infinite words. Being the function counting distinct blocks (factors) of each length, it is one of the most natural measures of complexity of a word. A famous theorem by Morse and Hedlund [1] characterizes ultimately periodic sequences as the ones having bounded complexity.

Sturmian words have the lowest possible unbounded complexity ($n+1$ factors of each length n). They make up one of the most studied family of infinite words, not just because of their theoretical interest (see [2] for a general introduction, or [3] for a recent survey). From the definition, it follows that Sturmian words are on a binary alphabet, and have exactly one *left special* factor of each length n (a factor is left special if it is a suffix of at least two distinct factors of length $n+1$).

As is well known, a first natural generalization of Sturmian words for alphabets with an arbitrary number of letters was introduced by Arnoux and Rauzy [4]. An infinite word s is *Arnoux-Rauzy* (or *strict episturmian*, see below) if it is recurrent (i.e., all factors of s occur infinitely often) and it has exactly one left special factor and one right special factor per length, that appear in s immediately preceded (resp. followed) by all letters occurring in s. More detailed definitions will be given in Sect. 2.

A remarkable property of Sturmian words, shared by Arnoux-Rauzy words, is their *closure under reversal*: if $w = a_1 a_2 \cdots a_n$ is a factor of an Arnoux-Rauzy word s with $a_i \in A$ for $i = 1, \ldots, n$, then $\tilde{w} = a_n a_{n-1} \cdots a_1$ is a factor of s too. This led Droubay, Justin, and Pirillo [5] to a generalization: an infinite word is *episturmian* if it has at most one left special factor per length, and is closed

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 259–266, 2009.

under reversal. Episturmian words are recurrent, but have no restriction on the number of letters immediately preceding left special factors. Thus the family of episturmian words strictly contains the one of Arnoux-Rauzy words.

The class of *ϑ-episturmian* words is a further generalization, recently introduced in [6] by substituting the reversal operator with any *involutory antimorphism* ϑ of A^*. Generalizing even more, by requiring the condition on special factors only for sufficient lengths, *ϑ-words with seed* are obtained (see [6]).

All such words have a *standard* counterpart, where the unique left special factors correspond to prefixes of the infinite word. For instance, a *ϑ-standard word with seed* is any infinite word s which is closed under ϑ and such that any sufficiently long left special factor of s is a prefix of it. For all the above classes, standard words are good representatives, in the sense that an infinite word s belongs to one of such classes if and only if s has the same set of factors as some standard word of that class (see [5,6]).

Our main result shows that, in the standard case, even when the further step of dropping the "closure under some ϑ" requirement is made, the large class of words thus obtained retains a strong link with Arnoux-Rauzy words. More precisely, we will prove the following.

Theorem 1. *Let $s \in A^\omega$ satisfy the following two conditions for all $n \geq N$, where $N \geq 0$:*

1. *any left special factor of s having length n is a prefix of s,*
2. *s has at most one right special factor of length n.*

Then there exists $B \subseteq \mathrm{alph}(s)$ and a standard Arnoux-Rauzy word $t \in B^\omega$ such that s is a morphic image (under an injective morphism) of t.

In the next section we shall give all the formal definitions and preliminary results needed for our proof, which will be given in Sect. 3. For more basics about combinatorics on words, we refer to [7]. For more details on episturmian words and their generalizations, see [3,5,8,9,6,10,11].

2 Basic Definitions and Results

In the following, A will denote a finite alphabet, A^* the free monoid of words over A, and A^ω the set of infinite words over A. The identity element of A^* is the *empty word* ε.

Let s be a finite or infinite word. The set of letters occurring in s is denoted by $\mathrm{alph}(s)$. A *factor* of s is any finite word w such that $s = uwv$ for suitable words u, v; if u (resp. v) is the empty word we call w a *prefix* (resp. *suffix*) of s. A *border* of $s \in A^*$ is a word which is both a prefix and a suffix of s. If s is nonempty, we denote by s^f its first letter, and if s is also finite we denote by s^ℓ its last letter[1]. With $\mathrm{Fact}(s)$, $\mathrm{Pref}(s)$, and $\mathrm{Suff}(s)$ we denote respectively the set

[1] This notation should not be confused with powers of a word; no such power occurs in this paper.

of factors, prefixes, and suffixes of s. The *factor complexity* of s is the function $c_s : \mathbb{N} \to \mathbb{N}$ defined by $c_s(n) = \#(A^n \cap \mathrm{Fact}(s))$ for all $n \geq 0$. We remark that the complexity of an infinite word s is a nondecreasing function.

Let $w \in \mathrm{Fact}(s)$. A factor v of s is called a *right* (resp. *left*) *extension* of w in s if w is a proper prefix (resp. suffix) of v. If $|w| = n$, the *right* (resp. *left*) *degree* of w in s is the number of its distinct right (resp. left) extensions of length $n+1$. For all $n \geq 0$, any factor of s of length $n+1$ is uniquely determined by its first letter and by its suffix of length n, or by its last letter and by its prefix of length n. An immediate consequence is the following well-known identity:

$$\sum_{w \in \mathrm{F}_s(n)} \deg^-(w) = c_s(n+1) = \sum_{w \in \mathrm{F}_s(n)} \deg^+(w) \ , \tag{1}$$

in which the operators $\deg^-$ and $\deg^+$ denote the left and right degree respectively, and $\mathrm{F}_s(n) = A^n \cap \mathrm{Fact}(s)$.

We recall that w is called a *right* (resp. *left*) *special* factor of s if its right (resp. left) degree is at least 2, i.e., if there exist two distinct letters a and b such that wa and wb (resp. aw and bw) are factors of s. If a factor of s is both left and right special, then it is called *bispecial*.

A *complete return* to w in s is any factor of s containing exactly two occurrences of w, one as a prefix and the other as a suffix. If $z = vw$ is a complete return to w, then v is called a *return word* to w (cf. [12]).

An infinite word s is *recurrent* if each of its factors has infinitely many occurrences in s; it is *uniformly recurrent* if the gaps between consecutive occurrences of any factor are bounded. Equivalently, s is uniformly recurrent if for all factors w there are finitely many distinct return words to w in s.

Given any prefix p of an infinite word s, there exists a unique factorization of s by means of the return words to p in s. By mapping each return word to a different letter of a suitable alphabet, and then applying such a map to s thus factorized, we obtain a *derivated word* of s with respect to p (cf. [12]). Clearly, s is a morphic image of its derivated words.

The following simple lemma is the first basic ingredient needed for our main result.

Lemma 1. *Let s be an infinite word such that any sufficiently long left special factor of s is a prefix of it. Then s is recurrent.*

Proof. By contradiction, suppose there exists a factor w of s having only finitely many occurrences in s, and let λw be the prefix of s ending with the rightmost occurrence of w in s. Then all prefixes of s from length $|\lambda w|$ onward do not reoccur in s, and so have left degree 0.

We claim that this implies that s has also at least one left special factor for each length $n \geq |\lambda w|$. Indeed, for all such n the left sum in (1) has $c_s(n) = \#\,\mathrm{F}_s(n)$ terms. Since the prefix has left degree 0, there must be a term greater than 1 in order to have $c_s(n+1) \geq c_s(n)$ (which is true as s is infinite). By definition, a factor with left degree greater than 1 is a left special factor.

For sufficiently large n, such a factor should be a prefix of s by hypothesis. We have reached a contradiction. □

An infinite word s is *periodic* if it can be written as $s = vvv\cdots = v^\omega$ for some finite word v. An *ultimately periodic* word is an infinite word of the form uv^ω for some $u, v \in A^*$. As is well known (see for instance [13, Lemma 1.4.4]), a recurrent ultimately periodic word is necessarily periodic.

We need one of the most well-known and useful restatements of the theorem of Morse and Hedlund (cf. [1, Theorem 7.3]):

Theorem 2. *An infinite word s is ultimately periodic if and only if $c_s(n) = c_s(n+1)$ for some $n \geq 0$.*

As a consequence of Lemma 1, we obtain the following specialization.

Proposition 1. *An infinite word s is (purely) periodic if and only if it has no left special factor of some length n.*

Proof. If $s = p^\omega$ with $p \in A^*$, then s has no left special factors of length $|p|$. Conversely, assume that s has no left special factor of length n. This implies

$$\#(A^n \cap \mathrm{Fact}(s)) = \#(A^{n+1} \cap \mathrm{Fact}(s)) \ ,$$

so that by Theorem 2, s is ultimately periodic. Clearly s has no left special factor of any length $k \geq n$, thus it trivially satisfies the hypothesis of Lemma 1. Therefore s is recurrent, and hence periodic. □

The following proposition was proved in [10, Lemma 7] under different hypotheses. We report an adapted proof for the sake of completeness.

Proposition 2. *Let s be a recurrent aperiodic infinite word. Then every factor w of s is contained in some bispecial factor of s.*

Proof. Since s is recurrent, we can consider a complete return z to w in s. Writing $z = vw$, it cannot happen that the factor w is always preceded by v in s, otherwise s would be periodic. Thus some suffix of z of length at least $|w|$ must be a left special factor of s. Let $x \in A^*$ be of minimal length such that xw is a left special factor of s. Such a word is trivially unique, and w is always preceded in s by x. In a similar way, there exists a unique $y \in A^*$ of minimal length such that wy is right special in s, and w is always followed by y.

Since xw is left special in s and xw is always followed by y one has that xwy is also left special. Similarly, since wy is right special and always preceded by x, xwy is right special. Hence every factor w of s is contained in some bispecial factor $W = xwy$ of s. □

A recurrent word s is an *Arnoux-Rauzy* word if it has exactly one left special factor and one right special factor of each length, all of degree $\#\,\mathrm{alph}(s)$. It is natural to extend this definition to the case of a unary alphabet $A = \{a\}$; the word a^ω is considered an Arnoux-Rauzy word, since it has a unique factor of each length, clearly not special but of degree $1 = \#A$.

Thus we can reformulate the definition as follows: a recurrent word s is Arnoux-Rauzy if for all $n \geq 0$, all factors of length n have minimum left degree (i.e. 1) except one whose left degree is maximum($\#\,\mathrm{alph}(s)$), and the same

occurs for right degrees (but the two special factors may be different). By (1), this implies $c_s(n+1) = c_s(n) + \#\,\mathrm{alph}(s) - 1$ and then, as $c_s(0) = 1$,

$$c_s(n) = 1 + (\#\,\mathrm{alph}(s) - 1)n \quad \text{for all } n \in \mathbb{N} \ .$$

Arnoux-Rauzy words are uniformly recurrent (cf. [5]); this was part of the definition in [4]. An Arnoux-Rauzy word s is *standard* if its left special factors are prefixes of s.

Example 1. A well-known standard Arnoux-Rauzy word is the so-called *Tribonacci* (or Rauzy) word

$$\tau = abacabaabacababacabaabacabacabaabacabab\cdots$$

which can be obtained as a fixed point of the morphism $a \to ab$, $b \to ac$, $c \to a$ (see [4,5]).

Remark 1. In order to show that a given infinite word s is a standard Arnoux-Rauzy word, it is sufficient to prove the following two conditions:

1. s has exactly one factor of right degree $\#\,\mathrm{alph}(s)$ for each length,
2. every left special factor of s is a prefix of it.

Indeed, under such hypotheses s is recurrent by Lemma 1. Moreover, in view of (1), by the first condition we derive

$$c_s(n+1) \geq \#\,\mathrm{alph}(s) + c_s(n) - 1 \tag{2}$$

for all $n \geq 0$; by condition 2, all factors which are not prefixes have left degree 1, so that equality holds in (2) and there is one factor of left degree $\#\,\mathrm{alph}(s)$. In conclusion, all factors of length n have left degree 1, except one which has left degree $\#\,\mathrm{alph}(s)$, and the same occurs for right degrees, for all n; hence s is an Arnoux-Rauzy word (standard by condition 2).

3 Proof of Theorem 1

Suppose first that s has no left special factor of some length n. Then s is periodic by Proposition 1, so that it is trivially a morphic image of x^ω for any $x \in \mathrm{alph}(s)$.

Now let us assume that s has at least one left special factor of each length – exactly one, from length N on. By Lemma 1, s is recurrent, so that by Proposition 2 it has infinitely many bispecial factors, which we denote by $W_0 = \varepsilon, W_1, \ldots, W_n, \ldots$, where $|W_i| \leq |W_{i+1}|$ for all $i \geq 0$. Let j be the least index such that $|W_j| \geq N$. Since prefixes (resp. suffixes) of left (resp. right) special factors are left (resp. right) special themselves, by conditions 1 and 2 it follows that W_i is a border of W_{i+1} for all $i \geq j$, and the sequence whose n-th term is the (right) degree of W_n for all $n \geq j$ is then non-increasing. Hence there exists $k \geq j$ such that W_n has the same degree of W_k for all $n \geq k$, that is, the above considered sequence is constant from its k-th term on. We set

$$B = \{x \in A \mid W_k x \in \mathrm{Fact}(s)\} \subseteq \mathrm{alph}(s) \ ,$$

so that $\#B$ is, by definition, the degree of W_k.

We now consider the return words to $w = W_k$ in s. Let $u_1 w = w v_1$ and $u_2 w = w v_2$ be any two distinct complete returns to w in s, and let us show that $v_1^f \neq v_2^f$. Indeed, let p be the longest common prefix of v_1 and v_2. If $p = v_1$, then $|v_2| > |v_1|$ as $v_1 \neq v_2$; since $w v_1 = u_1 w$, there is an internal occurrence of w in $w v_2$, contradicting the definition of complete return. The same argument applies if $p = v_2$. Thus p is a proper prefix of both v_1 and v_2, so that wp is a right special factor of s. Since $|w| \geq N$, and w is a right special factor of s, by condition 2 it follows that w is a suffix of wp. This implies $p = \varepsilon$, since otherwise there would be an internal occurrence of w in $w v_1$ and $w v_2$. Hence $v_1^f \neq v_2^f$ as desired. Since w is also left special in s, using a symmetric argument one can prove that $u_1^\ell \neq u_2^\ell$.

From this it follows that for each $x \in B$, there exists a unique complete return $u_x w = w v_x$ to w in s, such that $v_x^f = x$. We define a morphism $\varphi : B^* \to A^*$ by $\varphi(x) = u_x$. Note that φ is injective, as $\varphi(B)$ is a suffix code having the same cardinality as B.

By definition, we have $s = \varphi(t)$, where $t \in B^\omega$ is a derivated word of s with respect to its prefix w. We note that, as a consequence of the definition of return words, one has

$$z \in \mathrm{Fact}(t) \Leftrightarrow \varphi(z) w \in \mathrm{Fact}(s)\,, \quad z \in \mathrm{Pref}(t) \Leftrightarrow \varphi(z) w \in \mathrm{Pref}(s)\ . \tag{3}$$

We will prove that t is a standard Arnoux-Rauzy word; it suffices (see Remark 1) to show that t has exactly one right special factor of each length, that each right special factor has degree $\#B$, and finally that all left special factors of t are prefixes of it.

Clearly t is not periodic, as $s = \varphi(t)$ and s is not periodic. Hence t has right special factors of any length. Let z_1 and z_2 be any two right special factors of t having the same length. Thus there exist distinct letters $x_1, y_1, x_2, y_2 \in B$ such that $x_i \neq y_i$ and $z_i x_i, z_i y_i \in \mathrm{Fact}(t)$ for $i = 1, 2$. By (3), this implies $\varphi(z_i x_i) w, \varphi(z_i y_i) w \in \mathrm{Fact}(s)$. For $\alpha \in \{x_i, y_i\}$ and $i = 1, 2$ we have

$$\varphi(z_i \alpha) w = \varphi(z_i) u_\alpha w = \varphi(z_i) w v_\alpha \in \mathrm{Fact}(s)$$

with $v_{x_i}^f \neq v_{y_i}^f$, so that $\varphi(z_1) w$ and $\varphi(z_2) w$ are right special factors of s. By condition 2, either $\varphi(z_1) w \in \mathrm{Suff}(\varphi(z_2) w)$, or vice versa. The word w has $|z_1| + 1 = |z_2| + 1$ occurrences in both $\varphi(z_1) w$ and $\varphi(z_2) w$, and it is a prefix of both, by the definition of return word. Hence we derive $\varphi(z_1) w = \varphi(z_2) w$, so that $z_1 = z_2$ by the injectivity of φ.

If z is a right special factor of t, by the above argument $\zeta := \varphi(z) w$ is right special in s. As $|\zeta| \geq |w|$, the word ζ is a suffix of some W_n with $n \geq k$, so that $\zeta x = \varphi(z) w x \in \mathrm{Fact}(s)$ for all $x \in B$. Since the only complete return to w in s starting with wx is $w v_x$, it follows that

$$\varphi(z) w v_x = \varphi(z) u_x w = \varphi(zx) w \in \mathrm{Fact}(s)\ ,$$

so that $zx \in \mathrm{Fact}(t)$ for all $x \in B$, proving that z has right degree $\#B$.

Let now z' be a left special factor of t, and let $xz', yz' \in \mathrm{Fact}(t)$ for some distinct letters $x, y \in B$. Then $\varphi(xz') w, \varphi(yz') w \in \mathrm{Fact}(s)$. As $\varphi(x)^\ell = u_x^\ell \neq$

$u_y^\ell = \varphi(y)^\ell$, $\varphi(z')w$ is a left special factor of s. By condition 1, it follows $\varphi(z')w \in \mathrm{Pref}(s)$ and then $z' \in \mathrm{Pref}(t)$ by (3). □

4 Concluding Remarks

Theorem 1 shows that what seems to be a natural (though very wide) generalization of the standard episturmian words, retains a strong connection with Arnoux-Rauzy words.

One could ask whether such result could be improved to obtain a full characterization of morphic images of standard Arnoux-Rauzy words. However, the converse of Theorem 1 is false; a simple counterexample is given by the image s of the Tribonacci word under the episturmian morphism (cf. [5,8]) $f : a \to a$, $b \to ba$, $c \to ca$. Indeed,

$$s = f(\tau) = abaacaabaaabaacaabaabaacaabaaabaacaabaa\cdots$$

does not satisfy condition 1 of Theorem 1 for any N, as aa is a left special factor of s which is not a prefix of it. Nevertheless, this counterexample suggests that the situation could be better in the general (non-standard) case, since the word s, being episturmian, *does* have only one left special factor and one right special factor of each length.

By modifying the proof of Theorem 1 suitably, it is not difficult to show the following:

Theorem 3. *If $s \in A^\omega$ is recurrent and has at most one left special factor and one right special factor for all lengths $k \geq N$, then there exist $B \subseteq A$, an injective morphism $\varphi : B^* \to A^*$, and an Arnoux-Rauzy word $t \in B^\omega$ such that $s \in \mathrm{Suff}(\varphi(t))$.*

This is somehow weaker than the original Theorem 1, as we only get that s is *a suffix of* a morphic image of an Arnoux-Rauzy word. Therefore, any improvement of Theorem 3 would be welcome, as well as any step towards the converse (what can be said about special factors of morphic images of Arnoux-Rauzy words?). Having a simple characterization could help in a more general classification of infinite words with low factor complexity.

References

1. Morse, M., Hedlund, G.A.: Symbolic dynamics. American Journal of Mathematics 60, 815–866 (1938)
2. Berstel, J., Séébold, P.: Sturmian words. In: Lothaire, M. (ed.) Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2002)
3. Berstel, J.: Sturmian and episturmian words. In: Bozapalidis, S., Rahonis, G. (eds.) CAI 2007. LNCS, vol. 4728, pp. 23–47. Springer, Heidelberg (2007)
4. Arnoux, P., Rauzy, G.: Représentation géométrique de suites de complexité $2n+1$. Bulletin de la Société Mathématique de France 119, 199–215 (1991)

5. Droubay, X., Justin, J., Pirillo, G.: Episturmian words and some constructions of de Luca and Rauzy. Theoretical Computer Science 255, 539–553 (2001)
6. Bucci, M., de Luca, A., De Luca, A., Zamboni, L.Q.: On different generalizations of episturmian words. Theoretical Computer Science 393, 23–36 (2008)
7. Lothaire, M.: Combinatorics on Words. Addison-Wesley, Reading (1983)
8. Justin, J., Pirillo, G.: Episturmian words and episturmian morphisms. Theoretical Computer Science 276, 281–313 (2002)
9. de Luca, A., De Luca, A.: Pseudopalindrome closure operators in free monoids. Theoretical Computer Science 362, 282–300 (2006)
10. Bucci, M., de Luca, A., De Luca, A., Zamboni, L.Q.: On θ-episturmian words. European Journal of Combinatorics 30, 473–479 (2009)
11. Fischler, S.: Palindromic prefixes and episturmian words. Journal of Combinatorial Theory, Series A 113, 1281–1304 (2006)
12. Durand, F.: A characterization of substitutive sequences using return words. Discrete Mathematics 179, 89–101 (1998)
13. de Luca, A., Varricchio, S.: Finiteness and regularity in semigroups and formal languages. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Berlin (1999)

Monadic Datalog Tree Transducers

Matthias Büchse and Torsten Stüber*,**

Faculty of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
{buechse,stueber}@tcs.inf.tu-dresden.de

Abstract. We introduce a tree transducer model combining aspects of both attributed tree transducers and monadic datalog, thereby allowing to specify in one rule information transport for non-adjacent nodes. We show that our model is strictly more powerful than attributed tree transducers, and we identify a large syntactic subclass which is as powerful as attributed tree transducers. This is shown by an effective construction.

1 Introduction

In [1], (nondeterministic) attributed tree transducers, an abstract form of attribute grammars [2,3], are introduced as a formal model of syntax-directed semantics [4], that is, a model for specifying tree transformations. Monadic datalog [5], a syntactically restricted fragment of standard datalog [6], is a means of formally specifying node selection queries on trees.

We roughly sketch the underlying ideas of both formalisms: First, a set of (property) names is defined, called *attributes* and *predicates*, respectively. For a given tree t, these names are instantiated for every node of t, yielding *attribute instances* and *atom instances*. A finite set of rules is then used to restrict the set of *interpretations*, that is, mappings from these instances to some semantic domain. In the case of attributed tree transducers, the semantic domain is the set of trees languages over an output alphabet, for monadic datalog, this is the set of Boolean values. If necessary, one interpretation is chosen according to criteria specific to the respective model, and the semantics associated with t is the value of some designated name instance.

Rules of attributed tree transducers allow to specify the relationship of the values of attributes at nodes which are within the same *fork*, that is, some node together with its direct descendants. An example of a rule is

$$R_\sigma\colon \quad a(\pi) \to \gamma(a(\pi 2)) \ .$$

The symbol π represents the position of the root of a fork. Here, the value of the attribute a at any node w is obtained by putting a γ atop the value of a

* Corresponding author.

** The work of this author was partially supported by Deutsche Forschungsgemeinschaft, project DFG VO 1011/4-1.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 267–278, 2009.

at the second descendant of w, given that w is labeled by σ. (Note that the arrow is pointing in the direction opposite of information transport because the rule is understood as a term-rewriting rule.) Obviously, every rule of an attributed tree transducer contains exactly one free variable, and this variable is only constrained by the label restriction.

In contrast, rules of monadic datalog programs have an arbitrary number of free variables and constraints, thus specifying atom instance relationships much more freely. Consider the example rule

$$p(x) \leftarrow q(x_1), \mathrm{child}_1(x, x_1), \mathrm{child}_2(x, x_2), \mathrm{label}_\sigma(x_2), \mathrm{label}_\alpha(z) \ .$$

The comma is interpreted as conjunction. When instantiating the variables with nodes of t, we see that the value of the predicate p at any node w is determined by the value of the predicate q at the first descendant of w, given that the second descendant is labeled by σ and there is a node in t labeled by α.

In this paper, we introduce monadic datalog tree transducers, combining aspects of both approaches (under monadic datalog nomenclature)—the semantic domain of attributed tree transducers and the liberal style rules of monadic datalog. Attributed tree transducers can be described in our model without difficulty; the example rule mentioned above can be transferred as follows:

$$a(x) \leftarrow \gamma(a(y)) \,;\ \{\mathrm{child}_2(x, y), \mathrm{label}_\sigma(x)\} \ .$$

As opposed to monadic datalog, structural predicates—which are used to place constraints on the variables—and user-defined predicates have to be handled separately because the former are interpreted over the set of Booleans and the latter are interpreted over sets of output trees.

In the literature, different technical tools are known to define semantics for the formalisms we deal with here. Term rewriting [7] is usually used for attributed tree transducers [4]. In this approach, semantics can be evaluated in finite time exactly for a decidable subclass called *noncircular*. On the other hand, least fixpoint semantics is used for both attribute grammars [8] and monadic datalog. We define semantics for monadic datalog tree transducers in the latter way.

We give a coarse intuition. By computing all rule instances which are valid wrt an input tree t, we obtain a system of inequalities. The semantics associated with t is then obtained from the least interpretation satisfying the inequalities, where interpretations are ordered by the usual extension of the subset relation. Technically, the semantics is defined as the least fixpoint of an immediate consequence operator over the set of interpretations.

In Theorem 1, we will see that the semantics of a monadic datalog tree transducer can be evaluated in finite time iff the set of output trees is finite for every input tree. As a consequence, every semantics-preserving transformation on monadic datalog tree transducers automatically preserves computability. However, we do not know whether it is decidable for every monadic datalog tree transducer if its semantics can be evaluated in finite time.

Moreover, we will see in Theorem 2 that the class of tree transformations induced by monadic datalog tree transducers is a proper superclass of those

induced by attributed tree transducers. We establish a syntactic property called *restricted* which guarantees that the semantics can be computed by some attributed tree transducer (Theorem 3); this is shown by giving an effective construction. While restrictedness is not a necessary condition, we conjecture that it is the most general sufficient condition which only relies on the syntax of individual rules.

2 Preliminaries

We use $\mathbb{N}$ to denote the set of non-negative integers and $\mathbb{N}_+$ to denote $\mathbb{N} \setminus \{0\}$. For $n \in \mathbb{N}$ we abbreviate $\{i \in \mathbb{N}_+ \mid i \leq n\}$ by $[n]$. We denote the power set of a set A by $\mathcal{P}(A)$ and the set of finite subsets of A by $\mathcal{P}_{\text{fin}}(A)$.

Alphabets and ranked alphabets are defined as usual. We denote the set of strings over an alphabet Δ by Δ^*. We use $\Sigma^{(k)}$ to denote the set of k-ary symbols of a ranked alphabet Σ and write $\sigma^{(k)}$ for $\sigma \in \Sigma$ to indicate that the rank of σ is k. We call Σ *monadic* iff $\Sigma = \Sigma^{(0)} \cup \Sigma^{(1)}$. Let H be a set. The set $T_\Sigma(H)$ of *Σ-trees indexed by H* is defined as usual. We abbreviate $T_\Sigma(\emptyset)$ by T_Σ. Moreover, we define $\Sigma(H) = \{\sigma(h_1, \ldots, h_k) \mid k \in \mathbb{N}, \sigma \in \Sigma^{(k)}, h_1, \ldots, h_k \in H\} \subseteq T_\Sigma(H)$.

Let $t \in T_\Sigma$. By $\text{pos}(t) \subseteq \mathbb{N}^*$ we denote the set of positions of t and by $\text{ind}(t) \subseteq H$ the set of indices occurring in t; e.g. $t = \delta(\sigma(h_1), \alpha, h_2)$ and $H = \{h_1, h_2, h_3\}$ yield $\text{pos}(t) = \{\varepsilon, 1, 11, 2, 3\}$ and $\text{ind}(t) = \{h_1, h_2\}$. For $w, w' \in \text{pos}(t)$ we write $w \leq w'$ iff w is a prefix of w'. For every $w \in \text{pos}(t)$ and $t' \in T_\Sigma$ we write $t(w)$, $\text{rk}_t(w)$, $t|_w$, and $t[t']_w$ for the symbol that occurs at position w in t, the rank of $t(w)$, the subtree of t rooted at w, and the tree obtained from t by replacing the subtree rooted at w by t', respectively. The *height* $\text{height}(t)$ *of* t is defined as usual, where $\text{height}(\sigma) = 0$ for every $\sigma \in \Sigma^{(0)}$.

3 Monadic Datalog Tree Transducers

In this section we define the syntactic structure of monadic datalog tree transducers. Throughout this work, we fix an infinite set V, the elements of which being called *variables*. Moreover, we fix two ranked alphabets Σ and Δ (of *input symbols* and *output symbols*, respectively). Let $\text{maxrk}(\Sigma)$ denote the maximal rank of symbols in Σ. We define a ranked alphabet sp_Σ by letting

$$\text{sp}_\Sigma = \{\text{root}^{(1)}, \text{leaf}^{(1)}\} \cup \{\text{label}_\sigma^{(1)} \mid \sigma \in \Sigma\} \cup \{\text{child}_i^{(2)} \mid i \in [\text{maxrk}(\Sigma)]\} \ .$$

We refer to the elements of sp_Σ as *structural predicates over* Σ.

Definition 1. *A triple (P, R, q) is called a* monadic datalog tree transducer *(for short:* mdtt*) over Σ and Δ iff*

- *P is a monadic ranked alphabet (of* user-defined predicates*),*
- *$R \subseteq P(V) \times T_\Delta(P(V)) \times \mathcal{P}_{\text{fin}}(\text{sp}_\Sigma(V))$ is a finite set (of* rules*),*
- *$q \in P^{(1)}$ (called* query predicate*).*

A rule $r = (h, b, G) \in R$ *is denoted by* $h \leftarrow b$; G *and* h, b, *and* G *are called* head, body, *and* guard *of* r, *and are denoted by* r_{h}, r_{b}, *and* r_{G}, *respectively. We refer to the elements in* $P(V) \cup \mathrm{sp}_\Sigma(V)$ *as* atoms.

Now we define the semantics of mdtts. As a technical tool we will use common order theoretic concepts. An introduction into order theory can be found, e.g., in [9]. In the sequel let $M = (P, R, q)$ be an mdtt over Σ and Δ. Let $t \in T_\Sigma$. We refer to the elements in $P(\mathrm{pos}(t)) \cup \mathrm{sp}_\Sigma(\mathrm{pos}(t))$ as *atom instances (over* t*)*. The tree t constitutes a set $\mathrm{B}_t \subseteq \mathrm{sp}_\Sigma(\mathrm{pos}(t))$ of atom instances compatible with t defined as:

$$\begin{aligned}\mathrm{B}_t = {} & \{\mathrm{root}(\varepsilon)\} \cup \{\mathrm{leaf}(w) \mid w \in \mathrm{pos}(t), \mathrm{rk}_t(w) = 0\} \\ & \cup \{\mathrm{child}_i(w, wi) \mid w \in \mathrm{pos}(t), i \in [\mathrm{rk}_t(w)]\} \\ & \cup \{\mathrm{label}_\sigma(w) \mid w \in \mathrm{pos}(t), t(w) = \sigma\}\,.\end{aligned}$$

Let $r \in R$. We put $\mathrm{var}(r) = \mathrm{var}(r_{\mathrm{h}}) \cup \mathrm{var}(r_{\mathrm{b}}) \cup \mathrm{var}(r_{\mathrm{G}})$, where $\mathrm{var}(r_{\mathrm{h}})$ denotes the set of variables that occur in r_{h}, and likewise for r_{b} and r_{G}. An r, t*-variable assignment* is a mapping $\rho : \mathrm{var}(r) \to \mathrm{pos}(t)$. We lift ρ to $T_\Delta(P(\mathrm{var}(r)))$ and $\mathcal{P}(\mathrm{sp}_\Sigma(\mathrm{var}(r)))$ in an obvious manner; e.g. $\rho(v_1) = 1$ and $\rho(v_2) = \varepsilon$ yield $\rho(\delta(p(v_1), q(v_2))) = \delta(p(1), q(\varepsilon))$. We call ρ *valid* iff $\rho(r_{\mathrm{G}}) \subseteq \mathrm{B}_t$. For every $a \in P(\mathrm{pos}(t))$ we denote by $\Phi_{r,t,a}$ the set of valid r, t-variable assignments ρ such that $\rho(r_{\mathrm{h}}) = a$.

An *interpretation over* M *and* t is a mapping $I : P(\mathrm{pos}(t)) \to \mathcal{P}(T_\Delta)$. We denote the set of all interpretations over M and t by $\mathcal{I}_{M,t}$. We define a partial order $\leq$ on $\mathcal{I}_{M,t}$ as follows for every $I_1, I_2 \in \mathcal{I}_{M,t}$: $I_1 \leq I_2$ iff $I_1(a) \subseteq I_2(a)$ for every $a \in P(\mathrm{pos}(t))$. The resulting poset $(\mathcal{I}_{M,t}, \leq)$ is ω-complete. Let $I \in \mathcal{I}_{M,t}$. We extend I to $T_\Delta(P(\mathrm{pos}(t)))$ in the following sense: the I*-instance* $b[I] \in \mathcal{P}(T_\Delta)$ for $b \in T_\Delta(P(\mathrm{pos}(t)))$ is defined for every $\delta(b_1, \ldots, b_k) \in T_\Delta(P(\mathrm{pos}(t)))$ by

$$\delta(b_1, \ldots, b_k)[I] = \{\delta(t_1, \ldots, t_k) \mid t_1 \in b_1[I], \ldots, t_k \in b_k[I]\}\ .$$

We define the *immediate consequence operator* $\mathcal{T}_{M,t} : \mathcal{I}_{M,t} \to \mathcal{I}_{M,t}$ *over* M *and* t by letting for every $I \in \mathcal{I}_{M,t}$ and $a \in P(\mathrm{pos}(t))$:

$$\mathcal{T}_{M,t}(I)(a) = \bigcup_{r \in R} \bigcup_{\rho \in \Phi_{r,t,a}} \rho(r_b)[I]\ .$$

We use $\mathcal{T}^n_{M,t}$ to denote the n-fold composition of $\mathcal{T}_{M,t}$ for $n \in \mathbb{N}$. Obviously, $\mathcal{T}_{M,t}$ is ω-continuous and, thus, monotone wrt $\leq$. Hence, $\mathcal{T}^n_{M,t}(I_\emptyset) \leq \mathcal{T}^{n+1}_{M,t}(I_\emptyset)$ for every $n \in \mathbb{N}$, where $I_\emptyset$ denotes the minimal element of $\mathcal{I}_{M,t}$. Therefore, the fixpoint theorem of Knaster-Tarski [9, Theorem 1.5.7] yields that the least fixpoint of the mapping $\mathcal{T}_{M,t}$ is $\sup\{\mathcal{T}^n_{M,t}(I_\emptyset) \mid n \in \mathbb{N}\}$, which we denote by $\mathcal{T}^\omega_{M,t}$.

Definition 2. *The* tree transformation $[\![M]\!] : T_\Sigma \to \mathcal{P}(T_\Delta)$ computed by M *is defined for every* $t \in T_\Sigma$ *by* $[\![M]\!](t) = \mathcal{T}^\omega_{M,t}(q(\varepsilon))$. *The set of tree transformations computed by mdtts over* Σ *and* Δ *is denoted by* $\mathrm{MDTT}(\Sigma, \Delta)$.

Now we consider a simple example mdtt $M_{\mathrm{ex}} = (P, R, q)$ over $\Sigma = \{\alpha^{(0)}, \gamma^{(1)}\}$ and $\Delta = \Sigma \cup \{\delta^{(4)}\}$, where $P = \{q^{(1)}, r^{(1)}\}$ and R contains the rules

$$q(x) \leftarrow \delta(r(y), r(z), r(y), r(z))\,;\ \emptyset$$
$$r(x) \leftarrow \alpha\,;\ \{\mathrm{label}_\alpha(x)\}$$
$$r(x) \leftarrow \gamma(x_1)\,;\ \{\mathrm{label}_\gamma(x), \mathrm{child}_1(x, x_1)\}$$

Then $[\![M_{\mathrm{ex}}]\!](t) = \{\delta(t|_w, t|_v, t|_w, t|_v) \mid w, v \in \mathrm{pos}(t)\}$ for every $t \in T_\Sigma$.

If there is an $n \in \mathbb{N}$ with $\mathcal{T}^n_{M,t}(I_\emptyset)(q(\varepsilon)) = [\![M]\!](t)$, then $[\![M]\!](t)$ is obviously finite. It is easy to see that the converse holds as well because $\mathcal{T}_{M,t}$ is monotone wrt $\leq$. We show that in order to determine whether $[\![M]\!](t)$ is finite it suffices to consider $n = |P(\mathrm{pos}(t))|$.

Theorem 1. *Let $t \in T_\Sigma$. Then $[\![M]\!](t)$ is finite iff $[\![M]\!](t) = \mathcal{T}^n_{M,t}(\mathrm{I}_\emptyset)(q(\varepsilon))$, where $n = |P(\mathrm{pos}(t))|$.*

Corollary 1. *Let $h_{\mathrm{m}} = \max\{\mathrm{height}(r_{\mathrm{b}}) \mid r \in R\}$. Then for every $t \in T_\Sigma$ either the height of trees in $[\![M]\!](t)$ is unbounded or bounded by $h_{\mathrm{m}} \cdot |P(\mathrm{pos}(t))|$.*

We call M *executable* iff $[\![M]\!](t)$ is finite for every $t \in T_\Sigma$.

Let $M' = (P', R', q')$ be an mdtt over Σ and Δ. We call M and M' *equivalent* iff $[\![M]\!] = [\![M']\!]$. The following lemma provides a method for proving that M and M' are equivalent.

Lemma 1. *If for every $t \in T_\Sigma$ there is a fixpoint I of $\mathcal{T}_{M,t}$ and a fixpoint I' of $\mathcal{T}_{M',t}$ such that $I(q(\varepsilon)) = [\![M']\!](t)$ and $I'(q'(\varepsilon)) = [\![M]\!](t)$, then $[\![M]\!] = [\![M']\!]$.*

4 Comparison to Attributed Tree Transducers

We define attributed tree transducers according to [10, Sect. 2.3]. For the remainder of this paper, let $M = (P, R, q)$ be an mdtt over Σ and Δ. As a prerequisite, we fix an injective mapping $\mathrm{x} : \mathbb{N}^* \to V$; we denote $\mathrm{x}(\mathbb{N}^*)$ by X, $\mathrm{x}(w)$ by x_w, and $\{\mathrm{x}_1, \ldots, \mathrm{x}_k\}$ by X_k for every $w \in \mathbb{N}^*$ and $k \in \mathbb{N}$.

Definition 3. *The mdtt M is an* attributed tree transducer *(for short:* att*)* over Σ and Δ *iff there are disjoint sets A_{syn} and A_{inh} such that $P = P^{(1)} = A_{\mathrm{syn}} \cup A_{\mathrm{inh}}$, $q \in A_{\mathrm{syn}}$, and the following holds for every $r \in R$:*

- *either there are $k \in \mathbb{N}$, $\sigma \in \Sigma^{(k)}$, $a \in A_{\mathrm{syn}}(\{\mathrm{x}_\varepsilon\}) \cup A_{\mathrm{inh}}(\mathrm{X}_k)$ and $b \in T_\Delta(A_{\mathrm{syn}}(\mathrm{X}_k) \cup A_{\mathrm{inh}}(\{\mathrm{x}_\varepsilon\}))$ such that*

$$r = a \leftarrow b\,;\ \{\mathrm{label}_\sigma(\mathrm{x}_\varepsilon), \mathrm{child}_1(\mathrm{x}_\varepsilon, \mathrm{x}_1), \ldots, \mathrm{child}_k(\mathrm{x}_\varepsilon, \mathrm{x}_k)\}$$

- *or there are $a \in A_{\mathrm{inh}}(\{\mathrm{x}_\varepsilon\})$ and $b \in T_\Delta(A_{\mathrm{syn}}(\{\mathrm{x}_\varepsilon\}))$ such that*

$$r = a \leftarrow b\,;\ \{\mathrm{root}(\mathrm{x}_\varepsilon)\}\ .$$

The class of all tree transformations computed by attributed tree transducers over Σ and Δ is denoted by $\mathrm{ATT}(\Sigma, \Delta)$.

The class $\mathrm{ATT}(\Sigma, \Delta)$ defined here can be best compared with the class $\mathcal{TA}$ from [1], which differs from $\mathrm{ATT}(\Sigma, \Delta)$ in that it only takes *noncircular* atts

into account, a decidable syntactic subclass which guarantees that the semantics can be evaluated in finite time in their framework.

If M is an att, then for every rule $r \in R$ and every pair of variables $x, y \in \mathrm{var}(r)$, x and y are directly or indirectly connected by the child relation. In contrast, the mdtt M_{ex} does not have this property. We define a syntactic property based on variable connections which is satisfied by every att and show that it imposes a restriction on semantics.

Definition 4. *Let $r \in R$. We define the* variable connection relation $\sim_r$ of r *as the reflexive-transitive closure of* $\{(v_1, v_2) \in \mathrm{var}(r) \times \mathrm{var}(r) \mid \exists\, a \in r_{\mathrm{G}} \colon \{v_1, v_2\} \subseteq \mathrm{var}(a)\}$. *We call r* restricted *iff for every $w_1, w_2 \in \mathrm{pos}(r_{\mathrm{b}})$, we have: if $\emptyset \neq \big(\mathrm{var}(r_{\mathrm{b}}|_{w_1}) \times \mathrm{var}(r_{\mathrm{b}}|_{w_2})\big) \subseteq \sim_r$, then $\mathrm{var}(r_{\mathrm{b}}|_w) \times \mathrm{var}(r_{\mathrm{b}}|_w) \subseteq \sim_r$, where w is the longest common prefix of w_1 and w_2. Moreover, we call M restricted if every $r \in R$ is restricted. The class of all tree transformations computed by restricted mdtts over Σ and Δ is denoted by* $\mathrm{RMDTT}(\Sigma, \Delta)$.

Theorem 2. *There are Σ and Δ such that* $\mathrm{MDTT}(\Sigma, \Delta) \not\subseteq \mathrm{RMDTT}(\Sigma, \Delta)$.

Theorem 3. $\mathrm{RMDTT}(\Sigma, \Delta) = \mathrm{ATT}(\Sigma, \Delta)$.

In the rest of this section, we prove this theorem by introducing three syntactic properties for mdtts such that we obtain a sequence $P_1 \supset \ldots \supset P_5$ of syntactic classes where P_1 and P_5 are restricted and att, respectively, and for every $i \in [4]$ and $M \in P_i$ there is *effectively* an equivalent $M' \in P_{i+1}$. Hence, att is a normal form for restricted mdtts.

We note that restrictedness is no necessary condition for an mdtt to have a semantics in $\mathrm{ATT}(\Sigma, \Delta)$. However, we conjecture that it is the most general sufficient condition which only relies on the syntax of individual rules.

4.1 Connected

The first syntactic property is inspired by [5, Theorem 4.2]. A rule $r \in R$ is called *connected* iff $\sim_r = \mathrm{var}(r) \times \mathrm{var}(r)$. We call M *connected* iff every $r \in R$ is connected.

In the following, we motivate informally the constructions involved in giving a connected mdtt equivalent to M. To this end, consider the rule

$$r = p(x) \leftarrow \delta(q(x), p(y))\,;\ \{\mathrm{label}_\gamma(z)\}\ .$$

Let $t \in T_\Sigma$. We can make two observations: First, the variable z is not connected (i.e., related by $\sim_r$) to any variable occurring in r_{h} or r_{b}. Thus, if t contains a node labeled γ, we can omit the guard, and otherwise, we can omit the whole rule, each time preserving semantics for t. This idea can be used for a construction which does not depend on t, but suffice it to say that this involves duplication of all the remaining rules. Therefore, in order to have a terminating procedure, our construction deals with all rules at once. Second, the variable y, while trivially connected to some variable in the body, is not connected to the variable in

the head. In this case, we may replace $p(y)$ by $p'()$, where p' is a new nullary predicate, adding a rule $p'() \leftarrow p(y)\,;\ \emptyset$.

We define for every $r \in R$ the set of *independent guards* $I(r) = \{a \in r_{\mathrm{G}} \mid \forall v \in \mathrm{var}(a)\colon \forall v' \in \mathrm{var}(r_{\mathrm{h}}) \cup \mathrm{var}(r_{\mathrm{b}})\colon v \not\sim_r v'\}$ and for every $R' \subseteq R$ the set $I(R') = \bigcup_{r \in R'} I(r)$. We call M *semiconnected* iff r is connected or $I(r) = \emptyset$ for every $r \in R$.

Lemma 2. *There is a semiconnected mdtt M' equivalent to M such that M' is restricted if so is M.*

Proof. Without loss of generality, we assume that $\mathrm{var}(r_1) \cap \mathrm{var}(r_2) \neq \emptyset$ implies $r_1 = r_2$ for every $r_1, r_2 \in R$. For every $G \subseteq \mathrm{sp}_\Sigma(V)$ we define the *language accepted by* G by $L(G) = \{t \in T_\Sigma \mid \exists \rho : \mathrm{var}(G) \to \mathrm{pos}(t)\colon \rho(G) \subseteq \mathrm{B}_t\}$. Observe that $\mathrm{var}(G_1) \cap \mathrm{var}(G_2) = \emptyset$ implies $L(G_1 \cup G_2) = L(G_1) \cap L(G_2)$ for every $G_1, G_2 \subseteq \mathrm{sp}_\Sigma(V)$.

Let $R' \subseteq R$. We construct the semiconnected mdtt $\tau_1^{R'}(M) = (P, R_1, q)$ where $R_1 = \{r_{\mathrm{h}} \leftarrow r_{\mathrm{b}}\,;\ r_{\mathrm{G}} \setminus I(r) \mid r \in R'\}$. Note that the construction preserves restrictedness. If $t \in L(I(R'))$, then $[\![\tau_1^{R'}(M)]\!](t) \subseteq [\![M]\!](t)$. This holds because t satisfies all the guards which were omitted. If $t \in L(I(R')) \setminus \bigcup_{r \in R \setminus R'} L(I(r))$, then we even have equality for then we have indeed only removed rules which have no valid assignments.

Let $G \subseteq \mathrm{sp}_\Sigma(V)$ be finite. We construct an mdtt $\tau_2^G(M)$ such that the following holds: $[\![\tau_2^G(M)]\!](t) = [\![M]\!](t)$ if $t \in L(G)$, and $[\![\tau_2^G(M)]\!](t) = \emptyset$ otherwise. Thus, $[\![\tau_2^{I(R')}(\tau_1^{R'}(M))]\!](t) \subseteq [\![M]\!](t)$ for every $t \in T_\Sigma$. We define $\sim_G$ as the reflexive-transitive closure of $\{(a_1, a_2) \in G \times G \mid \mathrm{var}(a_1) \cap \mathrm{var}(a_2) \neq \emptyset\}$.

Let $\{G_1, \ldots, G_k\} = G/_{\sim_G}$. We construct $\tau_2^G(M) = (P \cup P', R \cup R', p_0)$ where $P' = \{p_0^{(1)}\} \cup \{p_1^{(0)}, \ldots, p_k^{(0)}\}$ is disjoint from P and (setting $p_{k+1} = q$) $R' = \{p_0(x) \leftarrow p_1()\,;\ \emptyset\} \cup \{p_i() \leftarrow p_{i+1}()\,;\ G_i \mid i \in [k]\}$. Note that this construction preserves semiconnectedness and restrictedness.

Finally, let J be a finite set and $(M_i \mid i \in J)$ a family of mdtts over Σ and Δ such that $M_i = (P_i, R_i, q_i)$ for every $i \in J$. The *union mdtt of* $(M_i \mid i \in J)$ is defined by $\biguplus_{i \in J} M_i = (P', R', q')$ where $P' = \{q'\} \cup \bigcup_{i \in J} P_i \times \{i\}$ and $R' = \{q'(x) \leftarrow (q_i, i)(x)\,;\ \emptyset \mid i \in J\} \cup \bigcup_{i \in J} R'_i$ where R'_i is obtained from R_i by replacing every occurrence of every $p \in P_i$ by (p, i). Clearly, $[\![\biguplus_{i \in J} M_i]\!](t) = \bigcup_{i \in J} [\![M_i]\!](t)$ for every $t \in T_\Sigma$, and $\biguplus_{i \in J} M_i$ is semiconnected (restricted) iff for every $i \in J$, so is M_i.

Now we are able to construct $M' = \biguplus_{R' \subseteq R} \tau_2^{I(R')}(\tau_1^{R'}(M))$. Clearly, M' is semiconnected, and M' is restricted if so is M. We show that $[\![M']\!] = [\![M]\!]$. Let $t \in T_\Sigma$ and $R' = \{r \in R \mid t \in L(I(r))\}$. Then it is easy to see that $t \in L(I(R')) \setminus \bigcup_{r \in R \setminus R'} L(I(r))$. Hence, $[\![M']\!](t) = [\![M]\!](t)$. □

Lemma 3. *Let M be restricted and semiconnected. Then there is a connected mdtt M' equivalent to M.*

Proof. Let $r \in R$, $C \in \mathrm{var}(r)/_{\sim_r}$ an equivalence class disjoint from $\mathrm{var}(r_{\mathrm{h}})$, and $G = \{a \in r_{\mathrm{G}} \mid \mathrm{var}(a) \subseteq C\}$. Since M is semiconnected, $C \cap \mathrm{var}(r_{\mathrm{b}}) \neq \emptyset$. Since M is restricted, there is a position $w \in \mathrm{pos}(r_{\mathrm{b}})$ such that for every $w' \in$

$\mathrm{pos}(r_\mathrm{b})$ with $r_\mathrm{b}(w) \in P(V)$, we have $w \leq w'$ iff $r_\mathrm{b}(w') \in P(C)$. Our construction replaces the subtree $r_\mathrm{b}|_w$ by an atom $p()$ (where p is a new predicate), removes from the guard of r the atoms in G, and adds a rule $p() \leftarrow r_\mathrm{b}|_w\,;\ G$. Clearly, this construction preserves semantics, semiconnectedness, and restrictedness. It is easy to see that applying it finitely often yields the desired result. □

We make an observation concerning connected mdtts which makes it possible to simplify further considerations significantly. Consider the rule

$$r = p(x) \leftarrow \delta(q(x), p(y))\,;\ \{\mathrm{child}_1(z, x), \mathrm{child}_2(z, y)\}\ .$$

For every $t \in T_\Sigma$ and valid r,t-variable assignment ρ, we obtain $\rho(x) = \rho(z)1$ and $\rho(y) = \rho(z)2$. Hence, we may reflect this fact in syntax by rephrasing r to

$$p(\mathrm{x}_1) \leftarrow \delta(q(\mathrm{x}_1), p(\mathrm{x}_2))\,;\ \{\mathrm{child}_1(\mathrm{x}_\varepsilon, \mathrm{x}_1), \mathrm{child}_2(\mathrm{x}_\varepsilon, \mathrm{x}_2)\}\ .$$

Now we generalize this. The mdtt M is called *positional* iff M is connected and for every $r \in R$, we have that $\mathrm{var}(r) \subseteq \mathrm{X}$, $\mathrm{x}_\varepsilon \in \mathrm{var}(r)$, and for every $w, w' \in \mathbb{N}^*$ and $i \in \mathbb{N}_+$, if $\mathrm{child}_i(\mathrm{x}_w, \mathrm{x}_{w'}) \in r_\mathrm{G}$, then $w' = wi$.

Lemma 4. *Let M be connected. There is a positional mdtt M' equivalent to M.*

Proof. Let $r \in R$. A mapping $\tau : \mathrm{var}(r) \to \mathbb{N}^*$ is called *r-position mapping* iff for every $v, v' \in \mathrm{var}(r)$ and $i \in \mathbb{N}_+$, if $\mathrm{child}_i(v, v') \in r_\mathrm{G}$, then $\tau(v)i = \tau(v')$. It can be shown that, if $\mathrm{var}(r) \neq \emptyset$, there is at most one r-position mapping τ such that $\tau(v) = \varepsilon$ for some $v \in \mathrm{var}(r)$, which we denote by τ_r. If $\mathrm{var}(r) = \emptyset$, we define τ_r to be the empty mapping. If τ_r does not exist, then $\Phi_{r,t,a}$ is empty for every t and a. It is decidable whether τ_r exists, and if it does, it can be computed. We construct $M' = (P, R', q)$ where $R' = \{\tau_r(r) \mid \exists r \in R\colon \tau_r \text{ exists}\}$ and $\tau_r(r)$ is obtained from r by replacing every occurrence of every $v \in \mathrm{var}(r)$ by $\mathrm{x}_{\tau_r(v)}$. □

4.2 Proper

The mdtt M is called *proper* iff all predicates in P are unary, i.e. $P^{(0)} = \emptyset$. In this section we show how to construct an equivalent mdtt M' that is proper. This construction maintains the positional property.

Lemma 5. *There is a proper mdtt M' equivalent to M such that M' is positional if so is M.*

Proof. The idea of our construction is to replace every $p \in P^{(0)}$ by a unary variant $\bar{p}$ of p. More precisely, we have to replace every occurrence of the atom $p()$ by the atom $\bar{p}(\mathrm{x}_\varepsilon)$; this guarantees that M' is positional if M is positional. Additionally we add rules that ensure that the value of the atom instance $\bar{p}(w)$ is equal for every position $w \in \mathrm{pos}(t)$.

We define $M' = (P', R', q)$ as follows: $P' = P^{(1)} \cup \{\bar{p}^{(1)} \mid p \in P^{(0)}\}$ and $R' = \{\bar{r} \mid r \in R\} \cup \{r^{\mathrm{up}}_{p,i}, r^{\mathrm{down}}_{p,i} \mid p \in P^{(0)}, i \in [\mathrm{maxrk}(\Sigma)]\}$, where $\bar{r}$ is obtained

from r by replacing for every $p \in P^{(0)}$ every occurrence of $p()$ by $\bar{p}(\mathrm{x}_\varepsilon)$, and for every $p \in P^{(0)}$ and $i \in [\mathrm{maxrk}(\Sigma)]$ we let

$$r^{\mathrm{up}}_{p,i} = \bar{p}(\mathrm{x}_\varepsilon) \leftarrow \bar{p}(\mathrm{x}_i)\,;\ \mathrm{child}_i(\mathrm{x}_\varepsilon, \mathrm{x}_i)\ ,$$
$$r^{\mathrm{down}}_{p,i} = \bar{p}(\mathrm{x}_i) \leftarrow \bar{p}(\mathrm{x}_\varepsilon)\,;\ \mathrm{child}_i(\mathrm{x}_\varepsilon, \mathrm{x}_i)\ .$$

Let $p \in P^{(0)}$. Then we have $\mathcal{T}^{\omega}_{M',t}(\bar{p}(w)) = \mathcal{T}^{\omega}_{M,t}(p())$ for every $w \in \mathrm{pos}(t)$. Then it is easy to see that M' is equivalent to M. Moreover, M' is obviously proper and it is positional if M is so. □

4.3 Local

If M is positional, then the set $\mathrm{var}(r)$ can be an arbitrary finite subset of X for every $r \in R$. However, for atts, $\mathrm{var}(r)$ must be a subset of X_ε, where $\mathrm{X}_\varepsilon = \{\mathrm{x}_\varepsilon\} \cup \{\mathrm{x}_i \mid i \in \mathbb{N}_+\}$. Therefore, the rules of an att take effect only on a local part of the input tree. Formalizing this, we define an order $\leq$ on X by letting $\mathrm{x}_w \leq \mathrm{x}_{w'}$ iff w is a prefix of w' for every $w, w' \in \mathbb{N}^*$. We let $\mathrm{var}_1(a) = \min(\mathrm{var}(a))$ for every $a \in \mathrm{sp}_\Sigma(\mathrm{X})$, and call an $r \in R$ *local* iff $\mathrm{var}_1(a) = \mathrm{x}_\varepsilon$ for every $a \in r_{\mathrm{G}}$. The mdtt M is called *local* iff it is proper, positional, and every rule in R is local.

In this section we show that for every proper and positional mdtt there is an equivalent local one. First we give a brief explanation of our construction. Let r be the rule $q(\mathrm{x}_\varepsilon) \leftarrow p(\mathrm{x}_{21})\,;\ \{\mathrm{child}_2(\mathrm{x}_\varepsilon, \mathrm{x}_2), \mathrm{child}_1(\mathrm{x}_2, \mathrm{x}_{21}), \mathrm{label}_\sigma(\mathrm{x}_2)\}$. In order to make r local we split it into local components while introducing an auxiliary predicate p'; hence, we obtain two rules

$$q(\mathrm{x}_\varepsilon) \leftarrow p'(\mathrm{x}_2)\,;\ \{\mathrm{child}_2(\mathrm{x}_\varepsilon, \mathrm{x}_2)\}\ ,$$
$$p'(\mathrm{x}_\varepsilon) \leftarrow p(\mathrm{x}_1)\,;\ \{\mathrm{child}_1(\mathrm{x}_\varepsilon, \mathrm{x}_1), \mathrm{label}_\sigma(\mathrm{x}_\varepsilon)\}\ .$$

Special care has to be taken if both the head and the body of the given rule belong entirely to one of the local components: consider the rule r' which originates from r by replacing the variable x_{21} in the body by x_ε. In this case we have to make a detour and construct three rules

$$q(\mathrm{x}_\varepsilon) \leftarrow p'(\mathrm{x}_2)\,;\ \{\mathrm{child}_2(\mathrm{x}_\varepsilon, \mathrm{x}_2)\}\ ,$$
$$p'(\mathrm{x}_\varepsilon) \leftarrow p''(\mathrm{x}_\varepsilon)\,;\ \{\mathrm{child}_1(\mathrm{x}_\varepsilon, \mathrm{x}_1), \mathrm{label}_\sigma(\mathrm{x}_\varepsilon)\}\ ,$$
$$p''(\mathrm{x}_2) \leftarrow p(\mathrm{x}_\varepsilon)\,;\ \{\mathrm{child}_2(\mathrm{x}_\varepsilon, \mathrm{x}_2)\}\ .$$

Lemma 6. *Let M be positional and proper. Then there is a local mdtt M_{loc} equivalent to M.*

Proof. We do not construct the mdtt M_{loc} at once. Rather we give a construction of a positional and proper mdtt M_1 and argue that a finite number of applications of this construction leads to M_{loc}. Assume that M is not yet local. Select a rule $r \in R$ that is not local and select a maximal x in the set $\{\mathrm{var}_1(a) \mid a \in r_{\mathrm{G}}\}$. Then $x \neq \mathrm{x}_\varepsilon$.

We let $A = \{a \in \{r_\mathrm{h}\} \cup \mathrm{ind}(r_\mathrm{b}) \mid \exists\, v \in \mathrm{var}(a) : v > x\}$ and $G = \{a \in r_\mathrm{G} \mid \mathrm{var}_1(a) = x\}$. We define an intermediate mdtt $M_0 = (P_0, R_0, q)$ by case distinction.

Case 1: $A = \emptyset$. Then $P_0 = P \cup \{p_1, p_2\}$ and $R_0 = (R \setminus \{r\}) \cup \{r_1, r_2, r_3\}$:

$$r_1 = r_\mathrm{h} \leftarrow p_1(x)\,;\ r_\mathrm{G} \setminus G\ ,$$
$$r_2 = p_1(x) \leftarrow p_2(x)\,;\ G\ ,$$
$$r_3 = p_2(x) \leftarrow r_b\,;\ r_\mathrm{G} \setminus G\ .$$

Case 2: $A \neq \emptyset$. Then $P_0 = P \cup \{\bar{a} \mid a \in A\}$ where $\bar{a}$ is a new predicate for every $a \in A$, and $R_0 = (R \setminus \{r\}) \cup \{\bar{r}\} \cup R'$, where $\bar{r}$ is obtained from r by replacing every occurence of every $a \in A$ by $\bar{a}(x)$ and replacing r_G by $r_\mathrm{G} \setminus G$, and R' is the smallest set such that for every $a \in A$:

- if $a = r_\mathrm{h}$, then R' contains the rule $a \leftarrow \bar{a}(x)\,;\ G$
- if $a \in \mathrm{ind}(r_\mathrm{b})$, then R' contains the rule $\bar{a}(x) \leftarrow a\,;\ G$

The mdtt M_1 is obtained from M_0 by making it positional.

Obviously, every rule r' that is added in this construction is either local or satisfies $r'_\mathrm{G} \subset r_\mathrm{G}$. Since a multiset order is terminating whenever it is defined based on a terminating strict order (see Theorem 2.5.5 in [7]) we obtain that a finite number of applications of this construction leads to a local mdtt M_loc.

4.4 Attributed Tree Transducers

In this section we show that we can transform every local mdtt M into an equivalent attributed tree transducer.

Lemma 7. *Let M be local. There is an att M' equivalent to M.*

Proof (Sketch). First let us list the syntactic differences between local mdtts and atts:

1. for atts certain atoms are mandatory in guards, e.g. $\mathrm{label}_\sigma(\mathrm{x}_\varepsilon)$ or $\mathrm{root}(\mathrm{x}_\varepsilon)$, and certain atoms are not allowed to occur, e.g. $\mathrm{leaf}(\mathrm{x}_\varepsilon)$,
2. the set of user-defined predicates is partitioned into inherited and synthesized attributes,
3. rules whose guard contains $\mathrm{root}(\mathrm{x}_\varepsilon)$ have to be of a very restricted form.

Our construction of M' is divided into three phases, each of which dealing with one of the syntactic differences in the order listed above.

Phase 1. This construction is split into five steps. In the first step we drop all rules r such that there are distinct $\sigma, \sigma' \in \Sigma$ with $\{\mathrm{label}_\sigma(\mathrm{x}_\varepsilon), \mathrm{label}_{\sigma'}(\mathrm{x}_\varepsilon)\} \subseteq r_\mathrm{G}$, since these rules are obviously inconsistent. In the second step we take care of all rules whose guard does not yet contain $\mathrm{label}_\sigma(\mathrm{x}_\varepsilon)$ for any $\sigma \in \Sigma$. For such a rule r we add for every $\sigma \in \Sigma$ a copy of r that additonally contains $\mathrm{label}_\sigma(\mathrm{x}_\varepsilon)$

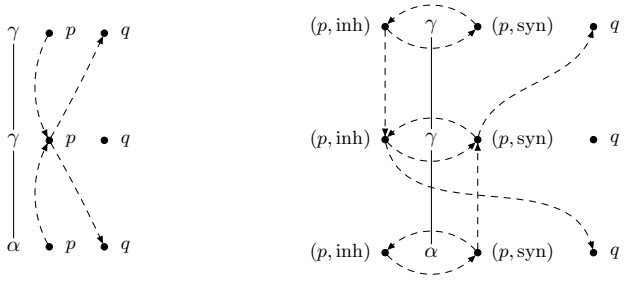


Fig. 1. Illustration of the construction of Phase 2

in the guard. Afterwards we remove r. In the resulting mdtt we have that for every rule r' there is a unique $\sigma \in \Sigma$ with $\mathrm{label}_\sigma(\mathrm{x}_\varepsilon) \in r'_{\mathrm{G}}$; we denote this σ by $\sigma_{r'}$. In the third step we add to the guard of every rule r the atoms $\mathrm{child}_1(\mathrm{x}_\varepsilon, \mathrm{x}_1), \ldots, \mathrm{child}_k(\mathrm{x}_\varepsilon, \mathrm{x}_k)$ where $k = \mathrm{rk}(\sigma_r)$.

In the fourth step we remove all rules that are obviously inconsistent; these are all rules r such that (i) $\mathrm{rk}(\sigma_r) > 0$ and $\mathrm{leaf}(\mathrm{x}_\varepsilon) \in r_{\mathrm{G}}$ or (ii) $\mathrm{child}_i(\mathrm{x}_\varepsilon, \mathrm{x}_i) \in r_{\mathrm{G}}$ for some $i > \mathrm{rk}(\sigma_r)$. In the fifth step we consider all rules r whose guard contains $\mathrm{leaf}(\mathrm{x}_\varepsilon)$: by the construction in the fourth step it is obvious that $\mathrm{rk}(\sigma_r) = 0$. Therefore $\mathrm{leaf}(\mathrm{x}_\varepsilon)$ is redundant in r_{G} and we can simply drop it. The constructions that are carried out in each step yield an mdtt equivalent to the original one. Note that we deal with all rules that contain $\mathrm{root}(\mathrm{x}_\varepsilon)$ in Phase 3.

Phase 2. Suppose that M has already passed Phase 1. We give an intuitive description of our construction. Assume that $P = \{p, q\}$ and consider the input tree $\gamma(\gamma(\alpha))$. Furthermore assume that R determines data transport from $p(\varepsilon)$ to $p(1)$, $p(11)$ to $p(1)$, $p(1)$ to $q(\varepsilon)$, and $p(1)$ to $q(11)$. This situation is depicted on the left-hand side of Fig. 1, where the data transport is represented by dashed arrows. Obviously, p behaves both like a synthesized and an inherited attribute. Thus, we have to replace p by a predicate (p, syn) (simulating its synthesizing behavior, i.e., transporting data bottom-up) and a predicate (p, inh) (simulating its inheriting behavior, i.e., transporting data top-down). Since (p, syn) should also receive all the data that p receives from the top part of the tree, we need to introduce a rule which transports data from (p, inh) to (p, syn). The same holds for (p, inh): here we need to add a rule for transporting data from (p, syn) to (p, inh) (note that we need to take special care for the root of the tree). This is illustrated on the right-hand side of Fig. 1.

Phase 3. Suppose that M has already passed Phases 1 and 2. If M is not already an att, then there is a rule r containing $\mathrm{root}(\mathrm{x}_\varepsilon)$. We remove this rule from M, add two new user-defined predicates (r, syn) and (r, inh), and add the rules

$$\begin{aligned} r_{\mathrm{h}} &\leftarrow (r, \mathrm{inh})(\mathrm{x}_\varepsilon)\,;\ r_{\mathrm{G}} \setminus \{\mathrm{root}(\mathrm{x}_\varepsilon)\}\ , \\ (r, \mathrm{inh})(\mathrm{x}_\varepsilon) &\leftarrow (r, \mathrm{syn})(\mathrm{x}_\varepsilon)\,;\ \{\mathrm{root}(\mathrm{x}_\varepsilon)\}\ , \\ (r, \mathrm{syn})(\mathrm{x}_\varepsilon) &\leftarrow r_{\mathrm{b}}\,;\ r_{\mathrm{G}} \setminus \{\mathrm{root}(\mathrm{x}_\varepsilon)\}\ . \end{aligned}$$

Obviously, these three new rules comply with the definition of atts. Thus, by doing this construction for every non-compliant rule we will eventually obtain an att equivalent to M. □

4.5 Open Problems

We did not yet address the following problems. (1) Is the subclass of executable mdtts decidable? (2) Do executable atts have the same computational power as noncircular atts [4]? (3) Characterizing the class $\mathrm{MDTT}(\Sigma, \Delta) \setminus \mathrm{RMDTT}(\Sigma, \Delta)$.

Acknowledgements. The authors are grateful to Heiko Vogler for helpful discussions and valuable suggestions.

References

1. Fülöp, Z.: On attributed tree transducers. Acta Cybernet 5, 261–279 (1981)
2. Knuth, D.: Semantics of context–free languages. Math. Systems Theory 2, 127–145 (1968)
3. Deransart, P., Jourdan, M., Lorho, B.: Attribute Grammars - Definitions, Systems and Bibliography. In: Deransart, P., Lorho, B., Jourdan, M. (eds.) Attribute Grammars. LNCS, vol. 323. Springer, Heidelberg (1988)
4. Fülöp, Z., Vogler, H.: Syntax-directed semantics — Formal Models Based on Tree Transducers. Monogr. Theoret. Comput. Sci. EATCS Ser. Springer, Heidelberg (1998)
5. Gottlob, G., Koch, C.: Monadic datalog and the expressive power of languages for web information extraction. J. ACM 51(1), 74–113 (2004)
6. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
7. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
8. Chirica, L.M., Martin, D.F.: An Order-Algebraic Definition of Knuthian Semantics. Math. Systems Theory 13, 1–27 (1979)
9. Wechler, W.: Universal Algebra for Computer Scientists, 1st edn. Monogr. Theoret. Comput. Sci. EATCS Ser., vol. 25. Springer, Heidelberg (1992)
10. Bloem, R., Engelfriet, J.: A comparison of tree transductions defined by monadic second order logic and by attribute grammars. J. Comput. System Sci. 61, 1–50 (2000)

On Extended Regular Expressions

Benjamin Carle and Paliath Narendran

Dept. of Computer Science
University at Albany–SUNY
Albany, NY 12222

Abstract. In this paper we extend the work of Campeanu, Salomaa and Yu [1] on extended regular expressions featured in the Unix utility *egrep* and the popular scripting language *Perl*. We settle the open issue of closure under intersection and provide an improved pumping lemma that will show that a larger class of languages is not recognizable by extended regular expressions. We also investigate some questions regarding extended multi-pattern languages introduced by Nagy in [2].

1 Introduction

Grep, a well-known command line search utility, is used regularly on Unix and other operating systems to find matching lines in files or standard input. Grep uses regular expressions to match patterns, thereby allowing a user to quickly find important data in very large files or command output. *Egrep,* a variant of grep, uses extended regular expressions[1] to increase the set of languages recognizable by the utility. Because the set of languages recognized by egrep is larger than that of theoretical regular expressions, it is important to understand the expressive power of this utility.

In this paper we extend the pioneering work of [1]; we show that the family of languages recognizable by extended regular expressions is not closed under intersection, thereby settling an open problem. Furthermore, we introduce a different pumping lemma and use that lemma to show a class of languages that satisfy the pumping property of [1] but are not expressible by extended regular expressions. We also investigate some decidability and complexity issues.

We also consider the work of Nagy [2] which extends the multi-pattern languages (MPL) defined by Kari, Mateescu, Paun and Salomaa [3]. We show that the class Nagy defines, which we call *extended multi-pattern languages* (EMPL), is a strict subclass of the family of languages recognized by egrep. We also settle some questions that are left open in [2].

2 Definitions

The syntax of extended regular expressions as in *egrep* and *Perl* is defined in [1]. Standard regular expressions, as specified in formal language theory, are

[1] Note our definition of extended regular expressions includes backreferences unlike the definition given in some other sources.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 279–289, 2009.

extended using *backreferences.* The backreference $\backslash n$ stands for the string previously matched by the regular expression between the n^{th} left parenthesis and the corresponding right parenthesis. As is well-known, this significantly increases expressive power; for instance, the expression `((aa)+a)\1*` specifies the language

$$\{\, a^i \mid i > 0 \text{ and } i \text{ is not a power of } 2\}$$

which is not even context-free. Similarly `(a+)(b+)\1\2` specifies

$$\{\, a^i b^j a^i b^j \mid i, j > 0\}$$

which is not context-free either.

For clarity, let us number left parentheses, starting with 1, from the left. Give the same numbers to the corresponding (matching) right parentheses.

$$\underset{1}{(}\ldots\underset{2}{(}\ldots\underset{2}{)}\ldots\underset{3}{(}\ldots\underset{3}{)}\ldots\underset{1}{)}$$

As in [1] we assume that any occurrence of a backreference `\m` in an extended regular expression is preceded by $\underset{m}{)}$.

Matching a string with an extended regular expression (eregex-matching) is often defined as follows (paraphrasing [4]):

1. If a is a symbol in the alphabet, then a matches a.
2. if r matches a string x, then $\underset{i}{(}\, r \,\underset{i}{)}$ matches x and the value x is assigned to $\backslash i$.
3. $\backslash j$ matches the string that has been assigned to it.
4. if r_1 and r_2 are eregexes, then $r_1 \cup r_2$ matches any string matched by either r_1 or r_2.
5. if r_1 and r_2 are eregexes, then $r_1 r_2$ matches any string of the form xy where r_1 matches x and r_2 matches y.
6. if r is an eregex, then r^* matches any string of the form $x_1 \ldots x_n$, $n \geq 0$, where r matches each x_i $(1 \leq i \leq n)$.

A more precise definition of a match is given in [1] using ordered trees. We give below that definition too, with a slight modification. Positions in an ordered tree are denoted by sequences of positive integers, with the empty sequence denoting the root position. (See [5] for a formal definition.) Note that left-to-right lexicographic order $\prec_{lex}$ among positions corresponds to *pre-order* traversal.

An ordered tree T is a valid match-tree for w and α if and only if:

1. The root of T has the label $(w,\ \alpha)$.
2. For every node $u \in dom(T)$,
   (a) if $T(u) = (w,\ a)$ for some $a \in \Sigma$, then u is a leaf node and $w = a$.
   (b) if $T(u) = (w,\ \beta_1\beta_2)$, then u has two children labeled, respectively, by $(w_1,\ \beta_1)$ and $(w_2,\ \beta_2)$ where $w_1 w_2 = w$.
   (c) if $T(u) = (w,\ \beta_1|\beta_2)$, then u has one child labeled by either $(w,\ \beta_1)$ or $(w,\ \beta_2)$.

(d) if $T(u) = (w,\ \beta^*)$, then either u is a leaf node and $w = \lambda$ or u has $k \geq 1$ children labeled by $(w_1,\ \beta), \ldots, (w_k,\ \beta)$ where each $w_i \in \Sigma^+$, and $w = w_1 \ldots w_k$.

(e) if $T(u) = (w,\ (\underset{i}{\ }\gamma\underset{i}{\ }))$, then it has one child labeled by $(w,\ \gamma)$.

(f) if $T(u) = (w,\ \backslash m)$, then u is a leaf node, $(\underset{m}{\ }\beta\underset{m}{\ })$ is a subexpression of α, and there is a node v to the left of u such that $T(v) = (w,\ (\underset{m}{\ }\beta\underset{m}{\ }))$ and no node between v and u has $(\underset{m}{\ }\beta\underset{m}{\ })$ in its label. In other words, w is the string previously (in the left-to-right pre-order) matched by $(\underset{m}{\ }\beta\underset{m}{\ })$.

The difference between this definition and the one in [1] is that unassigned backreferences are not set to the empty string λ as default in our definition. Thus there is no valid match-tree for b and `((aa)|\2b)`.

The language denoted by an extended regular expression α is defined as

$$\mathcal{L}(\alpha) = \{\, w \in \Sigma^* \mid (w,\ \alpha) \text{ is the label at the root of a valid match-tree } \}.$$

Let EREG be the family of languages defined by extended regular expressions. A language L is an EREG language if and only if there is an extended regular expression α such that $L = \mathcal{L}(\alpha)$. In relation to the regular languages (REG), it can be seen that

$$REG \subset EREG.$$

3 The Results on EREG Languages

Campeanu, Salomaa and Yu [1] proved the following pumping lemma for EREG languages. To the best of our knowledge, this is the only pumping lemma of its kind.

Lemma 1. (The CSY Pumping Lemma) [1] *Let α be an extended regular expression. Then there is a constant $N > 0$ such that if $w \in \mathcal{L}(\alpha)$ and $|w| > N$, then there is a decomposition $w = x_0 y x_1 y \cdots y x_m$, for some $m \geq 1$, such that*

1. $|x_0 y| < N$,
2. $|y| \geq 1$, *and*
3. $x_0 y^j x_1 y^j \cdots y^j x_m \in \mathcal{L}(\alpha)$ *for all* $j > 0$.

Lemma 2. *The language*

$$\mathcal{S} \ = \ \{\, a^i b a^{i+1} b a^k \mid k = i(i+1)k' \text{ for some } k' > 0, i > 0 \,\}$$

is not an EREG language.

Proof: Assume $\mathcal{S}$ is expressed by an eregex and let N be the constant given by the CSY pumping lemma. Consider $w = a^N b a^{N+1} b a^{N(N+1)}$. Then there is a decomposition $w = x_0 y x_1 y \ldots y x_m$ for some $m \geq 1$ and from the pumping

lemma $y = a^p$ for some $p \geq 1$. Since $|x_0 y| < N$ there must be at least one occurrence of y in a^N. Assume there are $q \geq 1$ occurrences of y in a^N. By (3) from the CSY pumping lemma there must also be q occurrences of y in a^{N+1} as otherwise $x_0 y^2 x_1 y^2 \dots y^2 x_m \notin \mathcal{S}$. Let r be the number of occurrences of y in $a^{N(N+1)}$ and note that $N(N+1) \geq rp \geq 0$. Now consider $x_0 y^2 x_1 y^2 \dots y^2 x_m = a^{N+qp} b a^{N+1+qp} b a^{N(N+1)+rp} \in \mathcal{S}$. Then

$$k_2(N+qp)(N+1+qp) = N(N+1) + rp$$

for some k_2. Since $rp \leq N(N+1)$, $N(N+1) + rp \leq 2(N(N+1))$. Since $qp \geq 1$, $(N+qp)(N+1+qp) \geq (N+1)(N+2) > N(N+1)$. Thus $k_2(N+qp)(N+1+qp) \geq k_2(N+1)(N+2) > k_2 N(N+1)$. $k_2 N(N+1) < 2(N(N+1))$ is only true for $k_2 = 1$. Thus we have $(N+qp)(N+1+qp) = N(N+1) + rp$, so

$$rp = q^2 p^2 + qp(2N+1) \tag{1}$$

Now consider $x_0 y^3 x_1 y^3 \dots y^3 x_m = a^{N+2qp} b a^{N+1+2qp} b a^{N(N+1)+2rp} \in \mathcal{S}$. Then it must be that

$$k_3(N+2qp)(N+1+2qp) = N(N+1) + 2rp$$

for some k_3. But note that $(N+2qp)(N+1+2qp) = N(N+1) + \mathbf{4q^2p^2} + 2qp(2N+1)$, whereas $N(N+1) + 2rp = N(N+1) + \mathbf{2q^2p^2} + 2qp(2N+1)$ by (1). Hence such a k_3 cannot exist. □

Theorem 1. *EREG languages are not closed under intersection.*

Proof: The language $\mathcal{S}$ of the previous lemma is the intersection of

$$\mathcal{L}(\texttt{(a+)b(\1a)b\1+}) \text{ and } \mathcal{L}(\texttt{(a+)b(\1a)b\2+}).$$ □

We can show, by a reduction from the membership problem for phrase structured grammars, that

Theorem 2. *The following problem is undecidable:*

Emptiness of Intersection of Extended Regular Expressions (EIERE):

Instance: Two eregexes α and β.
Question: Is $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$ empty?

Proof: The reduction is from the membership problem for phrase-structure grammars (MPSG), a known undecidable problem, as mentioned earlier. A phrase structure grammar is specified as $G = (V, \Sigma, P, S)$ where V is a finite nonempty set called the *total vocabulary*, $\Sigma \subseteq V$ is a finite nonempty set called the *terminal alphabet*, $N = V - \Sigma$ is the nonterminal alphabet, $S \in N$ is the *start symbol* and P is a finite set of rules (or productions) of the form $l \to r$ where $l \in V^* N V^*$ and $r \in V^*$. The membership problem MPSG is defined as follows:

Instance: Phrase-structure grammar $G = (V, \Sigma, P, S)$ and a string $w \in \Sigma^*$
Question: Is $w \in \mathcal{L}(G)$?

Given an instance of MPSG, we construct an instance of EIERE as follows: For each production $\mathbf{l_i} \to \mathbf{r_i} \in P$ let $\alpha_i = \#((\Sigma)*)\,\mathbf{l_i}\,((\Sigma)*)\#\backslash 1\,\mathbf{r_i}\,\backslash 3$, for some $\# \notin V$, for $1 \le i \le |P|$. Let $\alpha = (\alpha_1|\alpha_2|\ldots|\alpha_n)*$. Note that the backreferences will have to be renumbered, replacing each $\backslash j$ in α_i with $\backslash j'$ where $j' = 4(i-1)+j+1$.

So, $\mathcal{L}(\alpha)$ is the language of sequences of derivation steps (though not necessarily continuous).

$$\underbrace{\#w_1\#w_1'}_{\alpha_{i_1}}\underbrace{\#w_2\#w_2'}_{\alpha_{i_2}}\underbrace{\#w_3\#w_3'}_{\alpha_{i_3}}\ \ldots\ \underbrace{\#w_n\#w_n'}_{\alpha_{i_n}}$$

where each $w_i = x\mathbf{l}y$ and $w_i' = x\mathbf{r}y$ for $1 \le i \le n$ for some $x, y \in \Sigma*$ and some $\mathbf{l} \to \mathbf{r} \in P$.

We now define β to enforce derivation continuity. Consider $\beta_0 = \#((\Sigma)*)\,\#\,\backslash 1$ which matches strings of the form $\#w_i\#w_i$ for $w_i \in \Sigma*$. Let $\beta = \#S(\#((\Sigma)*)\,\#\,\backslash 2) * \#w$. Then $\mathcal{L}(\beta)$ contains all strings of the form

$$\#S\,\underbrace{\#w_1\#w_1}\underbrace{\#w_2\#w_2}\ \ldots\ \underbrace{\#w_n\#w_n}\#w \ \text{ for some } \ n \ge 0$$

where each underbraced segment matches β_0.

Thus $\mathcal{L}(\beta)$ is the language of all continuous steps. (Not necessarily derivation steps from G.)

Finally, if we take the intersection of the two languages, namely $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$, we get strings of the form

$$\#S\,\underbrace{\#w_1\#w_1}\underbrace{\#w_2\#w_2}\ \ldots\ \underbrace{\#w_{n-1}\#w_{n-1}}\#w_n$$

where $w_i = x\mathbf{l}y$ and $w_{i+1} = x\mathbf{r}y$ for each $1 \le i < n; x, y \in \Sigma*$; and $\mathbf{l} \to \mathbf{r} \in P$ and $\mathbf{S} \to \mathbf{w_1} \in P$, $w_n = w$. That is, we get sequences of continuous derivation steps beginning at S and ending in w.

Therefore, $w \in \mathcal{L}(G)$ iff $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta) \ne \phi$ □

Lemma 3. *Let α be an eregex. Then for any $k > 0$ there are positive integers $N(k)$ and m such that if $w \in \mathcal{L}(\alpha)$ and $|w| > N(k)$ then w has a decomposition $w = x_0yx_1y\cdots yx_{m'}$ $(m' < m)$ such that*

1. *$|y| \ge k$, and*
2. *$x_0y^jx_1y^j\cdots y^jx_{m'} \in \mathcal{L}(\alpha)$ for all $j > 0$.*

Proof: Let $N(k) = |\alpha|2^tk$ where t is the number of backreferences in α. Then if $|w| > N$ there is a substring of w of length $\ge k$ that matches a Kleene star in α. (Each backreference can at most double the length of the word it matches.) Let $m = t + 1$.

Let $w = x_0yz$ where y is the *rightmost* largest substring of w that matches a Kleene star. Then clearly $|y| \ge k$. Let t' equal the number of (direct or indirect) backreferences to any expression that contains this star. Let $m' = t' + 1$. Let $z = x_1yx_2yx_3\ldots x_{m'}$ where the multiple instances of y correspond to these backreferences. Then $w = x_0yx_1yx_2y\ldots yx_{m'}$ and clearly $x_0y^jx_1y^jx_2y^j\ldots y^jx_{m'} \in \mathcal{L}(\alpha)$ for all $j \ge 1$. □

Lemma 4. *The language* $\mathcal{P} = \{ wcw^R \mid w \in \{a,b\}^* \}$ *satisfies the CSY pumping property. (*w^R *stands for the reverse of* w*.)*

Lemma 5. *The language* $\mathcal{P} = \{ wcw^R \mid w \in \{a,b\}^* \}$ *is not an EREG language.*

Proof: Assume $\mathcal{P}$ is an EREG language. Let $k = 5$. Let $N(k)$ and m be the constants given by Lemma 5. Consider $w = (abaabb)^{N(k)}c(bbaaba)^{N(k)}$. Then there is a decomposition $w = x_0yx_1y\cdots yx_{m'}$ for some $m' < m$. By Lemma 3, $|y| \geq 5$. $(k = 5)$ Observe that $(abaabb)^{N(k)}$ and $(bbaaba)^{N(k)}$ do not share any common substrings of length ≥ 5. Therefore, y must occur to the left or the right of c, but not both. Consequently, $x_0y^2x_1y^2\ldots y^2x_{m'} \notin \mathcal{P}$. □

We now consider the Matching Problem for Extended Regular Expressions (MERE):

Instance: An eregex α and a string $w \in \Sigma^*$.
Question: Is (w,α) the label at the root of a valid match tree?

This has been shown to be NP-complete [4]. It turns out that the problem is NP-complete *even if the target alphabet is unary*:

Theorem 3. *The matching problem for extended regular expressions is NP-complete even when the target (subject) string is over a unary alphabet.*

Proof: Membership in NP follows from the earlier result. NP-hardness can be proved by a reduction from the vertex cover problem.
Vertex Cover (VC)

Instance: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Question: Is there a $V' \subseteq V$ such that $|V'| \leq k$ and
$\forall (u, v) \in E : \; u \in V' \vee v \in V'$?

Given an instance of VC, construct an instance of MERE as follows: Define $n = |V|$ and $m = |E|$. Without loss of generality, assume the vertices are numbered from 2 to $n+1$, so $V = \{2,3,\ldots n{+}1\}$ and $E \subseteq \{(i,j) \mid 2 \leq i \leq n+1, 2 \leq j \leq n+1\}$. (Note: The n vertices are numbered from 2 to $n+1$ to account for the shifting of backreferences caused by the outer parenthesis of α_0, defined below.) Let $\Sigma = \{a\}$. Let $w = a^{k+|E|} = a^{k+m}$.

Vertex Component: Construct α as follows:

Let $\alpha_0 = (\underset{1}{\ } (\underset{2}{a}\underset{2}{\ }) \mid (\underset{3}{a}\underset{3}{\ }) \mid (\underset{4}{a}\underset{4}{\ }) \mid \ldots \mid (\underset{n}{a}\underset{n}{\ }) \mid (\underset{n+1}{\ } \underset{n+1}{a} \underset{1}{\ }))*$

That is, α_0 is n copies of (a) connected by *or*, and then starred. Note that α_0 can be constructed in $O(|V|)$ time.

Edge Component: Assume the edges are ordered from 1 to m: $e_t \in E$ for $1 \leq t \leq m$. For each $e_t = (i,j) \in E$, let $\alpha_t = (\backslash i \mid \backslash j)$. That is, each α_t represents the t^{th} edge via backreferences, with the backreference incremented by one to account for the outer parenthesis in α_0. Note that this can be done in $O(|E|)$ time.

Finally, let $\alpha = \alpha_0\alpha_1\alpha_2 \ldots \alpha_m$. Note that α can be constructed in $O(|V|+|E|)$ time. We now show that α matches w iff G has a vertex cover of size $\leq k$. Suppose $V' \subseteq V$ is a vertex cover for G with $|V'| \leq k$. Then we can find a valid match tree for (w, α) as follows: We can safely assume that $|V'| = k$, since additional vertices from V can always be added to make this true. Let us start by matching k iterations of α_0, one for each $v \in V'$. For each $v \in V' \subseteq \{2,3,\ldots,n+1\}$ match the v^{th} option of the *or* in α_0 on a different iteration. More specifically, if we let $V' = \{v_1, v_2, \ldots, v_k\} \subseteq \{2, 3, \ldots, n+1\}$ then on the v_i^{th} iteration of α_0 match the v_i^{th} option of the *or* in α_0, which will assign a to $\backslash v_i$. Thus, α_0 matches a^k. Recall that each $\alpha_t = (\backslash i \mid \backslash j)$ for $1 \leq t \leq m$ represents edge $e_t = (i, j)$. Since V' is a vertex cover for G, at least one of $\{i, j\}$ is in V'. Therefore, at least one of $\{\backslash i, \backslash j\}$ is already defined in our match tree. If $\backslash i$ is defined, match it. Otherwise match $\backslash j$. Furthermore, any defined backreference $\backslash 2, \backslash 3, \ldots \backslash n+1$ can only match a single a, so $\alpha_1\alpha_2 \ldots \alpha_m$ matches a^m.

Therefore, $\alpha = \alpha_0\alpha_1\alpha_2 \ldots \alpha_m$ matches $a^k a^m = a^{k+m} = w$.

Conversely, suppose $(w, \alpha) = (a^{k+m}, \alpha)$ is the root of a valid match tree. Then we can find a vertex cover $V' \subseteq V$ for G with $|V'| \leq k$ as follows: To begin, let $V' = \phi$. Since there is a match for α, each of $\alpha_1\alpha_2 \ldots \alpha_m$ must be defined. Thus, each α_t for $1 \leq t \leq m$ matches a single a. For each $\alpha_t = (\backslash i \mid \backslash j)$, if $\backslash i$ is matched, let $V' = V' \cup \{i\}$, else if $\backslash j$ is matched, let $V' = V' \cup \{j\}$. Recall that $\alpha_1\alpha_2 \ldots \alpha_m$ matches a^m. Thus, V' is a vertex cover for G since it contains one vertex from each edge (from each corresponding α_i).

Then α_0 must match the remaining a^k. Therefore, there can be at most k unique backreferences defined, which means there can be at most k distinct vertices in V'. Therefore, $|V'| \leq k$ and G has a vertex cover of size $\leq k$.

Thus α matches w iff G has a vertex cover of size $\leq k$. Furthermore, the reduction can be done in $O(|V| + |E|) = O(|V|^2)$ time. Therefore, MERE is NP-complete. □

Remark: Notice that this proof crucially uses the (semantic) assumption that unassigned backreferences are not set to the empty string. The result can also be proved without using this 'feature'. However, the proof is a bit more complicated and we omit it here.

4 Extended Multi-Pattern Languages (EMPL)

Let Σ be a finite set of terminals $\{a_1, \ldots, a_n\}$ and $V = \{x_1, x_2, \ldots\}$ be an infinite set of variables ($\Sigma \cap V = \emptyset$). Then a *pattern* is a non-null finite string over $\Sigma \cup V$. We use the terms *erasing* (E) and *non-erasing* (NE) pattern languages in the following sense. Let $H_{\Sigma,V}$ be the set of morphisms $h : (\Sigma, V)^* \to (\Sigma, V)^*$. The E pattern language generated by a pattern π is defined as

$$L_E(\pi) = \{w \in \Sigma^* \mid \exists h \in H_{\Sigma,V}((\forall a \in \Sigma : h(a) = a) \wedge w = h(\pi))\}$$

The NE pattern language generated by a pattern π is defined as

$$\{w \in \Sigma^* \mid \exists h \in H_{\Sigma,V}((\forall a \in \Sigma : h(a) = a) \wedge (\neg\exists v \in V : h(v) = \lambda) \wedge w = h(\pi))\}$$

and denoted as $L_{NE}(\pi)$.

Given a set of patterns $\{\pi_1, \pi_2, \ldots, \pi_n\}$, the *E-multi-pattern language* (MPL-E) they define is $\bigcup_{i=1}^{n} L_E(\pi_i)$. Similarly, the *NE-multi-pattern language* (MPL-NE) defined by $\{\pi_1, \pi_2, \ldots, \pi_n\}$ is $\bigcup_{i=1}^{n} L_{NE}(\pi_i)$.

This notion was extended by Nagy [2] to that of EMP expressions in the following way:

Let $\{\pi_1, \pi_2, \ldots, \pi_n\}$ be a set of patterns. Each pattern π_i $(1 \leq i \leq n)$ is an EMP expression. If γ and δ are EMP expressions then

$\gamma \vee \delta$ is also an EMP expression (using the operation union),
$\gamma \cdot \delta$ is also an EMP expression (using the operation concatenation),
γ^* is also an EMP expression (using the operation Kleene star).

In other words, extended multi-pattern (EMP) expressions are obtained from the patterns $\pi_1, \ldots, \pi_n$ by using finitely many regular operators. The EMP expressions which can be obtained *without using union* $(\vee)$ are called *star-pattern* expressions (EMSP expressions).

The *erasing extended multi-pattern language* defined by an EMP expression can be obtained from the E pattern languages in the following way:

$L_E(\gamma \vee \delta) = L_E(\gamma) \cup L_E(\delta)$ (using the operation union),
$L_E(\gamma \cdot \delta) = L_E(\gamma) \cdot L_E(\delta)$ (using the operation concatenation),
$L_E(\gamma^*) = (L_E(\gamma))^*$ (using the operation Kleene star).

We then define EMPL-E (Extended Multi-Pattern Languages – Erasing) to be the family of erasing extended multi-pattern languages [2].

The non-erasing extended multi-pattern language defined by an EMP expression can be obtained from the NE pattern languages in the following way:

$L_{NE}(\gamma \vee \delta) = L_{NE}(\gamma) \cup L_{NE}(\delta)$ (using the operation union),
$L_{NE}(\gamma \cdot \delta) = L_{NE}(\gamma) \cdot L_{NE}(\delta)$ (using the operation concatenation),
$L_{NE}(\gamma^*) = (L_{NE}(\gamma))^*$ (using the operation Kleene star).

Let EMPL-NE (Extended Multi-Pattern Languages – Non-Erasing) stand for the family of non-erasing extended multi-pattern languages.

It is not hard to see that EMPL-E (resp. EMPL-NE) is the *regular closure* [7,8] of the family of E pattern (resp. NE pattern) languages.

If γ is an EMSP expression then $L_E(\gamma)$ is an erasing extended multi-star-pattern language. Let EMSPL-E be the family of erasing extended multi-star-pattern languages. Similarly, if γ is an EMSP expression then $L_{NE}(\gamma)$ is a non-erasing extended multi-star-pattern language. Let EMSPL-NE be the family of non-erasing extended multi-star-pattern languages.

Finally, let EMPL = EMPL-E ∪ EMPL-NE, and likewise, let EMSPL = EMSPL-E ∪ EMSPL-NE. Now two questions arise:

Question 1: Does EMPL-E = EMPL-NE (= EMPL)?

Question 2: Does EMSPL-E = EMSPL-NE (= EMSPL)?

[2] Note that this is not the same as the family PL(REG, REG) defined in [6].

We answer Question 1 affirmatively and Question 2 negatively.

Lemma 6. *The language* $L = \{ xbx \mid x \in \{a,b\}^* \} \in$ *EMSPL-E.*

Proof: Let $\Sigma = \{a,b\}$ and $v \in V$. Then $\alpha = vbv$ is an EMSP expression and $L_E(\alpha) = L$. □

Lemma 7. *The language* $L = \{ xbx \mid x \in \{a,b\}^* \} \notin$ *EMSPL-NE.*

Proof: (by contradiction). Assume L is an NE star-pattern language. Then there is an NE star-pattern expression α such that $L_{NE}(\alpha) = L$. α cannot contain the union operator since it is an NE star-pattern expression.

Clearly α cannot have a star as its outer-most operator. Otherwise, α would match λ, which is not in L. Define a language to be *non-trivial* if and only if it is neither empty nor the singleton set $\{\lambda\}$.

Claim: L is not the concatenation of two non-trivial languages.

Proof: Assume the contrary and let $L = A \circ B$ with A and B non-trivial. Without loss of generality assume that $b \in B$. Hence every non-empty string in A must be of the form $ubbu$ for some $u \in \{a,b\}^*$. Now consider the string aba which belongs to L. aba has to be in B since no non-empty prefix of it can be in A. But then the word equation $ubbuaba =^? xbx$ has no solution. □

Since α cannot be a single pattern either, the result follows[3].

Lemma 8. *EMSPL-E* $\neq$ *EMSPL-NE.*

Proof: The language L of Lemma 6 (and 7) is in EMSPL-E, but is not in EMSPL-NE. □

Lemma 9. *EMPL-NE = EMPL-E.*

Proof: This follows from the results of [3]. We omit the proof. □

Lemma 10. *Every EMPL language is semi-linear.*

Proof-sketch: Every language in MPL is semi-linear [3]. The family of semi-linear languages is closed under union, concatenation and star. □

Lemma 11. *The EREG language* $\mathcal{L}$`((a*)b(\1))` $= \{a^i b a^i \mid i \geq 0\}$ *is not an EMPL.*

Proof-idea: If any of the patterns used in the defining expression contains a variable, then it can be replaced with bb. □

Theorem 4. *EMPL* $\subset$ *EREG.*

[3] Thus without backreferences the analogous result to Theorem 17 of [2] does not hold.

Proof-idea: EREG $\not\subseteq$ EMPL follows from the previous lemma. EMPL $\subseteq$ EREG can be shown as follows: Given an EMP expression γ over terminal alphabet $\Sigma = \{a_1, \ldots, a_n\}$, we can construct an equivalent EREG pattern α. Let $\{\pi_1, \pi_2, \ldots, \pi_n\}$ be the set of patterns that occur within γ. For each pattern π_i $(1 \leq i \leq n)$, replace the first occurrence of a variable within π_i with $((a_1|a_2|\ldots|a_n)*)$, where k is the index of the outer left parenthesis we are adding. Replace each subsequent occurrence of the same variable with $\backslash k$. □

Lemma 12. *The following problem is undecidable:*

Instance: Two sets of patterns P_1 and P_2.
Question: Is $(L_E(P_1))^* \subseteq (L_E(P_2))^*$?

Proof idea: The problem of deciding, given two patterns α and β, whether $L_E(\alpha) \subseteq L_E(\beta)$ is known to be undecidable [9]. Let # be a new symbol, not present in the alphabet Σ of α and β. Let $\Omega = \Sigma \cup \{\#\}$. Now form the sets of patterns

$$\Gamma = \{\#\alpha\#\} \text{ and } \Delta = \{\#\beta\#,\ \#x_1\#x_2\#\}.$$

Claim 1: $(L_E(\Gamma))^* \subseteq (L_E(\Delta))^*$ over Ω if and only if $L_E(\alpha) \subseteq L_E(\beta)$ over Σ.

Claim 2: $(L_E(\Gamma))^* \subseteq (L_E(\Delta))^*$ if and only if $(L_E(\Gamma \cup \Delta))^* = (L_E(\Delta))^*$.

Thus the equivalence problem for EMSPL-E is undecidable. □

The same technique will work for EMSPL-NE, except that Δ will have to be defined a little differently, as $\{\#\beta\#,\ \#x_1\#x_2\#,\ \#\#x_2\#,\ \#x_1\#\#,\ \#\#\#\}$. It can also be shown that

Lemma 13. *For every EMP of the form α^* (i.e., with star as the outermost operator), there is an EMSP γ such that $L_{NE}(\gamma) = L_E(\alpha^*)$.*

Proof-sketch: Since the families EMPL-E and EMPL-NE are the same, there must be an EMP β such that $L_{NE}(\beta) = L_E(\alpha)$. It can be shown (see e.g., [10]) that an expression equivalent to β^* that does not use $\vee$ can be found. □

Theorem 5. *The equivalence problem for EMSPL-NE is undecidable.*

This settles an open problem given in [2].

Acknowledgements. We thank Colin Scheriff for work on an earlier version of the paper [11] and in particular on the proof of Lemma 3. We also thank the referees for their insightful comments.

References

1. Câmpeanu, C., Salomaa, K., Yu, S.: A formal study of practical regular expressions. Int. J. Found. Comput. Sci. 14, 1007–1018 (2003)
2. Nagy, B.: On the language equivalence of NE star-patterns. Inf. Process. Lett. 95, 396–400 (2005)

3. Kari, L., Mateescu, A., Paun, G., Salomaa, A.: Multi-pattern languages. Theor. Comput. Sci. 141, 253–268 (1995)
4. Aho, A.V.: Algorithms for finding patterns in strings. In: Handbook of Theoretical Computer Science. Algorithms and Complexity (A), vol. A, pp. 255–300. MIT Press, Cambridge (1990)
5. Gallier, J.H.: Logic for Computer Science: Foundations of Automatic Theorem Proving. Harper & Row (1986)
6. Dumitrescu, S., Păun, G., Salomaa, A.: Languages associated to finite and infinite patterns. Rev. Roum. Math. Pures Appl. 41, 607–625 (1996)
7. Bertsch, E., Nederhof, M.J.: Regular closure of deterministic languages. SIAM J. Comput. 29, 81–102 (1999)
8. Kutrib, M., Malcher, A.: Finite turns and the regular closure of linear context-free languages. Discrete Applied Mathematics 155, 2152–2164 (2007)
9. Jiang, T., Salomaa, A., Salomaa, K., Yu, S.: Decision problems for patterns. J. Comput. Syst. Sci. 50, 53–63 (1995)
10. Nagy, B.: A normal form for regular expressions. In: Eighth International Conference on Developments in Language Theory (DLT 2004), Auckland, New Zealand (2004), www.cs.auckland.ac.nz/CDMTCS/researchreports/252dlt04.pdf
11. Carle, B., Narendran, P., Scheriff, C.: On extended regular expressions. In: Twenty-first International Workshop on Unification (UNIF 2007), Paris, France (June 2007)

Multi-tilde Operators and Their Glushkov Automata

Pascal Caron, Jean-Marc Champarnaud, and Ludovic Mignot

LITIS, University of Rouen, France
{pascal.caron,jean-marc.champarnaud}@univ-rouen.fr,
ludovic.mignot@etu.univ-rouen.fr

Abstract. Classical algorithms convert arbitrary automata into regular expressions that have an exponential size in the size of the automaton. There exists a well-known family of automata, obtained by the Glushkov construction (of an automaton from an expression) and named Glushkov automata, for which the conversion is linear. Our aim is to extend the family of Glushkov automata. A first step for such an extension is to define a new family of regular operators and to check that the associated extended expressions have good properties: existence of normal forms, succinctness with respect to equivalent simple expressions, and compatibility with Glushkov functions. This paper addresses this first step and investigates the case of multi-tilde operators.

1 Introduction

The framework of this paper is the translation from a regular expression into a finite automaton as well as the inverse translation. Numerous studies have been devoted to this topic, leading to polynomial algorithms for both sides of the conversion (the first ones are due to McNaughton and Yamada [1] for both constructions, to Glushkov [2] for constructing an automaton, and to Brzozowski and McCluskey [3] for constructing an expression).

Many research works have focused on producing a small automaton (as efficiently as possible). For example there exist quadratic algorithms[1] for converting an expression into its Glushkov automaton [6], its Antimirov automaton [7] or its follow automaton [8]. Moreover, the Glushkov automaton of an expression with n occurences of symbols (we say that its alphabetic width is equal to n) has only $n+1$ states; the Antimirov automaton and follow automaton (that are quotients of the Glushkov automaton) have at most $n+1$ states.

On the other hand, classical algorithms compute expressions the size of which is exponential with respect to the number of states of the automaton and studies that address the problem of producing a short expression are not so many. Let us cite the heuristic-based approaches presented in [9] and in [10] that both

[1] The fastest conversion from an expression into an automaton is the $O(n(\log(n))^2)$ algorithm [4] based on the Common Follow Sets introduced by [5]. The number of states of the resulting automaton is a polynomial of n.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 290–301, 2009.

aim at reducing the size of the expression computed by the state elimination algorithm [3]. Recently, several questions were raised in [11] concerning the descriptional complexity of regular expressions and more precisely the effect of regular operations on this complexity. An answer is given in [12], concerning the operation of removing the empty word that is proved to incur at most a quasi-linear increase in regular expressions. More recently, exponential lower bounds have been provided for the intersection and shuffle operations as well as a doubly-exponential lower bound for the complementation [13], [14]. In [15] it is shown that quadratic size expressions can be computed for language quotient operations and cubic size expressions for circular shift operation.

In this paper, we also address the problem of computing short expressions, and we focus on a specific kind of conversion based on Glushkov automata. Actually the particularity of Glushkov automata is the following: any regular expression of width n can be turned into its Glushkov $(n+1)$-state automaton; if a $(n+1)$-state automaton is a Glushkov one, then it can be turned into an expression of width n. The latter property is based on the characterization of the family of Glushkov automata presented in [16]. Our aim is to extend this family according to the following schema. In a first step, we design new families of regular operators supporting Glushkov computation and we check that they lead to extended expressions significantly shorter than equivalent simple expressions. In a second step, we give an algorithm to turn a Glushkov $(n+1)$-state automaton into an extended expression of width n.

Presently, we have completed the study of two new families of operators: multi-bars and multi-tildes. The first step includes a large amount of technical work, in particular for investigating the properties of extended expressions and for determining normal forms. It is the reason why multi-bar presentation in [17] and multi-tilde presentation in this paper only address this first step. Notice that although multi-bar and multi-tilde operators are based on dual elementary operations: eliminating (resp. adding) the empty word from (resp. to) the language of an expression, there exists no obvious way allowing to go from multi-bar properties to multi-tilde ones and respective proofs are quite different. It is the reason why two separate studies have been presented. Let us point out that it is necessary to consider extended expressions mixing operators from both of these families for investigating the second step.

The following section recalls fundamental notions concerning regular expressions and finite automata. In Section 3 we introduce the family of multi-tilde operators and we study their properties in Section 4. In Section 5 we introduce the notion of an expression in tilde normal form and we compute the Glushkov functions of multi-tilde operators in Section 6.

2 Preliminaries

Let us first review basic notions concerning regular expressions and finite automata. For a comprehensive treatment of this domain, references [18], [19] can be consulted.

Let Σ be an alphabet and ε be the empty word. Let E be a regular expression over Σ. The language denoted by the expression E is $L(E)$. The expression E is a *simple regular expression* if the only operators used are union $(+)$, concatenation $(\cdot)$ and Kleene closure $(*)$. The alphabetical width $|E|$ of an expression E is the number of occurences of symbols in E. An expression in which every symbol occurs only once is called a *linear expression*. The expression E is nullable if its language contains the empty word. Let $P = E_1 \cdots E_n$ be a concatenation of n expressions. We will denote by $E_{i,j}$ the factor $E_i \cdots E_j$ of P and we will assume that $L(E_{i,j}) = \{\varepsilon\}$ when $i > j$. Two factors $E_{i,j}$ and $E_{i',j'}$ of P overlap if and only if $1 \leq i \leq i' \leq j \leq j' \leq n$. We will denote by $E_{1:n}$ the list $(E_1, \ldots, E_n)$. Given a set S, we denote by $\mathrm{Card}(S)$ the number of elements of S.

In the following, we will focus on new operators fitting with the Glushkov automaton computation. The main interest of the Glushkov automaton of a linear expression E is to provide a one-to-one mapping from the set of symbols of the expression onto the set of states of the automaton. Glushkov [2] and McNaughton and Yamada [1] have defined independently the Glushkov automaton of an expression E and proved that it recognizes the language $L(E)$. We recall here this definition.

In order to specify their position in the expression, symbols are subscripted following the order of reading. For example, starting from $E = (a + b)a^*$, one obtains the linear expression $\overline{E} = (a_1 + b_2)a_3^*$. The set of positions of the expression is denoted by $\mathrm{Pos}(E)$. The mapping h: $\mathrm{Pos}(E) \to \Sigma$ gives the symbol that is at a given position. In the previous example, we have $\mathrm{h}(2) = b$. Four functions are defined in order to compute an automaton: $\mathrm{First}(E)$ is the set of initial positions of words of $L(\overline{E})$, $\mathrm{Last}(E)$ is the set of final positions of words of $L(\overline{E})$ and $\mathrm{Follow}(E, k)$ is the set of positions immediately following the position k in a word of $L(\overline{E})$. The $\mathrm{Null}(E)$ function is defined as $\{\varepsilon\}$ if E is nullable, $\emptyset$ otherwise. These four functions allow us to define the Glushkov automaton M_E of E as follows: $M_E = (\Sigma, Q, \{0\}, F, \delta)$ where $Q = \mathrm{Pos}(E) \cup \{0\}$ is the set of states and 0 is not in $\mathrm{Pos}(E)$. The set of final states is $F = \mathrm{Last}(E)$ if $\mathrm{Null}(E) = \emptyset$, $F = \mathrm{Last}(E) \cup \{0\}$ otherwise, and the set of transitions is defined by $\delta = \{(x, \mathrm{h}(y), y) \mid x \in \mathrm{Pos}(E) \text{ and } y \in \mathrm{Follow}(E, x)\} \cup \{(0, \mathrm{h}(y), y) \mid y \in \mathrm{First}(E)\}$.

3 The Family of Multi-tilde Operators

Given a list $E_{1:n}$ of regular expressions, we study the family of languages denoted by the expressions obtained from the expression $P = E_{1,n}$ by simultaneously adding the empty word to several (and possibly overlapping) factors $E_{i,f}$ of P. Throughout this paper we will assume that for all $1 \leq k \leq n$ the expression E_k is not equal to the empty word nor to the empty set.

3.1 Multi-tilde Definition

We first define a new operator, called tilde, that can be applied to any regular expression and that is such that $L(\widetilde{E}) = L(E) \cup \{\varepsilon\}$. A multi-tilde operator

applies to a list $E_{1:n}$ of regular expressions and allows us to apply the tilde operator to several factors of the expression $E_{1,n}$. The position of these factors is retrieved from a set of couples $T = ((i_k, f_k))_{1 \leq k \leq \mathrm{Card}(T)}$ such that for all $k \in [\![1, \mathrm{Card}(T)]\!], 1 \leq i_k \leq f_k \leq n$. Such an operator based on a list T is denoted by P_T and a couple in T is called a tilde. By convention, the list T is finite and ordered according to the lexicographic order over $\mathbb{N}^2$ and, if empty, we assume that $P_T(E_{1:n}) = E_{1,n}$. For example, if $T = \big((1,2)(2,3)\big)$, the effect of the operator P_T on the list (E_1, E_2, E_3) is to add the empty word to the factors $E_1 \cdot E_2$ and $E_2 \cdot E_3$.

We first examine a basic case where the language $L(P_T(E_{1:n}))$ is easily defined.

Definition 1. *A finite list of couples S is said to be free if and only if, $\forall (i,f)$, $(i',f') \in S$, we have $[\![i,f]\!] \cap [\![i',f']\!] = \emptyset$.*

In the case where a multi-tilde is defined by a free list S, all the factors defined by the list can simultaneously be made nullable (by adding the empty word to each of them) and thus the language is defined as follows:

Definition 2. *Let $E_{1:n}$ be a list of expressions. Let $S = ((i_k, f_k) \mid \forall k, 1 \leq i_k \leq f_k \leq n)_{1 \leq k \leq s = \mathrm{Card}(S)}$ be a free list of couples. The language $L(P_S(E_{1:n}))$ is defined as follows:*

$$L(P_S(E_{1:n})) = L(E_{1,i_1-1}) \cdot (L(E_{i_1,f_1}) \cup \{\varepsilon\}) \cdot L(E_{f_1+1,i_2-1}) \\ \cdots (L(E_{i_s,f_s}) \cup \{\varepsilon\}) \cdot L(E_{f_s+1,n})$$

The case of a non-free list can be illustrated by the following example.

Example 1. We consider the list $T = \big((1,2)(2,3)(3,4)\big)$ and the expression $E = P_T(a,b,c,d)$, that can be represented by $\widetilde{a\ \ b\ \ c\ \ d}$.

Since the sublist $S = \big((1,2)(3,4)\big)$ is a free one, the factors ab and cd can simultaneously be made nullable by adding the empty word to each of them. On the other hand, it is not possible to simultaneously substitute $ab + \varepsilon$ to the factor ab and $bc + \varepsilon$ to the factor bc. The same reasoning applies to the factors bc and cd. Therefore we consider all the free sublists of T and define the language $L(E)$ as the union of their languages.

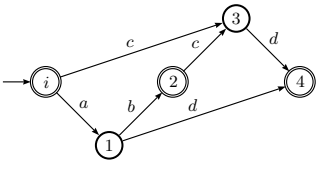


Fig. 1. An automaton recognizing the language $L\big(P_{((1,2)(2,3)(3,4))}(a,b,c,d)\big)$

In our example, we get: $L(E) = L(\widetilde{a \cdot b} \cdot c \cdot d) \cup L(a \cdot \widetilde{b \cdot c} \cdot d) \cup L(a \cdot b \cdot \widetilde{c \cdot d}) \cup L(\widetilde{a \cdot b} \cdot \widetilde{c \cdot d})$ and thus $L(E) = \{abcd, cd, ab, ad, \varepsilon\}$.

Finally, for an arbitrary list T of couples, $L(P_T(E_{1:n}))$ is defined as follows:

Definition 3. *Let $E_{1:n}$ be a list of expressions. Let $T = ((i_k, f_k) \mid \forall k, 1 \leq i_k \leq f_k \leq n)_{1 \leq k \leq \mathrm{Card}(S)}$ be a list of couples. Let $\mathcal{T}$ be the set of all the free sublists of T. Then the language $L(P_T(E_{1:n}))$ is defined as follows:*

$$L\big(P_T(E_{1:n})\big) = \bigcup_{T' \in \mathcal{T}} L(P_{T'}(E_{1:n}))$$

This new family of multi-tilde operators leads us to consider the following extension of regular expressions.

Definition 4. *An extended to multi-tilde regular expression (**EMTRE**) is inductively defined by:*

$$\begin{array}{ll} \mathbf{E} = \emptyset & \mathbf{E} = \mathbf{F} + \mathbf{G} \text{ where } F \text{ and } G \text{ are two EMTREs} \\ \mathbf{E} = \varepsilon & \mathbf{E} = \mathbf{F} \cdot \mathbf{G} \text{ where } F \text{ and } G \text{ are two EMTREs} \\ \mathbf{E} = \mathbf{a} \text{ with } a \in \Sigma & \mathbf{E} = \mathbf{F}^* \text{ where } F \text{ is an EMTRE} \end{array}$$
$$\mathbf{E} = \mathbf{P_T(E_{1:n})} \text{ where } E_{1:n} \text{ is a list of (not equal to } \emptyset \text{ nor } \varepsilon) \text{ EMTREs}$$

3.2 Reduction Power of Multi-tildes

Multi-tildes, similarly to multi-bars [17] lead to a reduction of the alphabetic width that is generally more important than by a straightforward factorization of an equivalent simple regular expression. This fact is illustrated by Example 2.

Example 2. Let $E = \widetilde{a \quad b \quad c}$ be an EMTRE, with $|E| = 3$. The language $L(E)$ is recognized by the automaton of Figure 2. Any equivalent simple regular expression is wider than E; for example, the width of the expression $E' = a(bc + \varepsilon) + c$ is equal to 4.

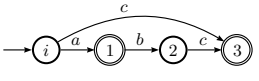


Fig. 2. An automaton recognizing the language $L\big(P_{((1,2)(2,3))}(a,b,c)\big)$

Example 2 can be generalized by considering the language L_n denoted by the EMTRE $E_n = P_{T_n}(a_1, \ldots, a_n)$, with $T_n = \big((1,2)(2,3)\ldots(n-1,n)\big)$. The importance of the multi-tilde operators can be shown by comparing the width of the generalized expression E_n (that is equal to n) and the minimal width of an equivalent simple expression E'_n. It is well known that there exists no polynomial algorithm for computing a regular expression of minimal width denoting a given regular language [20], even if this language is a finite one. Therefore we proceed as follows: we start from $E'_3 = a_1(a_2a_3 + \varepsilon) + a_3$ (that can be easily shown to be an expression of minimal width), and we apply a specific procedure to construct E'_n from E'_{n-1}. Although it is not proved that the alphabetic width of E'_n is minimal, this expression seems to be a good candidate for minimality. Moreover, it can be shown that the expression E'_n has an exponential width with respect to n. Let us remark that the alphabet of the languages $(L_n)_{n \geq 3}$ does not have a constant size.

4 Properties of Multi-tildes

In this section we first introduce some notation and then state properties of multi-tilde operators that will be useful in the next sections.

4.1 Definitions and Notation

Let $E = P_T(E_{1:n})$ be an EMTRE. Let $t = (i, f) \in T$; we set $\text{Ranks}(t) = [\![i, f]\!]$. Let $k \in [\![i, f]\!]$; we set $\text{Tildes}(k) = \{(i, f) \in T \mid i \leq k \leq f\}$. A tilde $(i, f) \in T$ is *overlapping* if and only if there exists $(i', f') \in T$ such that $i' < i \leq f' < f$ or $i < i' \leq f < f'$. A tilde $t = (i, f) \in T$ is *included* if and only if there exists a tilde $(i', f') \in T \setminus \{t\}$ such that $i' \leq i \leq f \leq f'$. A tilde in T is *overhanging* if and only if it is not overlapping. We say that T is *overlapping* if and only if every tilde of T is overlapping. We say that T is *continuous* if and only if for all $k \in [\![1, n-1]\!]$, $\text{Tildes}(k) \cap \text{Tildes}(k+1) \neq \emptyset$. We say that T has a *minimal arity* if and only if $\forall k \in [\![1, n-1]\!]$, $\text{Tildes}(k) \neq \text{Tildes}(k+1)$.

Definition 5. *Let $i, f \in [\![1, n]\!] \mid i \leq f$. The linear EMTRE $E = P_T(E_{1:n})$ is said to be (i, f)-nullable if and only if $L(E_{1,i-1}) \cdot L(E_{f+1,n}) \subset L(E)$. Let S be a list of couples. The expression E is said to be S-nullable if and only if it is (i, f)-nullable for all $(i, f) \in S$.*

Definition 6. *A tilde $t = (i, f) \in T$ is useful for a linear EMTRE $E = P_T(E_{1:n})$ if and only if the languages $L(P_T(E_{1:n}))$ and $L(P_{T\setminus\{t\}}(E_{1:n}))$ are different. The list T is useful if and only if every tilde of T is useful.*

Let us consider a linear EMTRE $E = P_T(E_{1:n})$. Let $w = w_1 \cdots w_n$ be a word such that for all $k \in [\![1, n]\!], w_k \in L(E_k) \cup \{\varepsilon\}$. Let us suppose that there exists a factor $w_i \cdots w_f = \varepsilon$. This factor has a *left* (resp. *right*) *ε-extension* in w if there exists $k = i - 1$ (resp. $k = f + 1$) such that $w_k = \varepsilon$. If a factor of w has no ε-extension, then, it is *ε-maximal in w*.

4.2 Nullability Properties

Lemma 1. *Let $E = P_T(E_{1:n})$ be a linear EMTRE and $w \in L(E)$. Then there exists a unique decomposition $w = w_1 \cdots w_n$ such that $\forall k \in [\![1, n]\!]$, $w_k \in L(E_k) \cup \{\varepsilon\}$.*

Lemma 2. *Let $E = P_T(E_{1:n})$ be a linear EMTRE. Let $i, f \in [\![1, n]\!] \mid i \leq f$. The expression E is (i, f)-nullable if and only if there exists a free list $S = \big((i_k, f_k) \mid k \in [\![1, \text{Card}(S)]\!]\big)$ of couples satisfying the following conditions:*

- *$i_1 = i$, $f_{\text{Card}(S)} = f$ and $\forall k \in [\![1, \text{Card}(S) - 1]\!]$, $f_k = i_{k+1} - 1$,*
- *$\forall (i_k, f_k) \in S$, we have: $(i_k, f_k) \in T$ or E_{i_k, f_k} is nullable.*

Proof. We set $L' = L(E_{1,i-1}) \cdot L(E_{f+1,n})$.

(1) $\Leftarrow$ (2) : let S be a list satisfying the previous conditions. We consider the partition $(S_1 = S \cap T, S_2 = S \setminus S_1)$ of S. Since S is free, S_1 is a free sublist of

T. By Definition 3 it holds that $L(P_{S_1}(E_{1:n})) \subset L(E)$. Moreover since for every couple (i_k, f_k) in S_2 the expression E_{i_k,f_k} is nullable, we have $L(P_{S_1 \cup S_2}(E_{1:n})) = L(P_{S_1}(E_{1:n}))$. Finally, $L' \subset L(P_S(E_{1:n}))$ and the expression E is (i,f)-nullable.

(1) ⇒ (2) : we suppose that E is (i,f)-nullable. By Definition 5, $L' = L(E_{1,i-1}) \cdot L(E_{f+1,n}) \subset L(E)$. Let $w = w_1 \cdots w_{i-1} \cdot w_{f+1} \cdots w_n$ be a word in L' such that for all $k \in [\![1, i-1]\!] \cup [\![f+1, n]\!]$, $w_k \in L(E_k) \setminus \{\varepsilon\}$. Since $L' \subset L(E)$, there exists a free subslist S of T such that $w \in L(P_S(E_{1:n}))$. Let (i_1, f_1) (resp. (i_s, f_s)) be the least (resp. the greatest) element in the ordered list S and $(i_{k+1}, i_{f_{k+1}})$ be the least element greater than (i_k, f_k). Let us consider the set V of couples defined by: $V = \{(i, i_1 - 1), (f_s + 1, n)\} \cup \{(f_k + 1, i'_{k+1} - 1) \mid (i_k, f_k), (i_{k+1}, f_{k+1}) \in T\}$. If $S = \emptyset$ we set $V = V \cup \{(i, f)\}$. There exists no couple (i', f') in V satisfying $i \leq i' \leq f' \leq f$ and $\varepsilon \notin L(E_{i',f'})$. Otherwise, by Definition 3 the word w would not be in $L(P_S(E_{1:n}))$. Therefore there exist only two cases: either $E_{i',f'}$ is nullable for any couple in V, or for all $k \in [\![1, \mathrm{Card}(S) - 1]\!]$, $f_k = i_{k+1}$, $i_1 = i$ and $f_{\mathrm{Card}(S)} = f$. In these two cases, we can construct a list of couples satisfying the conditions of the lemma. □

Proposition 1. *Let $E = P_T(E_{1:n})$ be a linear EMTRE. Let $w = w_1 \cdots w_n$ be a word such that for all $j \in [\![1, n]\!]$, $w_j \in L(E_j) \cup \{\varepsilon\}$. Let $S = \big((i_k, f_k) \mid k \in [\![1, \mathrm{Card}(S)]\!]\big)$ be a free list such that $\forall k \in [\![1, \mathrm{Card}(S)]\!]$, $w_{i_k} \cdots w_{f_k}$ is an ε-maximal factor in w. Then the three following conditions are equivalent:*

(1) *$L(P_S(E_{1:n})) \subset L(E)$,*
(2) *E is S-nullable,*
(3) *$w \in L(E)$.*

Proof. **(1) ⇒ (2) :** according to Definition 2, $\forall (i,f) \in S$, $L(P_{((i,f))}(E_{1:n})) \subset L(P_S(E_{1:n})) \subset L(E)$. Thus, for each (i,f) of S, E is (i,f)-nullable and then by definition E is S-nullable.

(2) ⇒ (3) : according to Lemma 2, for each couple (i_k, f_k) of S, there exists a list S_k and a partition (S_{1k}, S_{2k}) of S_k such that $S_{1k} = S_k \cap T$ and $S_{2k} = S_k \setminus S_{1k}$. Since S is free, the list $S' = \bigcup_{k \in [\![1, \mathrm{Card}(S)]\!]} S_{1k}$ is free too and $S' \subset T$. Following Definition 3, $L(P_{S'}(E_{1:n})) \subset L(E)$. Moreover, since for every couple (i_k, f_k) of $S'' = \bigcup_{k \in [\![1, \mathrm{Card}(S)]\!]} S_{2k}$, the factor E_{i_k,f_k} is nullable and since $\forall (i_{k'}, f_{k'}) \in S'$, $[\![i_{k'}, f_{k'}]\!] \cap [\![i_k, f_k]\!] = \emptyset$, we have $L(P_{S' \cup S''}(E_{1:n})) = L(P_{S'}(E_{1:n}))$. Finally we get $L(P_S(E_{1:n})) \subset L(E)$ and then $w \in L(E)$.

(3) ⇒ (1) : we suppose that $w \in L(E)$. Since for all $k \in [\![1, \mathrm{Card}(S)]\!]$, $w_{i_k} \cdots w_{f_k}$ is an ε-maximal factor in w, we have $w \in L(P_S(E_{1:n}))$. By Definition 2, there exists a free sublist S' of T such that $w \in L(P_{S'}(E_{1:n}))$. Let $w' = w'_1 \cdots w'_n$ be a word in $L(P_S(E_{1:n}))$ such that $w' \notin L(P_{S'}(E_{1:n}))$. It implies that there exists an ε-maximal $w'_i \cdots w'_f$ in w', such that $P_{S'}(E_{1:n})$ is not (i,f)-nullable. By hypothesis, there exists $k \in [\![1, \mathrm{Card}(S)]\!]\big)$ such that $i_k \leq i \leq f \leq f_k$ and thus $P_{S'}(E_{1:n})$ is not (i_k, f_k)-nullable. Contradiction with $w \in L(P_{S'}(E_{1:n}))$. Finally, $L(P_S(E_{1:n})) \subset L(P_{S'}(E_{1:n})) \subset L(E)$. □

Corollary 1. *Let $E = P_T(E_{1:n})$ be a linear EMTRE. Let $i, f \in [\![1, n]\!]$ with $f \geq i$. Let $w = w_1 \cdots w_{i-1} \cdot w_{f+1} \cdots w_n$ with for all $k \in [\![1, i-1]\!] \cup [\![f+1, n]\!]$, $w_k \in L(E_k) \setminus \{\varepsilon\}$. The following two conditions are equivalent:*

(1) *$w \in L(E)$,*
(2) *E is (i, f)-nullable.*

The following lemma will be useful to construct a normal form for E.

Lemma 3. *Let $E = P_T(E_{1:n})$ be a linear EMTRE. A tilde $t = (i, f) \in T$ is useful for E if and only if $P_{T\setminus\{t\}}(E_{1:n})$ is not (i, f)-nullable.*

5 Definition of Multi-tildes in Tilde Normal Form

Two multi-tilde operators operating on a same list of expressions with distinct lists of couples can recognize the same language. This fact can be illustrated by the following example.

Example 3. Let $E_1 = \overbrace{a\ \underbrace{b}\ \underbrace{c}\ d}$ and $E_2 = \overbrace{a\ \underbrace{b}}\ \overbrace{\underbrace{c}\ d}$, that are respectively based on the set of tildes:

$$\{(1,2)(2,2)(2,3)(3,3)(3,4)\} \text{ and } \{(1,2)(2,2)(3,3)(3,4)\}.$$

It can be checked that the languages $L(E_1)$ and $L(E_2)$ are equal.

We define a set of necessary and sufficient conditions that allow us to define a normal form for multi-tilde operators.

Definition 7. *Let $E = P_T(E_{1:n})$ be a linear EMTRE. The expression E is said to be in tilde normal form (TNF) if and only if the following properties are checked:*

- *T is useful for E,*
- *either T is overlapping or* $\mathrm{Card}(T) = 1$*,*
- *T is continuous,*
- *the arity of T is minimal.*

An expression satisfying these conditions is denoted by $E = \overbrace{P_T}(E_{1:n})$.

Example 4. The syntax tree of the expression $E_3 = \overbrace{(a \cdot \underbrace{b}\,)} \cdot \overbrace{(\underbrace{c} \cdot d)}$ is represented by Figure 3.

It can be checked that this expression satisfies each condition of Definition 7 and that it is equivalent to the expressions E_1 and E_2 of Example 3. Let us remark that the graphical representations of E_2 and E_3 are very close, although the syntax trees are different.

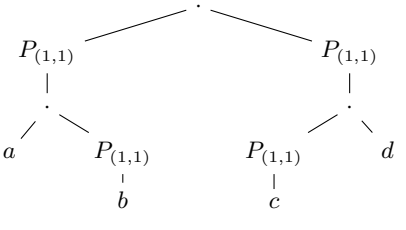


Fig. 3. The expression E_3 is in TNF

An EMTRE is said to be in TNF if and only if its linearized version $\overline{E}$ is in TNF.

Definition 8. *An* extended to normal tilde regular expression*(**ETRE**) is inductively defined as follows:*

$$\begin{array}{ll} \mathbf{E} = \emptyset & \mathbf{E} = \mathbf{F} + \mathbf{G} \text{ with } F \text{ and } G \text{ two ETREs} \\ \mathbf{E} = \varepsilon & \mathbf{E} = \mathbf{F} \cdot \mathbf{G} \text{ with } F \text{ and } G \text{ two ETREs} \\ \mathbf{E} = \mathbf{a} \text{ with } a \in \Sigma & \mathbf{E} = \mathbf{F}^* \text{ with } F \text{ an ETRE} \end{array}$$

$$\mathbf{E} = \widetilde{P_T}\,(\mathbf{E_{1:n}}) \text{ where } E_{1:n} \text{ is a list of (not equal to } \emptyset \text{ nor } \varepsilon) \text{ ETREs}$$

We now prove that the set of conditions of Definition 7 allow us to define a normal form.

Lemma 4. *Let* $E = \widetilde{P_T}\,(E_{1:n})$ *be an ETRE. Then* $L(E) \neq \emptyset$ *and* $L(E) \neq \{\varepsilon\}$.

Proposition 2. *Let* $E_1 = \widetilde{P_{T_1}}\,(E_{1:n})$ *and* $E_2 = \widetilde{P_{T_2}}\,(E_{1:n})$ *be two linear ETREs. The two following conditions are equivalent:*

(1) $L(E_1) = L(E_2)$,
(2) $T_1 = T_2$.

Proof

(1) ⇒ (2) : Let $t = (i, f) \in T_1 \triangle T_2$ such that the length $l_t = f - i + 1$ of t be minimal. Let us suppose that $t \in T_1$. Then, E_1 is (i, f)-nullable. Let us suppose that E_2 is (i, f)-nullable. If $t \in T_2$, we have a contradiction with $t \in T_1 \triangle T_2$. If $\varepsilon \in L(E_{i,f})$, we have a contradiction with the utility of t for E_1 (see Lemma 3). If there exist several couples allowing E_2 to be (i, f)-nullable, and if there exists no tilde $(i', f') \in T_2$ such that $i \leq i' \leq f' < f$ or $i < i' \leq f' \leq f$, the situation is similar as in the previous case. If there exists a tilde, then we can choose it not to be in T_1 (if there does not exist such a tilde, then the combination of tildes and of nullable factors of E_2 make the tilde t useless for E_1). Contradiction with the minimality of l_t. And then, as the expression E_1 is (i, f)-nullable while the expression E_2 is not, their languages are disjoint.

(1) ⇐ (2) : Trivial. □

Proposition 3. *Let* $E = P_T(E_{1:n})$ *be an EMTRE. Then, there exists an ETRE* E' *such that* $L(\overline{E}) = L(\overline{E'})$.

6 Glushkov Functions

Definition 9. *Let* $E = \widetilde{P_T}\,(E_{1:n})$ *be an ETRE with* $\mathrm{Card}(T) = m$ *and* $T = \big((i_k, f_k)\big)_{k \in [\![1,m]\!]}$. *The Glushkov functions associated to the* $\widetilde{P_T}$ *operator are defined as follows:*

(9.1) $\mathrm{Pos}(E) = \bigcup_{k=1}^{n} \mathrm{Pos}(E_k)$

(9.2) $\mathrm{Null}(E) = \begin{cases} \varepsilon & \text{if } E \text{ is } (1,n)\text{-nullable} \\ \emptyset & \text{otherwise} \end{cases}$

(9.3) $\mathrm{First}(E) = \mathrm{First}(E_1) \cup \bigcup_{k \text{ s.t. } E \text{ is } (1,k-1)\text{-nullable}} \mathrm{First}(E_k)$

(9.4) $\mathrm{Last}(E) = \mathrm{Last}(E_n) \cup \bigcup_{k \text{ s.t. } E \text{ is } (k+1,n)\text{-nullable}} \mathrm{Last}(E_k)$

(9.5) $\forall x \in \mathrm{Pos}(E),\ \exists k \in [\![1,n]\!] \mid x \in \mathrm{Pos}(E_k)$. *Then:*

$$\mathrm{Follow}(x,E) = \begin{cases} \mathrm{Follow}(x,E_k) & \text{if } x \notin \mathrm{Last}(E_k) \\ & \text{or if } x \in \mathrm{Pos}(E_n) \\ \mathrm{Follow}(x,E_k) \cup \mathrm{Follow}(x,E_{k+1}) \cup & \\ \quad \bigcup_{k' \text{ s.t. } E \text{ is } (k+1,k'-1)\text{-nullable}} \mathrm{First}(E_{k'}) & \text{otherwise} \end{cases}$$

Proposition 4. *Let* $E = \widetilde{P_T}(E_{1:n})$ *be an ETRE and let* G_E *be its Glushkov automaton. Then, we have* $L(E) = L(G_E)$.

Proof. The definition of Glushkov functions for $+$, $\cdot$ and * operators induces that:

(P1) $a_x \in \Sigma_E \Leftrightarrow x \in \mathrm{Pos}(E)$
(P2) $\mathrm{Null}(E) = \begin{cases} \{\varepsilon\} & \text{if and only if } \varepsilon \in L(E) \\ \emptyset & \text{otherwise} \end{cases}$
(P3) $\exists w = a_x \cdot w' \in L(E) \Leftrightarrow x \in \mathrm{First}(E)$
(P4) $\exists w = w' \cdot a_x \in L(E) \Leftrightarrow x \in \mathrm{Last}(E)$
(P5) $\exists w = w' \cdots a_x \cdot a_{x'} \cdots w'' \in L(E) \Leftrightarrow x' \in \mathrm{Follow}(E,x)$.

Let us prove that Propositions **(P1)** to **(P5)** are still true when extending Glushkov functions to the $\widetilde{P_T}$ operator.

(P1) $\mathrm{Pos}(E)$**:** $a_x \in \Sigma_E \Leftrightarrow \exists k \in [\![1,n]\!] \mid a_x \in \Sigma_{E_k} \Leftrightarrow x \in \mathrm{Pos}(E)$.

(P2) $\mathrm{Null}(E)$**:** According to Corollary 1, we have: $\varepsilon \in L(E) \Leftrightarrow E$ is $(1,n)$-nullable. Thus, the extension of the $\mathrm{Null}(E)$ function is correct.

(P3) $\mathrm{First}(E)$**:**
($\Rightarrow$) According to Lemma 4, we have $L(E) \neq \emptyset$ and $L(E) \neq \varepsilon$. Let $w = a_x \cdot w' \in L(E)$ with $a_x \in \Sigma_E$. Let k be such that $a_x \in \Sigma_{E_k}$, which means that $x \in \mathrm{Pos}(E_k)$. Then, there exists $a_x \cdot w'_k \in L(E_k)$. Thus, we have $x \in \mathrm{First}(E_k)$. If $k = 1$, following Definition 9.3, $x \in \mathrm{First}(E)$. Otherwise, it implies that $w = w_1 \cdots w_{k-1} \cdot w_k \cdot w_{k+1} \cdots w_n$ with $w_k = a_x \cdot w'_k$ and $w_1 \cdots w_{k-1} = \varepsilon$, then $w_1 \cdots w_k \neq \varepsilon$. According to Proposition 1 and as $w_1 \cdots w_{k-1}$ is ε-maximal, there exists a free list S which contains the couple $(1, k-1)$ such that E is S-nullable. Thus, E is $(1, k-1)$-nullable and following Definition 9.3, $x \in \mathrm{First}(E)$.

($\Leftarrow$) Suppose that $x \in \mathrm{First}(E)$ and let $k \in [\![1,n]\!]$ such that $x \in \mathrm{First}(E_k)$. We know that $\forall i \in [\![1,n]\!]$, $L(E_i) \neq \emptyset$ and $L(E_i) \neq \{\varepsilon\}$ according to Definition 8. Suppose that $w = w_k \cdots w_n$ with $\forall i \in [\![k,n]\!], w_i \in L(E_i)$ and $w_i \neq \varepsilon$. If $k = 1$, $w \in L(E_1 \cdots E_n)$ and following Definition 3, $L(E_1 \cdots E_n) \subset L(E)$ and then

$w \in L(E)$. If $k \neq 1$, then, according to Definition 9.3, E is $(1, k-1)$-nullable, and according to Corollary 1, $w \in L(E)$.

(P4) Last(E)**:** The proof is similar to (P3) reasoning on last symbols of a word.

(P5) Follow(E, x)**:**

($\Rightarrow$) Let $x' \in \text{Follow}(E, x)$. Then $\exists k' \in [\![1, n]\!] \mid x' \in \text{Pos}(E'_k)$. **(1)** Suppose that $k = k'$. Then $\exists w_k = w'_k \cdot a_x \cdot a_{x'} \cdot w''_k \in L(E_k)$. If $w = w_1 \cdots w_k \cdots w_n$ with $\forall i \in [\![1, n]\!] \backslash \{k\}, w_i \in L(E_i) \backslash \{\varepsilon\}$ (this decomposition exists according to Definition 8), we have following Definition 3, $w \in L(E_{1,n}) \subset L(E)$, and then $w \in L(E)$. **(2)** Let us suppose $k \neq k'$. According to Definition 9.5, $x \in \text{Last}(E_k)$, $k \neq n$ and $x' \in \text{First}(E_{k'})$. Thus, $\exists w_k = w'_k \cdot a_x \in L(E_k)$ and $\exists w_{k'} = a_{x'} \cdot w''_{k'} \in L(E_{k'})$. Let $w = w_1 \cdots w_k \cdot w'_k \cdots w_n$ with $\forall i \in [\![1, n]\!] \setminus [\![k, k']\!], w_i \in L(E_i) \setminus \{\varepsilon\}$ (this decomposition exists according to Definition 8). **(a)** Suppose that $k' = k + 1$. According to Definition 3, $w \in L(E_1 \cdots E_k \cdot E_{k+1} \cdots E_n) \subset L(E)$, and $w \in L(E)$. **(b)** Suppose that $k' > k + 1$. If $x' \in \text{Follow}(E, x)$, then, we know, according to Definition 9.5 that E is $(k+1, k'-1)$-nullable, and then according to Corollary 1, $w \in L(E)$.

($\Leftarrow$) Suppose that $\exists w = w' \cdot a_x \cdot a_{x'} \cdot w'' \in L(E)$. There exists $k' \mid n > k' \geq k > 1$ such that $x' \in \text{Pos}(E_{k'})$. **(1)** Suppose that $k = k'$. According to Lemma 1, w is equal to $w_1 \cdots w_k \cdots w_n$ with $\forall i \in [\![1, n]\!]$, $w_i \in L(E_i)$ and $w_k = w'_k \cdot a_x \cdot a_{x'} \cdot w''_k$. Thus, $x' \in \text{Follow}(E_k, x)$ and then $x' \in \text{Follow}(E, x)$. **(2)** Suppose that $k \neq k'$. According to Lemma 1, $w = w_1 \cdots w_k \cdot w_{k'} \cdots w_n$ with $\forall i \in [\![1, n]\!]$, $w_i \in L(E_i)$, $w_k = w'_k \cdot a_x$ and $w_{k'} = a_{x'} \cdot w''_{k'}$. **(a)** If $k' = k + 1$, as $x' \in \text{First}(E_{k+1})$, we have $x' \in \text{Follow}(E, x)$. **(b)** If $k' > k + 1$, according to Proposition 1, there exists a free list $S = \big((i, f)_j\big)_{j \in [\![1, \text{Card}(S)]\!]}$ such that for all $(i, f) \in S$, the factor $w_i \cdots w_f$ is ε-maximal, and E is S-nullable. As $w_{k+1} \cdots w_{k'-1}$ is ε-maximal, $(k+1, k'-1) \in S$ and then E is $(k+1, k'-1)$-nullable. Thus, $x' \in \text{Follow}(E, x)$ following Definition 9.5.

7 Conclusion

Adding multi-tilde operators to the set of regular operators allow us to provide shorter expressions for denoting a given regular language. Given an extended expression, it is always possible to compute an equivalent expression in tilde normal form. Such expressions are compatible with Glushkov functions, and thus, any extended expression can still be turned into an equivalent Glushkov automaton. The same properties have been proved for multi-bar operators [17]. In order to achieve the second step of our programme we have considered expressions that are augmented by both multi-bar and multi-tilde operators and we have made use of the properties of the associated tilde normal form to recover an extended expression from a given Glushkov automaton. This result should be communicated shortly.

References

1. McNaughton, R.F., Yamada, H.: Regular expressions and state graphs for automata. IEEE Transactions on Electronic Computers 9, 39–57 (1960)
2. Glushkov, V.M.: On a synthesis algorithm for abstract automata. Ukr. Matem. Zhurnal 12(2), 147–156 (1960) (in Russian)
3. Brzozowski, J.A., McCluskey, E.J.: Signal flow graph techniques for sequential circuit state diagrams. IEEE Trans. on Electronic Computers EC-12(2) (1963)
4. Hagenah, C., Muscholl, A.: Computing epsilon-free nfa from regular expressions in $O(n \log^2(n))$ time. ITA 34(4), 257–278 (2000)
5. Hromkovic, J., Seibert, S., Wilke, T.: Translating regular expressions into small ε-free nondeterministic finite automata. J. Comput. Syst. Sci. 62(4), 565–588 (2001)
6. Brüggemann-Klein, A.: Regular expressions into finite automata. Theoret. Comput. Sci. 120(2), 197–213 (1993)
7. Champarnaud, J.M., Ziadi, D.: From c-continuations to new quadratic algorithms for automata synthesis. Internat. J. Algebra Comput. 11(6), 707–735 (2001)
8. Ilie, L., Yu, S.: Follow automata. Inf. Comput. 186(1), 140–162 (2003)
9. Delgado, M., Morais, J.: Approximation to the smallest regular expression for a given regular language. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 312–314. Springer, Heidelberg (2005)
10. Han, Y.S., Wood, D.: Obtaining shorter regular expressions from finite-state automata. Theor. Comput. Sci. 370(1-3), 110–120 (2007)
11. 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)
12. Ziadi, D.: Regular expression for a language without empty word. Theor. Comput. Sci. 163(1&2), 309–315 (1996)
13. 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)
14. Gruber, H., Holzer, M.: Finite automata, digraph connectivity, and regular expression size. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 39–50. Springer, Heidelberg (2008)
15. Gruber, H., Holzer, M.: Language operations with regular expressions of polynomial size. In: Câmpeanu, C. (ed.) 10th International Workshop on Descriptional Complexity of Formal Systems (DCFS 2008), Charlottetown, Canada, pp. 182–193 (2008)
16. Caron, P., Ziadi, D.: Characterization of Glushkov automata. Theoret. Comput. Sci. 233(1–2), 75–90 (2000)
17. Caron, P., Champarnaud, J.M., Mignot, L.: A new family of regular operators fitting with the position automaton computation. In: et al. (eds.) International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2009), Špindleruv Mýn, Czech Republic. LNCS (2009)
18. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)
19. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Word, Language, Grammar, vol. I, pp. 41–110. Springer, Berlin (1997)
20. Conway, J.H.: Regular algebra and finite machines. Chapman and Hall, Boca Raton (1971)

Non-uniform Cellular Automata*

Gianpiero Cattaneo[1], Alberto Dennunzio[1], Enrico Formenti[2,**], and Julien Provillard[3]

[1] Università degli Studi di Milano–Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione,
Viale Sarca 336, 20126 Milano, Italy
{cattang,dennunzio}@disco.unimib.it
[2] Université de Nice-Sophia Antipolis, Laboratoire I3S,
2000 Route des Colles, 06903 Sophia Antipolis, France
enrico.formenti@unice.fr
[3] Ecole Normale Supérieure de Lyon,
46, Allée d'Italie, 69364 Lyon, France
julien.provillard@ens-lyon.fr

Abstract. In this paper we begin the study the dynamical behavior of non-uniform cellular automata and compare it to the behavior of "classical" cellular automata. In particular we focus on surjectivity and equicontinuity.

1 Introduction and Motivations

Cellular automata (CA) are a well-known formal model for complex systems that is used in many scientific fields [1,2,3]. Uniformity is one of the main characteristics of this model. Indeed, a cellular automaton is made of identical finite automata arranged on a regular lattice. The state of each automaton is updated by a local rule on the basis of the state of the automaton itself and of the one of a fixed set of neighbors. At each time-step, the same (here comes uniformity) local rule is applied to all finite automata in the lattice. For recent results on CA dynamics and an up-to-date bibliography see for instance [4,5,6,7,8,9,10,11,12,13,14,15,16].

In this paper we study a more general setting relaxing the uniformity constraint. Assume to use CA for simulating a physical or natural phenomenon. Relaxing the uniformity constraint can be justified in several situations:

Generality. In many phenomena, each individual locally interacts with others but maybe these interactions depend on the individual itself or on its position in the space.

* This work has been supported by the Interlink/MIUR project "Cellular Automata: Topological Properties, Chaos and Associated Formal Languages", by the ANR Blanc "Projet Sycomore" and by the PRIN/MIUR project "Formal Languages and Automata: Mathematical and Applicative Aspects".

** Corresponding author.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 302–313, 2009.

Structural stability. Assume that we are investigating the robustness of a system *w.r.t.* some specific property P. If some individuals change their "standard" behavior does the system still have property P? What is the "largest" number of individuals that can change their default behavior so that the system does not change its overall evolution?

Reliability. CA are more and more used to perform (fast parallel) computations (see for example [2]). Each cell of the CA is implemented by a simple electronic device (FPGAs for example). Then, how reliable are computations w.r.t. failure of some of these devices? (Here failure is interpreted as a device which behaves differently from its "default" way).

Finally, remark that the study of these generalizations has an interest in its own since the new model coincides with the set of continuous functions in Cantor topology.

2 Non-uniform Cellular Automata ν-$\mathcal{CA}$

In the present paper we focus on the one-dimensional case (i.e. the lattice is $\mathbb{Z}$). Remark that most of the following definitions can be easily extended to higher dimensions. Before introducing the formal definition of ν-$\mathcal{CA}$, one should recall the definition of cellular automaton.

Let A be a finite set describing the possible states of any cell. A *configuration* is a snapshot of the states of all cells in the CA, i.e., a function from $\mathbb{Z}$ to A. Given a configuration x, we denote by $x_i \in A$ the state of the cell in position $i \in \mathbb{Z}$. For a fixed state $a \in A$, a configuration is *a-finite* if only a finite number of automata in the CA are in a state different from a.

A (one-dimensional) *CA* is a structure $\langle A, r, f\rangle$, where A is the above introduced finite set of *states*, also called the *alphabet*, and $f : A^{2r+1} \to A$ is the *local rule* whose *radius* is r. The *global rule* induced by a CA $\langle A, r, f\rangle$ is the map $F : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ defined by

$$\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z},\ F(x)_i = f(x_{i-r}, \ldots, x_{i+r})\ . \tag{1}$$

This rule describes the global evolution of the CA from the generic configuration $x \in A^{\mathbb{Z}}$ at time-step $t \in \mathbb{N}$ to the configuration $F(x)$ at the next time-step $t+1$.

The configuration set $A^{\mathbb{Z}}$ is equipped with the distance $d(x, y) = 2^{-n}$ where $n = \min\{i \geq 0 : x_i \neq y_i \text{ or } x_{-i} \neq y_{-i}\}$. With respect to the topology induced by d, the configuration set is a Cantor space and F is continuous. Hence, $(A^{\mathbb{Z}}, F)$ is a discrete (time) dynamical system.

Notation. Given a configuration $x \in A^{\mathbb{Z}}$, for any pair $i, j \in \mathbb{Z}$, with $i \leq j$, we denote by $x_{[i,j]}$ the word $x_i \cdots x_j \in A^{j-i+1}$, i.e., the portion of the configuration x inside the interval $[i, j] = \{k \in \mathbb{Z} : i \leq k \leq j\}$. A *cylinder* of block $u \in A^k$ and position $i \in \mathbb{Z}$ is the set $[u]_i = \{x \in A^{\mathbb{Z}} : x_{[i,i+k-1]} = u\}$. Cylinders are clopen sets *w.r.t.* the metric d.

The meaning of (1) is that the same local rule f is applied to each site of the CA. Relaxing this last constraint gives us the definition of a ν-$\mathcal{CA}$. More formally one can give the following.

Definition 1 (Non-Uniform Cellular Automaton (ν-$\mathcal{CA}$)). *A* non-uniform Cellular Automaton *(ν-$\mathcal{CA}$) is a structure $\langle A, \{h_i, r_i\}_{i\in\mathbb{Z}}\rangle$ defined by a family of local rules $h_i : A^{2r_i+1} \to A$ of radius r_i all based on the same set of states A.*

Similarly to CA, one can define the global rule of a ν-$\mathcal{CA}$ as the map $H : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ given by the law

$$\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \quad H(x)_i = h_i(x_{i-r_i}, \ldots, x_{i+r_i}) \ .$$

From now on, we identify a ν-$\mathcal{CA}$ (resp., CA) with the discrete dynamical system induced by itself or even with its global rule H (resp., F).

It is well known that the Hedlund's Theorem [17] characterizes CA as the class of continuous functions commuting with the shift map $\sigma : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$, where $\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \sigma(x)_i = x_{i+1}$. It is possible to give a characterization also of the class of ν-$\mathcal{CA}$ since it is straightforward to prove the following.

Proposition 1. *A function $H : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ is the global map of a ν-$\mathcal{CA}$ iff it is continuous.*

The previous proposition gives the important information that the pair $(A^{\mathbb{Z}}, H)$ is a discrete dynamical system, but in many practical cases this setting is by far too general to be useful. Therefore, we are going to focus our attention only over two special subclasses of ν-$\mathcal{CA}$.

Definition 2 (dν-$\mathcal{CA}$). *A ν-$\mathcal{CA}$ H is a dν-$\mathcal{CA}$ if there exist two natural k, r and a rule $h : A^{2r+1} \to A$ such that for all integers i with $|i| > k$ it holds that $h_i = h$. In this case, we say that H has default rule h.*

Definition 3 (rν-$\mathcal{CA}$). *A ν-$\mathcal{CA}$ H is a rν-$\mathcal{CA}$ if there exists an integer r such that any local rule h_i has radius r. In this case, we say that H has radius r.*

The first class restricts the number of positions at which non-default rules can appear, while the second class restricts the number of different rules but not the number of occurrences nor it imposes the presence of a default rule. Some easy examples follow.

Example 1. Consider the ν-$\mathcal{CA}$ $H^{(1)} : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ defined as $\forall x \in A^{\mathbb{Z}}, H^{(1)}(x)_i = 1$ if $i = 0$; 0 otherwise. Remark that $H^{(1)}$ is a dν-$\mathcal{CA}$ which cannot be a CA since it does not commute with σ. So the class of ν-$\mathcal{CA}$ is larger than the original class of all CA.

Example 2. Consider the ν-$\mathcal{CA}$ $H^{(2)} : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ defined as $\forall x \in A^{\mathbb{Z}}, H^{(2)}(x)_i = 1$ if i is even; 0 otherwise. Remark that $H^{(2)}$ is a rν-$\mathcal{CA}$ but not a dν-$\mathcal{CA}$.

Example 3. Consider the ν-$\mathcal{CA}$ $H^{(3)} : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ defined as $\forall x \in A^{\mathbb{Z}}, H^{(3)}(x)_i = x_0$. Remark that $H^{(3)}$ is a ν-$\mathcal{CA}$ but not a rν-$\mathcal{CA}$.

Focusing the study on dν-$\mathcal{CA}$ and rν-$\mathcal{CA}$ is not a great loss in generality since (at some extent) each ν-$\mathcal{CA}$ can be viewed as the limit of a sequence of dν-$\mathcal{CA}$.

Proposition 2. *$\mathcal{CA} \subsetneq d\nu\text{-}\mathcal{CA} \subsetneq r\nu\text{-}\mathcal{CA} \subsetneq \nu\text{-}\mathcal{CA}$, where $\mathcal{CA}$ is the set of all CA.*

Proof. The inclusions $\subseteq$ easily follow from the definitions. For the strict inclusions refer to Examples 1 to 3. □

Similarly to what happens for CA one can prove the following.

Proposition 3. *For every rν-$\mathcal{CA}$ H on the alphabet A there exists a radius 1 rν-$\mathcal{CA}$ H' and a bijective continuous mapping ϕ such that $H \circ \phi = \phi \circ H'$. That is, H is topologically conjugated to H'.*

Proof. Let H be a rν-$\mathcal{CA}$. If H has radius $r = 1$ then this result is trivially true using the identity as a conjugacy map. If $r > 1$, let $B = A^r$ and define $\phi : A^{\mathbb{Z}} \to B^{\mathbb{Z}}$ as $\forall i \in \mathbb{Z}, \phi(x)_i = x_{[ir,(i+1)r)}$. Then, it is not difficult to see that the rν-$\mathcal{CA}$ $(B^{\mathbb{Z}}, H')$ of radius 1 defined as $\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, H'(x)_i = h'_i(x_{i-1}, x_i, x_{i+1})$ is topologically conjugated to H via ϕ, where $\forall u, v, w \in B, \forall i \in \mathbb{Z}, \forall j \in \{0, \ldots, r-1\}, (h'_i(u, v, w))_j = h_{ir+j}(u_{[j,r)} v w_{[0,j]})$. □

Finally, the following result shows that every rν-$\mathcal{CA}$ is a subsystem of a suitable CA. Therefore, the study of rν-$\mathcal{CA}$ dynamics might reveal new properties for CA and *vice-versa*.

Theorem 1. *Any rν-$\mathcal{CA}$ $H : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ of radius r is a subsystem of a CA, i.e., there exist a CA $F : B^{\mathbb{Z}} \to B^{\mathbb{Z}}$ on a suitable alphabet B and a continuous injection $\phi : A^{\mathbb{Z}} \to B^{\mathbb{Z}}$ such that $\phi \circ H = F \circ \phi$.*

Proof. Consider a rν-$\mathcal{CA}$ $H : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ of radius r. Remark that there are only $n = |A|^{|A|^{2r+1}}$ distinct functions $h_i : A^{2r+1} \to A$. Take a numbering $(f_j)_{1 \leq j \leq n}$ of these functions and let $B = A \times \{1, \ldots, n\}$. Define the mapping $\phi : A^{\mathbb{Z}} \to B^{\mathbb{Z}}$ such that $\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \phi(x)_i = (x_i, k)$, where k is the integer for which $H(x)_i = f_k(x_{i-r}, \ldots, x_{i+r})$. Clearly, ϕ is injective and continuous. Now, define a CA $F : B^{\mathbb{Z}} \to B^{\mathbb{Z}}$ using the local rule $f : B^{2r+1} \to B$ such that

$$f((x_{-r}, k_{-r}), \ldots, (x_0, k_0), \ldots, (x_r, k_r)) = (f_{k_0}(x_{-r}, \ldots, x_r), k_0) \ .$$

It is not difficult to see that $\phi \circ H = F \circ \phi$. □

3 CA vs. ν-$\mathcal{CA}$

In this section, we investigate some differences in dynamical behavior between CA and ν-$\mathcal{CA}$. As we are going to see, many characteristics which are really specific to the whole class of CA are lost in the larger class of ν-$\mathcal{CA}$. This will be explored by showing via counter-examples that these properties are not satisfied by the whole class of ν-$\mathcal{CA}$.

First of all, let us recall that given a ν-$\mathcal{CA}$ H, a configuration $x \in A^{\mathbb{Z}}$ is an *ultimately periodic* point of H if there exist $p, q \in \mathbb{N}$ such that $H^{p+q}(x) = H^q(x)$. If $q = 0$, then x is *periodic*, i.e., $H^p(x) = x$. The minimum p for which $H^p(x) = x$

holds is called *period* of x. A ν-$\mathcal{CA}$ is *surjective* (resp. *injective*) if its global rule is surjective (resp. injective).

It is well known that in the case of CA the collection of all ultimately periodic configurations is dense in the configurations space $A^{\mathbb{Z}}$. This property is not true in the general case of ν-$\mathcal{CA}$. We will show this result making reference to the following interesting example of ν-$\mathcal{CA}$.

Example 4. Let $A = \{0, 1\}$ and define the following dν-$\mathcal{CA}$ $H^{(4)} : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ as

$$\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \quad H^{(4)}(x)_i = \begin{cases} x_i & \text{if } i = 0 \\ x_{i-1} & \text{otherwise} \end{cases} .$$

The first no–go result is relative to the above example.

Proposition 4. *The set of ultimately periodic points of $H^{(4)}$ is not dense.*

Proof. Let $H = H^{(4)}$. Denote by P and U the sets of periodic and ultimately periodic points, respectively. Let $E = \{x \in A^{\mathbb{Z}} : \forall i \in \mathbb{N}, x_i = x_0\}$. Take $x \in P$ with $H^p(x) = x$. Remark that the set $B_x = \{i \in \mathbb{N} : x_i \neq x_0\}$ is empty. Indeed, by contradiction, assume that $B \neq \emptyset$ and let $m = \min B$. It is easy to check that $\forall y \in A^{\mathbb{Z}}, \forall i \in \mathbb{N}, H^i(y)_{[0,i]} = {y_0}^{i+1}$, hence $x_m = H^{pm}(x)_m = x_0$, contradiction. Thus $x \in E$ and $P \subseteq E$.

Let $y \in H^{-1}(E)$. We show that $B_y = \emptyset$. By contradiction, let $n = \min B_y$. Since $H(y)_{n+1} = y_n \neq y_0 = H(y)_0$, then $H(y) \notin E$. Contradiction, then $y \in E$ and $H^{-1}(E) \subseteq E$. So $\forall n \in \mathbb{N}, H^{-n}(E) \subseteq E$. Moreover, $U = \bigcup_{n\in\mathbb{N}} H^{-n}(P) \subseteq \bigcup_{n\in\mathbb{N}} H^{-n}(E) \subseteq E$ and E is not dense. □

The following proposition proves that $H^{(4)}$ is not surjective, despite it is based on two local rules each of which generates a surjective CA (namely, the identity CA and the shift CA). Moreover, unlike the CA case (see [17]), $H^{(4)}$ has no configuration with an infinite number of pre-images although it is not surjective.

Proposition 5. *The dν-$\mathcal{CA}$ $H^{(4)}$ is not surjective and any configuration has either* 0 *or* 2 *pre-images.*

Proof. Since $\forall x \in A^{\mathbb{Z}}, H^{(4)}(x)_0 = H^{(4)}(x)_1$, configurations in the set $B = \{x \in A^{\mathbb{Z}} : x_0 \neq x_1\}$ have no pre-image. Then any $x \in A^{\mathbb{Z}} \setminus B$ has 2 pre-images y and z such that $\forall i \notin \{-1, 0\}, y_i = z_i = x_{i+1}, y_0 = z_0 = x_0, y_{-1} = 0; z_{-1} = 1$. □

In order to explore some other no-go results, we introduce an other example.

Example 5. Let $A = \{0, 1\}$ and define a ν-$\mathcal{CA}$ $H^{(5)} : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ by

$$\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \quad H^{(5)}(x)_i = \begin{cases} 0 & \text{if } i = 0 \\ x_{i-1} \oplus x_{i+1} & \text{otherwise} \end{cases} ,$$

where $\oplus$ is the xor operator.

The following results show that in the case of ν-$\mathcal{CA}$, the Moore-Myhill theorem on CA [18,19] is no more true.

Proposition 6. *The ν-$\mathcal{CA}$ $H^{(5)}$ is injective on the finite* 0*-configurations but it is not surjective.*

Proof. It is evident that $H^{(5)}$ is not surjective. Let x, y be two finite configurations such that $H^{(5)}(x) = H^{(5)}(y)$. By contradiction, assume that $x_i \neq y_i$, for some $i \in \mathbb{Z}$. Without loss of generality, assume that $i \in \mathbb{N}$. Since $x_i \oplus x_{i+2} = H^{(5)}(x)_{i+1} = H^{(5)}(y)_{i+1} = y_i \oplus y_{i+2}$, it holds that $x_{i+2} \neq y_{i+2}$ and, by induction, $\forall j \in \mathbb{N}, x_{i+2j} \neq y_{i+2j}$. We conclude that $\forall j \in \mathbb{N}, x_{i+2j} = 1$ or $y_{i+2j} = 1$ contradicting the assumption that x and y are finite. □

A ν-$\mathcal{CA}$ H is *positively expansive* if there exists $\epsilon > 0$ such for any pair of distinct $x, y \in A^{\mathbb{Z}}$, $d(H^n(x), H^n(y)) > \epsilon$ for some $n \in \mathbb{N}$. A ν-$\mathcal{CA}$ H is *transitive* if for any distinct pair of non-empty open sets $U, V \subset A^{\mathbb{Z}}$, there exists $n \in \mathbb{N}$ such that $H^n(U) \cap V \neq \emptyset$. Both of these properties are considered as standard indicators of chaotic behavior. We will show now that, unlike the CA case, positively expansive ν-$\mathcal{CA}$ are not necessarily transitive nor surjective.

Proposition 7. *The ν-$\mathcal{CA}$ $H^{(5)}$ is positively expansive but it is neither transitive nor surjective.*

Proof. Let $H = H^{(5)}$. By Proposition 6, H is not surjective and hence it is not transitive. Let x and y be two distinct configurations. Without loss of generality, one can assume that there exists $k = \min\{i \in \mathbb{N}, x_i \neq y_i\}$. If $k \leq 1$, we have $d(H^0(x), H^0(y)) \geq \frac{1}{2}$. Otherwise, $H(x)_{k-1} = x_{k-2} \oplus x_k \neq y_{k-2} \oplus y_k = H(y)_{k-1}$ and $H(x)_{[0,k-2]} = H(y)_{[0,k-2]}$. By induction on $k \in \mathbb{N}$, it is easy to see that $H^{k-1}(x)_1 \neq H^{k-1}(y)_1$. Hence $d(H^{k-1}(x), H^{k-1}(y)) \geq \frac{1}{2}$. Thus H is positively expansive with expansivity constant $\frac{1}{2}$. □

Example 6. Let $A = \{0, 1\}$ and define the ν-$\mathcal{CA}$ $H^{(6)} : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ as follows

$$\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \quad H^{(6)}(x)_i = \begin{cases} x_{i+1} & \text{if } i < 0 \\ x_0 & \text{if } i = 0 \\ x_{i-1} & \text{otherwise.} \end{cases}$$

Recall that for CA, the compactness of $A^{\mathbb{Z}}$ and the uniformity of the local rule allow one to prove that injective CA are surjective. The following result shows that this does not hold in the case of ν-$\mathcal{CA}$.

Proposition 8. *The ν-$\mathcal{CA}$ $H^{(6)}$ is injective but not surjective.*

Proof. Let $H = H^{(6)}$. Concerning non-surjectivity, just remark that only configurations x such that $x_{-1} = x_0 = x_1$ have a pre-image. Let $x, y \in A^{\mathbb{Z}}$ with $H(x) = H(y)$. Then, we have $\forall i > 0, x_{i-1} = y_{i-1}$ and $\forall i < 0, x_{i+1} = y_{i+1}$. So $x = y$ and H is injective. □

4 Surjectivity

In the context of (1D) CA, the notion of De Bruijn graph is very handy to find fast decision algorithms for surjectivity, injectivity and openness. Here, we extend

this notion to work with dν-$\mathcal{CA}$ and find decision algorithm for surjectivity. We stress that surjectivity is undecidable for two (or higher) dimensional dν-$\mathcal{CA}$, since surjectivity is undecidable for 2D CA [20].

Definition 4. *Consider a dν-$\mathcal{CA}$ H of radius r and let f be its default rule. Let $k \in \mathbb{N}$ be the largest integer such that $h_k \neq f$ or $h_{-k} \neq f$. The* De Bruijn graph *of H is the pair (V, E) where $V = A^{2r} \times \{-k-1, \ldots, k+1\}$ and E is the set of pairs $((a, i), (b, j))$ with label in $A \times \{0, 1\}$ and such that $\forall i \in \{0, \ldots, 2r-1\}, a_i = b_{i+1}$ and one of the following condition is verified*

a) $i = j = -k - 1$; in this case the label is $(f(a_0 b), 0)$
b) $i + 1 = j$; in this case the label is $(h_k(a_0 b), 0)$
c) $i = j = k + 1$; in this case the label is $(f(a_0 b), 1)$

By this graph, a configuration can be seen as a bi-infinite path on vertexes which passes once from a vertex whose second component is in $[-k + 1, k - 1]$ and infinite times from other vertices. The second component of vertices allows to single out the positions of local rules different from the default one. The image of a configuration is the sequence of first components of edge labels.

Theorem 2. *Surjectivity is decidable for dν-$\mathcal{CA}$.*

Proof. We show that a dν-$\mathcal{CA}$ H is surjective iff its De Bruijn graph G recognizes the language $(A \times \{0\})^*(A \times \{1\})^*$ when it is considered as the graph of a finite state automaton. Denote by (a_1, a_2) any word of $(A \times \{0, 1\})^*$. Let k be as in Definition 4.

Assume that H is surjective and take $u \in (A \times \{0\})^*(A \times \{1\})^*$. Let n be the number of 0's appearing in u (in the second component). We have three cases:

1. If $n = 0$ then there exists $v \in A^*$ such that $f(v) = u$ and we can construct u by the sequence of vertices $(v_{[0,2r)}, k + 1), \ldots, (v_{[|v|-2r,|v|)}, k + 1)$.
2. If $0 < n < |u|$ then there exists $v \in A^*$ such that $h_{k+1-n}(v) = u$. We can construct u by the sequence of vertices $(v_{[0,2r)}, u_0), \ldots, (v_{[|v|-2r,|v|)}, u_{|v|-2r-1})$ where
$$u_j = \begin{cases} -k - 1 & \text{if } k + 1 - n + j < k \\ k + 1 & \text{if } k + 1 - n + j > k \\ k + 1 - n + j & \text{otherwise} \end{cases}$$
3. If $n = |u|$ then there exists $v \in A^*$ such that $f(v) = u$ and we can construct u by the sequence of vertices $(v_{[0,2r)}, -k - 1), \ldots, (v_{[|v|-2r,|v|)}, -k - 1)$.

For the opposite implication, assume that G recognizes $(A \times \{0\})^*(A \times \{1\})^*$. Take $y \in A^{\mathbb{Z}}$ and let $n > k$. Since G recognizes $(y_{[-n,n]}, 0^{n+k+1}1^{n-k})$, there exists $v \in A^*$ such that $H(v)_{[-n,n]} = y_{[-n,n]}$. Set $X_n = \left\{x \in A^{\mathbb{Z}}, x_{[n,n]} = y_{[-n,n]}\right\}$. For any $n \in \mathbb{N}$, X_n is non-empty and compact. Moreover, $X_{n+1} \subseteq X_n$. Therefore, $X = \bigcap_{n \in \mathbb{N}} X_n \neq \emptyset$ and $H(X) = \{y\}$. Hence, H is surjective. □

5 More on Dynamical Properties

5.1 Equicontinuity

Let H be a ν-$\mathcal{CA}$. A configuration $x \in A^{\mathbb{Z}}$ is an *equicontinuity point* for H if $\forall \varepsilon > 0$ there exists $\delta > 0$ such that for all $y \in A^{\mathbb{Z}}$, $d(y,x) < \delta$ implies that $\forall n \in \mathbb{N},\ d(H^n(y), H^n(x)) < \varepsilon$. A ν-$\mathcal{CA}$ is said to be *equicontinuous* if the set E of all its equicontinuous points is the whole $A^{\mathbb{Z}}$, while it is said to be *almost equicontinuous* if E is residual (i.e., E can be obtained by a countable intersection of dense open subsets). A word $u \in A^k$ is s-blocking ($s \leq k$) for a CA F if there exists an offset $j \in [0, k-s]$ such that for any $x, y \in [u]_0$ and any $n \in \mathbb{N}$, $F^n(x)_{[j,j+s-1]} = F^n(y)_{[j,j+s-1]}$. In [21], Kůrka proved that a CA is almost equicontinuous iff it is non-sensitive iff it admits a blocking word.

We now introduce a class of ν-$\mathcal{CA}$ which will be useful in the sequel. It is an intermediate class between dν-$\mathcal{CA}$ and rν-$\mathcal{CA}$.

Definition 5 (n-compatible rν-$\mathcal{CA}$). *A rν-$\mathcal{CA}$ H is* n-compatible with a local rule f *if for any $k \in \mathbb{N}$, there exist two integers $k_1 > k$ and $k_2 < -k$ such that $\forall i \in [k_1, k_1+n) \cup [k_2, k_2+n),\ h_i = f$.*

In other words, a ν-$\mathcal{CA}$ is n-compatible with f if, arbitrarily far from the center of the lattice, there are intervals of length n in which the local rule f is applied.

The notion of blocking word and the related results cannot be directly restated in the context of ν-$\mathcal{CA}$ because some words are blocking just thanks to the uniformity of CA. To overcome this problem we introduce the following notion.

Definition 6 (Strongly blocking word). *A word $u \in A^*$ is* strongly s-blocking *for a CA F of local rule f if there exists an offset $p \in [0, |u| - s]$ such that for any ν-$\mathcal{CA}$ H with $\forall i \in \{0, \ldots, |u|-1\}$, $h_i = f$ it holds that*

$$\forall x, y \in [u]_0, \forall n \geq 0, \quad H^n(x)_{[p,p+s)} = H^n(y)_{[p,p+s)} \ .$$

Roughly speaking, a word is strongly blocking if it is blocking whatever be the perturbations involving the rules in its neighborhood. The following extends Proposition 5.12 in [22] to strongly r-blocking words.

Proposition 9. *Any r radius CA F is equicontinuous iff there exists $k > 0$ such that any word $u \in A^k$ is strongly r-blocking for F.*

Proof. If any word is strongly blocking then F is obviously equicontinuous. For the opposite implication, by [22, Prop. 5.12], there exist $p > 0$ and $q \in \mathbb{N}$ such that $F^{q+p} = F^q$. As a consequence, we have that $\forall u \in A^*, |u| > 2(q+p)r \Rightarrow f^{p+q}(u) = f^q(u)_{[pr,|u|-(2q+p)r)}$. Let H be a ν-$\mathcal{CA}$ such that $h_i = f$ for each $i \in \{0, \ldots, (2p+2q+1)r-1\}$. For any $x \in A^{\mathbb{Z}}$ and $i \geq 0$, consider the following words: $s^{(i)} = H^i(x)_{[0,qr)}$, $t^{(i)} = H^i(x)_{[qr,(q+p)r)}$, $u^{(i)} = H^i(x)_{[(q+p)r,(q+p+1)r)}$, $v^{(i)} = H^i(x)_{[(q+p+1)r,(q+2p+1)r)}$, $w^{(i)} = H^i(x)_{[(q+2p+1)r,(2q+2p+1)r)}$. For all $i \in 0, \ldots, q+p$, $u^{(i)}$ is fully determined by $s^{(0)}t^{(0)}u^{(0)}v^{(0)}w^{(0)} = x_{[0,(2q+2p+1)r)}$. Moreover, for any natural i, we have $u^{(i+q+p)} = f^{q+p}(s^{(i)}t^{(i)}u^{(i)}v^{(i)}w^{(i)}) =$

$f^q(s^{(i)}t^{(i)}u^{(i)}v^{(i)}w^{(i)})_{[pr,(p+1)r)} = (t^{(i+q)}u^{(i+q)}v^{(i+q)})_{[pr,(p+1)r)} = u^{(i+q)}$. Summarizing, for all $i \in \mathbb{N}$, $u^{(i)}$ is determined by the word $x_{[0,(2q+2p+1)r)}$ which is then strongly r-blocking. Since x had been chosen arbitrarily, we have the thesis. □

Theorem 3. *Let F be a CA with local rule f admitting a strongly r-blocking word u. Let H be a $r\nu$-$\mathcal{CA}$ of radius r. If H is $|u|$-compatible with f then H is almost equicontinuous.*

Proof. Let p and n be the offset and the length of u, respectively. For any $k \in \mathbb{N}$, consider the set $T_{u,k}$ of configurations $x \in A^{\mathbb{Z}}$ having the following property $\mathcal{P}$: there exist $l > k$ and $m < -k$ such that $x_{[l,l+n)} = x_{[m,m+n)} = u$ and $\forall i \in [l, l+n) \cup [m, m+n)\ h_i = f$. Remark that $T_{u,k}$ is open, being a union of cylinders. Clearly, each $T_{u,k}$ is dense, thus the set $T_u = \bigcap_{k \geq n} T_{u,k}$ is residual. We claim that any configuration in T_u is an equicontinuity point. Indeed, choose arbitrarily $x \in T_u$. Set $\epsilon = 2^{-k}$, where $k \in \mathbb{N}$ is such that $x \in T_{u,k}$. Then, there exist $k_1 > k$ and $k_2 < -k - n$ satisfying $\mathcal{P}$. Fix $\delta = \min\{2^{-(k_1+n)}, 2^{-k_2}\}$ and let $y \in A^{\mathbb{Z}}$ be such that $d(x, y) < \delta$. Then $y_{[k_2,k_1+|u|)} = x_{[k_2,k_1+|u|)}$. Since u is r-blocking, $\forall t \in \mathbb{N}$, $H^t(x)$ and $H^t(y)$ are equal inside the intervals $[k_1+p, k_1+p+r]$ and $[k_2+p, k_2+p+r]$, then $d(H^t(x), H^t(y)) < \epsilon$. □

In a similar manner one can prove the following.

Theorem 4. *Let F be an equicontinuous CA of local rule f. Let $k \in \mathbb{N}$ be as in Proposition 9. Any $r\nu$-$\mathcal{CA}$ k-compatible with f is equicontinuous.*

5.2 Sensitivity to the Initial Conditions

Recall that a CA F is *sensitive to the initial conditions* (or simply *sensitive*) if there exists a constant $\varepsilon > 0$ such that for any configuration $x \in A^{\mathbb{Z}}$ and any $\delta > 0$ there is a configuration $y \in A^{\mathbb{Z}}$ such that $d(y, x) < \delta$ and $d(F^n(y), F^n(x)) \geq \varepsilon$ for some $n \in \mathbb{N}$.

Example 7. Let $A = \{0, 1, 2\}$ and consider the CA whose local rule $f : A^3 \to A$ is defined as follows: $\forall x, y \in A$, $f(x, 0, y) = 1$ if $x = 1$ or $y = 1$, 0 otherwise; $f(x, 1, y) = 2$ if $x = 2$ or $y = 2$, 0 otherwise; $f(x, 2, y) = 0$ if $x = 1$ or $y = 1$, 2 otherwise.

Proposition 10. *The CA defined in Example 7 is almost equicontinuous.*

Proof. Just remark that the number of 0s inside the word 20^i2 is non-decreasing. Thus 202 is a 1-blocking word. □

The following example defines a ν-$\mathcal{CA}$ which is sensitive to the initial conditions although its default rule give rise to an almost equicontinuous CA.

Example 8. Consider the dν-$\mathcal{CA}$ $H^{(8)} : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ defined as follows

$$\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \quad H^{(8)}(x)_i = \begin{cases} 1 & \text{if } i = 0 \\ f(x_{i-1}, x_i, x_{i+1}) & \text{otherwise} \end{cases},$$

where f and A are as in Example 7.

Remark that positive and negative cells do not interact each other under the action of $H^{(8)}$. Therefore, in order to study the behavior of $H^{(8)}$, it is sufficient to consider the action of $H^{(8)}$ on $A^{\mathbb{N}}$.

Lemma 1. *For any* $u \in A^*$, $\exists n_0 \in \mathbb{N}$ *such that* $\forall n > n_0, (H^{(8)})^n(u0^\infty)_1 = 1$.

Lemma 2. $\forall u \in A^*$, $\forall n_0 \geq 0, \exists n > n_0, (H^{(8)})^n(u2^\infty)_1 = 2$.

Proposition 11. *The dν-$\mathcal{CA}$* $H^{(8)}$ *is sensitive.*

Proof. Let $H = H^{(8)}$ and $\mathcal{F}$ be the set of all a-finite configurations for $a \in \{0, 2\}$. By a theorem of Knudesen [23], we can prove the statement *w.r.t.* $\mathcal{F}$. Then, for any $u \in A^*$. Build $x = u0^\infty$ and $y = u2^\infty$. By Lemma 1 and 2, there exists n such that $1 = H^n(x)_1 \neq H^n(y)_1 = 2$. And hence H is sensitive with sensitivity constant $\epsilon = 1/2$. □

The following example shows that default rules individually defining almost equicontinuous CA can also constitute ν-$\mathcal{CA}$ that have a completely different behavior from the one in Example 8.

Example 9. Let $A = \{0, 1, 2\}$ and define the local rule $f : A^3 \to A$ as: $\forall x, y, z \in A$, $f(x, y, z) = 2$ if $x = 2$ or $y = 2$ or $z = 2$, z otherwise. The CA F of local rule f is almost equicontinuous since 2 is a blocking word. The restriction of F to $\{0, 1\}^{\mathbb{Z}}$ gives the shift map which is sensitive. Thus F is not equicontinuous. Define now the following dν-$\mathcal{CA}$ $H^{(9)}$:

$$\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \quad H^{(9)}(x)_i = \begin{cases} 2 & \text{if } i = 0 \\ f(x_{i-1}, x_i, x_{i+1}) & \text{otherwise .} \end{cases}$$

Proposition 12. *The dν-$\mathcal{CA}$* $H^{(9)}$ *is equicontinuous.*

Proof. Let $n \in \mathbb{N}, x, y \in A^{\mathbb{Z}}$ be such that $x_{[-2n,2n]} = y_{[-2n,2n]}$. Since H is of radius 1, $\forall k \leq n, H^k(x)_{[-n,n]} = H^k(y)_{[-n,n]}$ and $\forall k > n$, $H^k(x)_{[-n,n]} = 2^{2n+1} = H^k(y)_{[-n,n]}$. So, H is equicontinuous. □

5.3 Expansivity and Permutivity

Recall that a rule $f : A^{2r+1}$ is *leftmost* (resp., rightmost) *permutive* if $\forall u \in A^{2r}, \forall b \in A, \exists! a \in A, f(au) = b$ (resp., $f(ua) = b$). This definition can be easily extended to ν-$\mathcal{CA}$. Indeed, we say that a rν-$\mathcal{CA}$ is leftmost (resp. rightmost) permutive if all h_i are leftmost (resp. rightmost) permutive. A ν-$\mathcal{CA}$ is *permutive* if it is leftmost or rightmost permutive.

In a very similar way to CA, given a ν-$\mathcal{CA}$ H and two integers $a, b \in \mathbb{N}$ with $a < b$, the *column subshift* $(\Sigma_{[a,b]}, \sigma)$ of H is defined as follows $\Sigma_{[a,b]} = \{y \in (A^{b-a+1})^{\mathbb{N}} : \exists x \in A^{\mathbb{Z}}, \forall i \in \mathbb{N}, y_i = H^i(x)_{[a,b]}\}$. Consider the map $\mathcal{I}_{[a,b]} : A^{\mathbb{Z}} \to \Sigma_{[a,b]}$ defined as $\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{N}, \mathcal{I}_{[a,b]}(x)_i = H^i(x)_{[a,b]}$. It is not difficult to see that $\mathcal{I}$ is continuous and surjective. Moreover $H \circ \mathcal{I}_{[a,b]} = \mathcal{I}_{[a,b]} \circ \sigma$. Thus (Σ_I, σ) is a *factor* of the ν-$\mathcal{CA}$ $(A^{\mathbb{Z}}, H)$ and we can lift some properties from $(\Sigma_{[a,b]}, \sigma)$ to $(A^{\mathbb{Z}}, H)$. The following result tells that something stronger happens in the special case of leftmost and rightmost permutive rν-$\mathcal{CA}$.

Theorem 5. *Any leftmost and rightmost permutive $r\nu$-$\mathcal{CA}$ of radius r is conjugated to the full shift* $((A^{2r})^{\mathbb{N}}, \sigma)$.

Proof. Just remark that the map $\mathcal{I}_{[1,2r]} : A^{\mathbb{Z}} \to (A^{2r})^{\mathbb{N}}$ is bijective. □

The requirements of the previous theorem are very strong. Indeed, there exist ν-$\mathcal{CA}$ which are topologically conjugated to a full shift but that are not permutive. As an example, consider the ν-$\mathcal{CA}$ H defined as follows

$$\forall x \in A^{\mathbb{Z}}, \forall i \in \mathbb{Z}, H(x)_i = \begin{cases} x_{i-1} \text{ if } i \leq 0 \\ x_{i+1} \text{ otherwise } . \end{cases}$$

Then, $\Sigma_{[0,1]} = (A^2)^{\mathbb{N}}$ et $\mathcal{I}_{[0,1]}$ is injective.

6 Conclusions

In this paper we started exploring the dynamical behavior of ν-$\mathcal{CA}$. Many specific properties for CA are no longer true for ν-$\mathcal{CA}$. However, under certain conditions, some stability forms turned out to be quite robust when altering a CA to obtain a ν-$\mathcal{CA}$. Despite of the many no-go results proved in this paper, we strongly believe that ν-$\mathcal{CA}$ can be useful for many practical applications and hence deserve further studies.

References

1. Farina, F., Dennunzio, A.: A predator-prey ca with parasitic interactions and environmentals effects. Fundamenta Informaticae 83, 337–353 (2008)
2. Chaudhuri, P., Chowdhury, D., Nandi, S., Chattopadhyay, S.: Additive Cellular Automata Theory and Applications, vol. 1. IEEE Press, Los Alamitos (1997)
3. Chopard, B.: Modelling physical systems by cellular automata. In: Rozenberg, G., et al. (eds.) Handbook of Natural Computing: Theory, Experiments, and Applications. Springer, Heidelberg (to appear, 2009)
4. Formenti, E., Kůrka, P.: Dynamics of cellular automata in non-compact spaces. In: Meyers, B. (ed.) Mathematical basis of cellular automata. Encyclopedia of Complexity and System Science. Springer, Heidelberg (2008)
5. Kůrka, P.: Topological dynamics of one-dimensional cellular automata. In: Meyers, B. (ed.) Mathematical basis of cellular automata. Encyclopedia of Complexity and System Science. Springer, Heidelberg (2008)
6. Cervelle, J., Dennunzio, A., Formenti, E.: Chaotic behavior of cellular automata. In: Meyers, B. (ed.) Mathematical basis of cellular automata. Encyclopedia of Complexity and System Science. Springer, Heidelberg (2008)
7. Kari, J.: Tiling problem and undecidability in cellular automata. In: Meyers, B. (ed.) Mathematical basis of cellular automata. Encyclopedia of Complexity and System Science. Springer, Heidelberg (2008)
8. Di Lena, P., Margara, L.: Undecidable properties of limit set dynamics of cellular automata. In: 26th Symposium on Theoretical Aspects of Computer Science (STACS 2009). LNCS. Springer, Heidelberg (to appear, 2009)

9. Di Lena, P., Margara, L.: Computational complexity of dynamical systems: the case of cellular automata. Information and Computation 206, 1104–1116 (2008)
10. Dennunzio, A., Formenti, E.: Decidable properties of 2D cellular automata. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 264–275. Springer, Heidelberg (2008)
11. Dennunzio, A., Formenti, E., Kůrka, P.: Cellular automata dynamical systems. In: Rozenberg, G., et al. (eds.) Handbook of Natural Computing: Theory, Experiments, and Applications. Springer, Heidelberg (to appear, 2009)
12. Dennunzio, A., Formenti, E.: 2D cellular automata: new constructions and decidable properties (submitted, 2009)
13. Acerbi, L., Dennunzio, A., Formenti, E.: Conservation of some dynamcal properties for operations on cellular automata. Theoretical Computer Science (to appear, 2009)
14. Dennunzio, A., Di Lena, P., Formenti, E., Margara, L.: On the directional dynamics of additive cellular automata. Theoretical Computer Science (to appear, 2009)
15. Dennunzio, A., Masson, B., Guillon, P.: Sand automata as cellular automata (submitted, 2009)
16. Dennunzio, A., Guillon, P., Masson, B.: Stable dynamics of sand automata. In: Fifth IFIP Confercence on Theoretical Computer Science (TCS 2008). IFIP, vol. 273, pp. 157–179. Springer, Heidelberg (2008)
17. Hedlund, G.A.: Endomorphisms and automorphisms of the shift dynamical system. Mathematical System Theory 3, 320–375 (1969)
18. Moore, E.F.: Machine models of self-reproduction. In: Proceedings of Symposia in Applied Mathematics, vol. 14, pp. 13–33 (1962)
19. Myhill, J.: The converse to Moore's garden-of-eden theorem. Proceedings of the American Mathematical Society 14, 685–686 (1963)
20. Kari, J.: Reversibility and surjectivity problems of cellular automata. Journal of Computer and System Sciences 48, 149–182 (1994)
21. Kůrka, P.: Languages, equicontinuity and attractors in cellular automata. Ergodic Theory & Dynamical Systems 17, 417–433 (1997)
22. Kůrka, P.: Topological and Symbolic Dynamics. Cours Spécialisés, vol. 11. Société Mathématique de France (2004)
23. Knudsen, C.: Chaos without nonperiodicity. American Mathematical Monthly 101, 563–565 (1994)

A Cryptosystem Based on the Composition of Reversible Cellular Automata

Adam Clarridge and Kai Salomaa

Queen's University, Kingston, Canada
{adam,ksalomaa}@cs.queensu.ca

Abstract. We present conditions which guarantee that a composition of marker cellular automata has the same neighbourhood as each of the individual components. We show that, under certain technical assumptions, a marker cellular automaton has a unique inverse with a given neighbourhood. We use these results to develop a working key generation algorithm for a public-key cryptosystem based on reversible cellular automata originally conceived by Kari. We conclude with a discussion on security and practical considerations for the cryptosystem and give several ideas for future work.

1 Introduction

Cryptography has been a part of our everyday lives for some time now. Most widely-used public-key encryption algorithms rely on advanced number theoretic results to achieve a high level of security, such as RSA, whose security is believed to rely on the hardness of the integer factorization problem. These systems tend to have relatively slow implementations [1], and since we will always want more efficient and secure encryption algorithms, it makes sense to consider alternate techniques. Cellular automata (CA) as a medium for encryption is an attractive idea in theory because most CA can be implemented on very fast hardware [2,3,4], hence a CA-based scheme may have the potential to encrypt and decrypt messages faster than existing techniques.

Most investigations into CA-based cryptosystems have been aimed at traditional secret-key systems [5,6,7,8,9,10]. There appear to be very few CA-based public-key cryptosystems in the literature; one is the Finite Automata Public-Key Cryptosystem, or Tao-Chen cryptosystem [1], although it uses nonhomogeneous CA. Kari's paper [11] outlines an idea for a public-key cryptosystem based on reversible cellular automata, and poses the question of how to implement the key generation algorithm. We now review this paper in some detail, as it is the main reference for our work.

The general objective of a public-key cryptosystem based on reversible cellular automata (RCA) is to design an RCA that is very hard to invert without some secret knowledge. That way, the RCA can be published and its inverse can be kept as the private key. Kari emphasizes the importance that the RCA be at least two-dimensional, since there exist algorithms to invert any one-dimensional RCA [12], and also because of the following theorem.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 314–325, 2009.

Theorem 1. [13] *It is undecidable if a given two-dimensional CA is reversible. This is true even when restricted to CA using the von Neumann neighbourhood.*

This theorem provides a sound theoretical basis for the security of Kari's public-key cryptosystem [11]. The basic idea outlined in the paper was to compose together several simple and reversible 'marker' CA (which we define in Section 2) in order to form a more complex cellular automaton $C = C_n \circ C_{n-1} \circ \cdots \circ C_1$, with inverse

$$C^{-1} = {C_1}^{-1} \circ {C_2}^{-1} \circ \cdots \circ {C_n}^{-1}.$$

Encryption occurs by encoding the message as the initial configuration of the CA, then evolving the composed CA for some k generations to obtain the ciphertext. The inverse automaton does not need to be computed explicitly; one need only apply each component of the composition in succession. The inverse is then applied for k iterations to decrypt the ciphertext. The composition $C_n \circ C_{n-1} \circ \cdots \circ C_1$ is the public key, and each of the inverse automata of the composition $({C_1}^{-1}, {C_2}^{-1}, \ldots, {C_n}^{-1})$ are kept as the private key. A well-constructed public key should be very hard to invert without knowledge of the components $C_1, C_2, \ldots, C_n$ because the neighbourhood size of the inverse automaton would be quite large.

Kari's paper [11] includes an example of a marker RCA composition with a 2-dimensional neighbourhood of 4 cells, and whose inverse has a 2-dimensional neighbourhood of 9 cells. The composition is made up of 5 very simple reversible marker CA. This is of course just an illustrative example, and Kari points out that longer and more complex (more states and a less restricted form) compositions would be needed in order to ensure security against brute force attacks. However, a public key with s states and neighbourhood size n requires s^n entries in its local rule table, so it is essential to try to keep n small so that the public key can be stored in reasonably sized memory.

The main issue preventing the practical implementation of Kari's cryptosystem is the question of how to choose (or randomly generate) reversible marker CA such that the neighbourhood size of the composition remains small. In this paper, we give one possible answer to this question and investigate the resulting working cryptosystem.

We will state some preliminary assumptions and definitions before discussing our results concerning the composition of a class of marker CA in Sections 2 and 3. We give an algorithm[1] for generating public and private keys in Section 4, and discuss practical implementation issues, security considerations, and ideas for future research in Section 5.

2 Preliminaries

In this paper we assume that in a cellular array containing $M_1 M_2 \cdots M_d$ cells, where M_i is the number of cells of each dimension for $i = 1, \ldots, k$, the neighbours of cells near the edge of the cellular array are determined by adding the

[1] Email the first author for a working software prototype.

component indices cyclically (modulo M_i). This is simply the toroidal boundary condition. The term 'neighbourhood' in this paper refers to either the pattern of cells around a cell, or the states themselves in the pattern, depending on the context.

A 'marker' CA is defined by a permutation ϕ of the state set, and a finite collection of patterns $P_1, P_2, \ldots, P_k$ around the origin. For each cell c, the local rule of the marker CA checks if any of the patterns $P_1, P_2, \ldots, P_k$ is present as the neighbourhood of c. If so, the permutation ϕ is applied to c's state, and if not, then c's state does not change. Marker CA are also known as 'marker automorphisms of the one-sided d-shift' [14] in the dynamical systems literature.

We define a 'fixed-domain' marker cellular automaton (or FDM CA) to be a five-tuple (d,S,N,A,f) with dimension d, state set S, neighbourhood vector $N = (\bar{n}_1, \bar{n}_2, \ldots, \bar{n}_k)$, $\bar{n}_i \in \mathbb{Z}^d$ for $i = 1, 2, \ldots, k$, acting set $A \subseteq S^k$ with entries corresponding to the positions defined by N, and a function $f : S \to S$. The local rule of an FDM CA acts on a cell c (in state s) in the following simple way: if the neighbours of c are in a state configuration corresponding to an element of A, then the state of c on the next generation is $f(s)$. Otherwise, the state of c does not change. An FDM CA is just a special type of marker CA where all of the patterns are mappings from N to S, hence the term 'fixed-domain'. Note that, conversely, an arbitrary marker CA can be represented as an FDM CA by choosing N to be sufficiently large.

In this paper, we use the terms 'invertible' and 'reversible' interchangeably when referring to cellular automata. Also we define compositions of cellular automata in the following way: for any two cellular automata C_1 and C_2 acting on the same cellular grid, one generation of the CA $C_2 \circ C_1$ refers to the application of one generation of C_1 followed by one generation of C_2.

3 Theoretical Results

3.1 Neighbourhood Size of Compositions

As we have noted above, for implementing a public-key cryptosystem based on compositions of RCAs, a desirable property is that the composition should have a small neighbourhood size. After defining some terms, we give necessary and sufficient conditions that characterize the effect on neighbourhood size of composing an FDM CA with an arbitrary CA.

Let B be an arbitrary cellular automaton with state set S, neighbourhood $N_B = (\bar{n}_1, \bar{n}_2, \ldots, \bar{n}_k), \bar{n}_i \in \mathbb{Z}^d$, $d \geq 1$, and local transition function $h_B : S^k \to S$ (h_B maps the neighbourhood of a cell to its next state).

Denote the set of all possible configurations of the neighbourhood $N_B = (\bar{n}_1, \bar{n}_2, \ldots, \bar{n}_k)$ of a cell c by

$$\mathbf{S}_{N_B}(c) = \{(s_{\bar{n}_1}, s_{\bar{n}_2}, \ldots, s_{\bar{n}_k}) \mid s_{\bar{n}_i} \in S \text{ for } i = 1, \ldots, k\},$$

where each $s_{\bar{n}_i}$ refers to the state of the cell in position $\bar{n}_i$.

The neighbourhood of the neighbourhood of a cell c contains any cell that is a neighbour to one of c's neighbours. Let us refer to this set as the *second order*

neighbourhood of c. We will assume without loss of generality that each cell is a neighbour to itself, so each cell in the neighbourhood of c belongs to its second order neighbourhood as well.

Denote the collection of all second order neighbourhoods of a cell c with neighbourhood $\mathbf{s} = (s_{\bar{n}_1}, s_{\bar{n}_2}, \ldots, s_{\bar{n}_k}) \in \mathbf{S}_{N_B}(c)$ by

$$\bar{\mathbf{S}}_{N_B}(\mathbf{s}) = \left\{ \begin{bmatrix} t_{\bar{n}_1+\bar{n}_1} & t_{\bar{n}_1+\bar{n}_2} & \ldots & t_{\bar{n}_1+\bar{n}_k} \\ t_{\bar{n}_2+\bar{n}_1} & t_{\bar{n}_2+\bar{n}_2} & \ldots & t_{\bar{n}_2+\bar{n}_k} \\ & \vdots & & \\ t_{\bar{n}_k+\bar{n}_1} & t_{\bar{n}_k+\bar{n}_2} & \ldots & t_{\bar{n}_k+\bar{n}_k} \end{bmatrix} \in S^{k\times k} \;\middle|\; \forall \bar{n} \in N_B, t_{\bar{n}} = s_{\bar{n}} \right\}.$$

The rows of each matrix in $\bar{\mathbf{S}}_{N_B}(\mathbf{s})$ are the neighbourhoods of each of the cells in $\mathbf{s}$. The states in positions $\bar{n}_1, \bar{n}_2, \ldots, \bar{n}_k$ are fixed (they are the states of $\mathbf{s}$), while the rest of the second order neighbourhood is arbitrary.

The automaton B's neighbourhood state changes for a cell c with neighbourhood $\mathbf{s}$ can be described by a function $\bar{\mathbf{h}}_B : \bar{\mathbf{S}}_{N_B}(\mathbf{s}) \to \mathbf{S}_{N_B}(c)$ which maps a second order neighbourhood to the next neighbourhood of c. The second order neighbourhood contains all the information needed in order to determine the next neighbourhood of c.

For the neighbourhood $\mathbf{s} \in \mathbf{S}_{N_B}(c)$ we denote the set of all possible next neighbourhoods by

$$\text{next\_neighbourhood}_B(\mathbf{s}) = \{\bar{\mathbf{h}}_B(\bar{\mathbf{r}}) \mid \bar{\mathbf{r}} \in \bar{\mathbf{S}}_{N_B}(\mathbf{s})\}.$$

Now we can state conditions which characterize change in neighbourhood size due to composition.

Proposition 1. *Let B be an arbitrary CA with dimension d, state set S, neighbourhood N_B, and local state transition function h_B. Let D be an FDM CA (d,S,N_B,A_D,f_D).*

The composition $D \circ B$ has neighbourhood equal to N_B if and only if for all $\mathbf{s} \in \mathbf{S}_{N_B}(c)$,

$$f_D(h_B(\mathbf{s})) \neq h_B(\mathbf{s}) \Rightarrow \begin{array}{l} \text{next\_neighbourhood}_B(\mathbf{s}) \subseteq A_D \text{ or} \\ \text{next\_neighbourhood}_B(\mathbf{s}) \cap A_D = \emptyset \end{array} \quad (1)$$

Proof. Suppose that condition (1) holds. We want to show that for all $\mathbf{s} \in \mathbf{S}_{N_B}(c)$, where c is some cell, we do not need any more information than the neighbourhood $\mathbf{s}$ to compute the transition of $D \circ B$. For neighbourhoods $\mathbf{s} \in \mathbf{S}_{N_B}(c)$ such that the left hand side of condition (1) is false, D always maps $h_B(\mathbf{s})$ to itself. Clearly in this case D does not create a dependence on a larger neighbourhood. Now consider neighbours $\mathbf{s} \in \mathbf{S}_{N_B}(c)$ such that both sides of the implication (1) are true. The right side of (1) means that all next possible neighbourhoods of $\mathbf{s}$ must either be contained in the acting set of D, or completely separate from the acting set of D. So the particular neighbourhood that $\mathbf{s}$ actually gets mapped to by B is not important, since D already knows from $\mathbf{s}$ whether or not it will act. If $\text{next\_neighbourhood}_B(\mathbf{s}) \subseteq A_D$, then D will apply

f_D to $h_B(\mathbf{s})$, and if next_neighbourhood$_B(\mathbf{s})$ is disjoint from A_D, then D will apply the identity map to $h_B(\mathbf{s})$. Thus, we can compute the transition of $D \circ B$ at a cell c knowing just the current states of its neighbours, $\mathbf{s}$.

Conversely, assume that (1) does not hold. This means that for some $\mathbf{s} \in \mathbf{S}_{N_B}(c)$ there exist $\bar{\mathbf{r}}_1, \bar{\mathbf{r}}_2 \in \bar{\mathbf{S}}_{N_B}(\mathbf{s})$ such that $\bar{\mathbf{h}}_B(\bar{\mathbf{r}}_1) \in A_D$ and $\bar{\mathbf{h}}_B(\bar{\mathbf{r}}_2) \notin A_D$, and that $f_D(h_B(\mathbf{s})) \neq h_B(\mathbf{s})$. Recall that $\bar{\mathbf{r}}_1$ and $\bar{\mathbf{r}}_2$ agree on the states of $\mathbf{s}$, and that $\bar{\mathbf{h}}_B$ denotes that function that maps a second order neighbourhood to a neighbourhood of B.

Consider a collection of cells in the configuration of $\bar{\mathbf{r}}_1$. When we apply $D \circ B$, the B automaton changes the states of $\mathbf{s}$ to a neighbourhood which is in A_D. The D automaton is then applied. So the next state of the cell c is $f_D(h_B(\mathbf{s}))$.

On the other hand, consider a collection of cells in the configuration of $\bar{\mathbf{r}}_2$. When we apply $D \circ B$, the B automaton changes the states of $\mathbf{s}$ to a neighbourhood which is not in A_D. The D automaton applies the identity map. So the next state of the cell c is $h_B(\mathbf{s})$.

Since $f_D(h_B(\mathbf{s})) \neq h_B(\mathbf{s})$, this means that the CA $D \circ B$ cannot have the neighbourhood N_B, since it depends on one or more of the states of $\bar{\mathbf{r}}_1$ and $\bar{\mathbf{r}}_2$ which differ and are outside of N_B. □

The condition of Proposition 1 can be used to inductively define a sequence of FDM CAs $C_1, C_2, \ldots, C_n$ such that $C_1 \circ C_2 \circ \cdots \circ C_n$ has the same neighbourhood as each of its components.

One interesting property of FDM CAs is that a carefully chosen composition can represent any cellular automaton.

Proposition 2. *Every cellular automaton C with neighbourhood N_C of size k and state space S can be represented exactly by a composition of $|S|^k + 1$ FDM CAs with the same neighbourhood, if the FDM CAs are allowed $|S| + |S|^k$ states.*

The proof is a fairly straightforward construction and is omitted for brevity, but can be found in [15]. Note that the upper bounds on the number of FDM CA required in the composition and the number of extra states required are not meant to be tight.

3.2 Reversibility

We now discuss the reversibility of FDM CA. In the following text we use the notation $\mathbf{s}[0]$ to refer to the state in the 'zero' position of a neighbourhood vector $\mathbf{s} \in \mathbf{S}_N$.

Lemma 1. *Let $C = (d, S, N, A_C, f)$ be an FDM CA, and assume the cell itself is part of N $(0 \in N)$ without loss of generality. Denote*

$$B = \{\mathbf{a} \in A_C \mid f(\mathbf{a}[0]) \neq \mathbf{a}[0]\}.$$

Then (d, S, N, B, f) is equivalent with C.

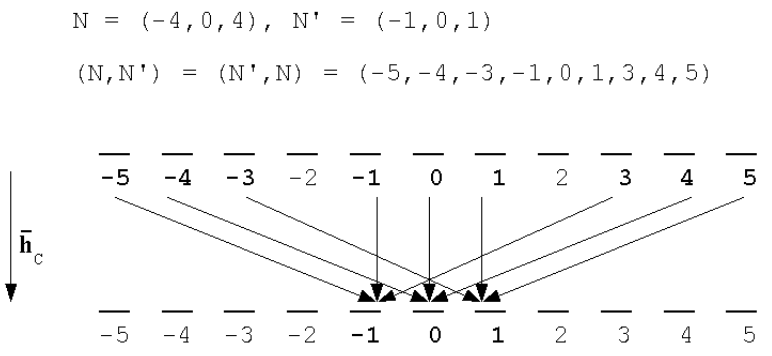


Fig. 1. An example of an (N, N')-neighbourhood mapping to an N' neighbourhood

Proof. The proof is immediate because tuples of A_C not in B do not affect the computation in any way. □

We say that an FDM CA C with acting set A_C is *reduced* if for every $\mathbf{a} \in A_C$, $f(\mathbf{a}[0]) \neq \mathbf{a}[0]$. By Lemma 1, without loss of generality we can assume that an arbitrary FDM CA is reduced.

We now generalize some of the definitions from Proposition 1 for an arbitrary FDM CA C=(d,S,N,A_C,f).

Let an arbitrary neighbourhood be denoted by N'. Then let the (N, N')-neighbourhood of a neighbourhood $\mathbf{s} \in \mathbf{S}_N$ be the configuration containing the N neighbours of each of the elements in the N' neighbourhood. Let the set of all (N, N')-neighbourhoods of $\mathbf{s}$ be denoted by $\bar{\mathbf{S}}_{(N,N')}(\mathbf{s})$. Note that an N-neighbourhood of an N'-neighbourhood is the same as an N'-neighbourhood of an N-neighbourhood, which can be more formally stated as follows. Let r be a vector in $\bar{\mathbf{S}}_{(N,N')}(\mathbf{s})$ and let $r_{N'}$ be the 'restriction' of r to the neighbourhood N', that is, $r_{N'}$ is an N'-neighbourhood around the zero-position. Then the (N', N)-neighbourhood of $r_{N'}$ equals r.

Let the transition function of C from (N, N')-neighbourhoods to N' neighbourhoods be denoted by $\bar{\mathbf{h}}_C$, which takes an (N, N')-neighbourhood configuration and the neighbourhood N' as input, and outputs C's action with that configuration on the neighbourhood of size N'. An illustration of an (N, N')-neighbourhood and how it maps to an N' neighbourhood is given in Figure 1.

Let the transition function from neighbourhoods to sets of possible output neighbourhoods be denoted by

$$\text{next\_neighbourhood}_C(\mathbf{s}, N') = \{\ \bar{\mathbf{h}}_C(\bar{\mathbf{r}}, N') \mid \bar{\mathbf{r}} \in \bar{\mathbf{S}}_{(N,N')}(\mathbf{s})\ \}.$$

The following result characterizes when a given FDM CA with neighbourhood N has an FDM CA inverse with neighbourhood N'.

Proposition 3. *Let C be a reduced FDM CA (d,S,N,A_C,f). Denote*

$$X = \bigcup_{\mathbf{a} \in A_C} \text{next\_neighbourhood}_C(\mathbf{a}, N'). \tag{2}$$

Then C has an FDM CA inverse with state set S and neighbourhood N' if and only if

$$(\forall\, \mathbf{a} \notin A_C)\;\; f(\mathbf{a}[0]) \neq \mathbf{a}[0] \Rightarrow \text{next\_neighbourhood}_C(\mathbf{a}, N') \cap X = \emptyset. \quad (3)$$

Proof. Assume condition (3) holds. Let us choose $C^{-1} = (d, S, N', A_{C^{-1}}, f^{-1})$ where $A_{C^{-1}} = X$, and show that it inverts C. Consider an arbitrary $\bar{\mathbf{r}} \in \bar{\mathbf{S}}_{(N,N')}(\mathbf{a})$ of a neighbourhood $\mathbf{a} \in \mathbf{S}_N$, such that $\bar{\mathbf{h}}_C(\bar{\mathbf{r}}, N') = \mathbf{b}$ $(\in \mathbf{S}_{N'})$. If $\mathbf{a} \in A_C$, then we know that $\mathbf{b} \in A_{C^{-1}}$ so C^{-1} will map $\mathbf{b}[0]$ to $f^{-1}(\mathbf{b}[0]) = \mathbf{a}[0]$. Now consider the case where $\mathbf{a} \notin A_C$. In this case we know that $\mathbf{a}[0] = \mathbf{b}[0]$. If $f(\mathbf{a}[0]) = \mathbf{a}[0] (= \mathbf{b}[0])$, then C^{-1} correctly maps $\mathbf{b}[0]$ back to itself since $f^{-1}(\mathbf{b}[0]) = \mathbf{b}[0]$. On the other hand, if $f(\mathbf{a}[0]) \neq \mathbf{a}[0]$, then from (3) we know that $\mathbf{b} \notin A_{C^{-1}}$, and again C^{-1} must map $\mathbf{b}[0]$ to itself.

Conversely, assume that C has an inverse FDM CA D with neighbourhood N' and let A_D be the active set of D. Since D must correctly 'map back' all states where C applied the function f, it is clear that X (as defined in (2)) is a subset of A_D.

It remains to show that (3) holds. For the sake of contradiction assume that $\mathbf{b} = \bar{\mathbf{h}}_C(\bar{\mathbf{r}}, N') \in X$, where $\bar{\mathbf{r}}$ is an (N, N')-neighbourhood of a neighbourhood $\mathbf{a} \notin A_C$, and $f(\mathbf{a}[0]) \neq \mathbf{a}[0]$. Since $\mathbf{a} \notin A_C$, we know that $\mathbf{a}[0] = \mathbf{b}[0]$. Since D is an inverse of C, the function used by D must be f^{-1}. Since $\mathbf{b} \in X \subseteq A_D$, the FDM CA D applies the function f^{-1} to $\mathbf{b}[0]$, but the result cannot be $\mathbf{a}[0]$ since that would imply $f(\mathbf{a}[0]) = \mathbf{b}[0] = \mathbf{a}[0]$, a contradiction. □

The following corollary addresses the uniqueness of FDM CA inverses.

Corollary 1. *Let C and X be as defined in Proposition 3, and let C have some inverse C^{-1} with neighbourhood N'. Then C^{-1} is the only reduced FDM CA with neighbourhood N' that inverts C.*

Proof. Any inverse of C must have function f^{-1}. Assume for the sake of contradiction that there exists an FDM CA D with neighbourhood N' that inverts C and has acting set $A_D \neq X$. In the proof of Proposition 3 we have observed that X must be a subset of A_D. Thus it is sufficient to show that there cannot be an element $\mathbf{b} \in A_D$ with $\mathbf{b} \notin X$.

Let $\mathbf{b} = \bar{\mathbf{h}}_C(\bar{\mathbf{r}}, N')$, where $\bar{\mathbf{r}} \in \bar{\mathbf{S}}_{(N,N')}(\mathbf{a})$, $\mathbf{a} \notin A_C$ since $\mathbf{b} \notin X$. Then D cannot be the inverse of C because C maps $\mathbf{a}[0]$ to itself, but D maps $\mathbf{b}[0] = \mathbf{a}[0]$ to $f^{-1}(\mathbf{b}[0])$, which cannot be equal to $\mathbf{b}[0]$ since $\mathbf{b} \in A_D$ and D is reduced. □

4 A Public-Key Cryptosystem

We want to use the idea of composing together many simple RCAs to form a complex RCA that is hard to invert, as outlined in the paper by Kari [11]. In order to make this idea work, we need to have some way to randomly generate a sequence of simple CAs such that the neighbourhood size of their composition remains small (or constant), and each CA in the composition is reversible.

We will demand that the neighbourhood size of each cellular automaton in the composition is the same, and that the entire composition has the same neighbourhood as any of the components. The components will all be FDM CAs. Since the neighbourhood, state set, and dimension are fixed, we must design an algorithm which generates acting sets and transition functions for each of the n components $C_1, C_2, \ldots, C_n$. From the theory in the previous section, we can now state some requirements for such an algorithm.

To maintain neighbourhood size during composition, the FDM CA C_j must have an acting set A_j and transition function f_j such that the composition $C_j \circ (C_{j-1} \circ C_{j-2} \circ \cdots \circ C_1)$ has the same neighbourhood, for all $j \in \{2, \ldots, n\}$. Referring to the condition from Proposition 1, we need to guarantee that for each neighbourhood, the next neighbourhood set of $C_{j-1} \circ C_{j-2} \circ \cdots \circ C_1$ is either completely contained in A_j or is disjoint from A_j. Denote by $T \subseteq S$ the "change set", that is, the set of all states that the composition $C_{j-1} \circ C_{j-2} \cdots \circ C_1$ can possibly change. One way we can be sure to retain neighbourhood size during composition is by setting A_j equal to the set of all neighbourhoods which contain a state in T. The condition from Proposition 1 is satisfied since all neighbourhoods containing states in T will certainly be mapped (by $C_{j-1} \circ C_{j-2} \circ \cdots \circ C_1$) to neighbourhoods which also contain states in T (assuming $f_1, f_2, \ldots, f_{j-1}$ are one-to-one mappings), and neighbourhoods which do not contain any states in T will clearly be mapped to neighbourhoods which do not contain any states in T. We use a less restricted version of this principle (which still satisfies the neighbourhood size preservation condition) in our algorithm to determine the acting set of each FDM CA in a composition.

The need for each of the FDM CAs in the composition to be invertible puts additional restrictions on their form. In order to be sure that the FDM CA is invertible, the set T which is used to find the acting set of each FDM CA must contain all states that the function f can change. The functions $f_1, f_2, \ldots, f_n$ must also be permutations (one-to-one mappings). We discuss the key generation algorithm in more detail in Section 4.1.

Once the component FDM CAs are generated, the public key is determined by sequentially applying $C_1, C_2, \ldots, C_n$ to each possible neighbourhood (using the neighbourhood as the starting configuration). The final state of the cell is recorded, and the public key is this mapping of neighbourhoods to states. The private key is not calculated explicitly; the CAs $C_1^{-1}, C_2^{-1}, \ldots, C_n^{-1}$ are simply applied sequentially for decryption. The message is encoded in a d dimensional grid and is evolved for a fixed number of iterations of the public key to produce the ciphertext. The ciphertext and number of iterations are sent as the encrypted message.

4.1 The Key Generation Algorithm

Our key generation scheme is given in Algorithm 1. We should note that in this algorithm, the `random_element` function returns a random element from a given set, the `random` function returns a floating point number between 0.0 and 1.0, the `random_permutation` function returns a random permutation mapping of a

given set, and the `random_binary` function returns a random binary string of a given length. Also, the `get_all_possible_neighbourhoods` function returns all possible neighbourhoods given a state set S and neighbourhood N.

Input: State space S, Neighbourhood N, Number of FDM CA n, $0 < p, q < 1$
Output: Set of reversible FDM CAs $C_1, C_2, \ldots, C_n$

Initialization
$\mathsf{T} \longleftarrow \emptyset$
T.add (`random_element` (S))
T.add (`random_element` $(S-\mathsf{T})$)
all_possible_neighbourhoods $\longleftarrow$ `get_all_possible_neighbourhoods` (S,N)

for $i \leftarrow 1$ **to** n **do**
  The following code determines f_i
  if `random` $() <$ p *and* $\mathsf{T} \neq S$ **then**
    T.add (`random_element` $(S-\mathsf{T})$)
  end
  $\mathsf{f}_i \longleftarrow$ `random_permutation` (T)

  The following code determines A_i
  binary_string $\longleftarrow$ `random_binary` $(|N|)$
  $\mathsf{A}_i \longleftarrow \emptyset$
  for neighbourhood $\in$ all_possible_neighbourhoods **do**
    unchanging_neighbourhood $\longleftarrow$ *True*
    for $j \leftarrow 1$ **to** $|N|$ **do**
      if neighbourhood $[j] \in \mathsf{T}$ **then**
        unchanging_neighbourhood $\longleftarrow$ *False*
        if binary_string $[j] = 1$ **then**
          A_i.add (neighbourhood)
          break
        end
      end
    end
    if unchanging_neighbourhood $=$ *True and* `random` $() <$ q **then**
      A_i.add (neighbourhood)
    end
  end
  $\mathsf{C}_i \longleftarrow$ $\{ S,N,\mathsf{A}_i,\mathsf{f}_i \}$
end

Algorithm 1. The public-key generation algorithm, discussed in Section 4.1

Initially T is a set $T \subseteq S$ of two random elements of S. The FDM CAs in the composition are then constructed in order from C_1 to C_n. Before choosing each C_i, with probability p a new element of S is added to the set T, and otherwise T stays the same. The function f_i is chosen for each FDM CA to be a random permutation of the set T, and f_i applies the identity map to states in $S - T$.

The only remaining task is to select the acting set. For each FDM CA in the composition, a random binary string of length $|N|$ is chosen. Every possible

neighbourhood is then considered as a candidate element of the acting set. If the candidate neighbourhood has a state which is an element of T and is also in a position corresponding to a '1' of the binary string, then it is added to the acting set. Also, if the neighbourhood contains only states which are not in T, then the neighbourhood is added to the acting set with probability q. For example, consider the case where $S = \{a, b, c\}$, $N = \{-1, 1\}$, $T = \{a, b\}$, and the random binary string is 01. Then the neighbourhood ca is a member of the acting set while ac is not, and cc will become a member of the acting set with probability q. Note that if the neighbourhood N contains the zero element, then clearly the case where the neighbourhood is added to the acting set with probability q is irrelevant since not even the state of the cell can change.

We now discuss the correctness of this algorithm, and begin by showing that the condition for constant neighbourhood size during composition holds. Assume we are attempting to determine the acting set of the i^{th} FDM CA in the composition, A_i, and let us first consider neighbourhoods which have at least one state in T. If a neighbourhood is in A_i, then at least one element of T occurring in the neighborhood corresponds to a '1' in the binary string. Since $C_i \circ C_{i-1} \circ \cdots \circ C_1$ is T-invariant (states in T are mapped to states in T), the neighbourhood will certainly be mapped to a neighbourhood in A_i. On the other hand, if a neighbourhood is not in A_i, then all occurrences of states in T correspond to '0' elements of the binary string. This neighbourhood is mapped to a neighbourhood where states in T also correspond to '0' elements of the binary string, and hence it is mapped to a neighbourhood which is not in A_i. Finally, a neighbourhood which contains no elements of T satisfies the condition of Proposition 1 since it must map to itself.

It remains to show that the condition for FDM CA reversibility holds for each C_i. Rather conveniently, the previous conditions actually allow (or demand) that A_i is also the acting set of the inverse FDM CA. Since any addition to the set T during the construction of each FDM CA happens before we choose A_i, we are guaranteed that elements of A_i will be mapped to elements of A_i, and elements not in A_i will not be mapped to A_i. So the condition from Proposition 3 also holds. We could not be sure of this if A_i was constructed with some T that did not correspond with the states that f_i changes.

Note that Algorithm 1 has running time exponential in neighbourhood size.

4.2 Security Concerns and Practical Considerations

Since the FDM CA compositions follow a specific form and are not general two-dimensional RCA, we cannot directly use Kari's result [13] to justify the security of the system, and hence the security of this cryptosystem is largely unknown to us. However, we do not believe that straightforward brute force attacks will work. If one attempted to guess at a composition of FDM CAs which resulted in the same public key, there are many choices for each CA and there are $n!$ ways to arrange them, since n is the number of CAs in the composition. One could also attempt to keep track of all global inputs and outputs for a fixed grid size in order to invert the composed CA. In this case the number of possible global

configurations is $|S|^{(g)}$ where g is the number of grid cells, so as long as the grid (the message) is relatively large this method will not work.

We also do not believe that the inverse $C_1^{-1} \circ C_2^{-1} \circ \cdots \circ C_n^{-1}$ can be guessed very easily. Although we do not calculate it explicitly, this CA must have a fairly large neighbourhood because for each composition in the sequence, the condition from Proposition 1 does not hold in general. Each time T changes during the generation of the FDM CA, the inverse automaton's neighbourhood size may increase, and this can happen at most $|S| - 2$ times. So there is a computable upper bound for the neighbourhood size of the inverse, given $C_1, C_2, \ldots, C_n$, but for reasonably large S and $d > 1$ this probably does not pose a security threat.

A user must choose the parameters of our algorithm with some care in order to prevent these brute force attacks and also to be able to encrypt and decrypt within a reasonable amount of time on a normal computer. An example is $N = \{(0, 1), (1, 0)\}$ (the top and right neighbours), $|S| \approx 25$, grid size $g \approx 500$, number of FDM CA in the composition $n \approx 100$, $p = q = 0.5$, and number of iterations ≈ 100. These sizes can probably be increased significantly if the algorithm were implemented on specialized parallel hardware (especially the grid size and number of iterations). We should note that the expected number of CA needed in the composition to just achieve $T = S$ is $(|S|-2) \cdot 1/p$, and so n should be chosen so that it is significantly larger than this quantity. If n is too small, then the composition will only change states in T, and all elements in $S-T$ that occur in the original message will occur in the same places in the ciphertext.

One security issue related to the last point is that with our key generation algorithm as written, it is very easy for an attacker to determine which state was last added to T. The public key will map this state to some other state regardless of the neighbourhood. Not much can be immediately done with this information, but perhaps it could be a starting point for a clever cryptanalytic algorithm to find each of the FDM CA in the composition in backwards order.

5 Conclusion and Future Work

We presented conditions which guarantee that compositions of fixed-domain marker cellular automata have the same neighbourhood as each of the individual components. We showed that, under certain technical assumptions, an FDM CA has a unique inverse with a given neighbourhood. We used these results to design, present, and show the correctness of a working key generation algorithm for a public-key cryptosystem originally conceived by Kari [11]. We also provided some preliminary cryptanalysis and gave some practical implementation notes.

This work provides several avenues for further research. We have given perhaps a more manageable definition of marker cellular automata, which could facilitate or help with additional theoretical development in related areas. The security of the cryptosystem presented in this work is currently unknown, and serious cryptanalysis is needed before more can be said in this regard. When generating the public key, there may be some alternate or more general way to choose the component CAs to produce a more efficient or secure system. If the cryptosystem

does not break easily then it would make sense to try to design an optimal hardware implementation and to do a corresponding feasibility analysis for real-world applications.

References

1. Tao, R., Chen, S.: On finite automaton public-key cryptosystem. Theoretical Computer Science 226(1-2), 143–172 (1999)
2. Charbouillot, S., Perez, A., Fronte, D.: A programmable hardware cellular automaton: Example of data flow transformation. In: 13th IEEE International Conference on Electronics, Circuits and Systems, pp. 1232–1235 (2006)
3. Franti, E., Slav, C., Balan, T., Dascalu, M.: Design of cellular automata hardware for cryptographic applications. In: CAS 2004 Int. Semiconductor Conference, vol. 2, pp. 463–466 (2004)
4. Zheng, Y., Imai, H.: A cellular automaton based fast one-way hash function suitable for hardware implementation. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, pp. 217–234. Springer, Heidelberg (1998)
5. Anghelescu, P., Ionita, S., Sofron, E.: Block encryption using hybrid additive cellular automata. In: HIS 2007: Proceedings of the 7th International Conference on Hybrid Intelligent Systems, Washington, DC, USA, pp. 132–137. IEEE Computer Society, Los Alamitos (2007)
6. Gutowitz, H.: Cryptography with dynamical systems. In: Goles, E., Boccara, N. (eds.) Cellular Automata and Cooperative Phenomena, pp. 237–274 (1993)
7. Gutowitz, H.: Method and apparatus for encryption, decryption, and authentication using dynamical systems. U.S. Patent 5365589 (1994)
8. Seredynski, M., Bouvry, P.: Block cipher based on reversible cellular automata. Congress on Evolutionary Computation 2, 2138–2143 (2004)
9. Srebrny, M., Such, P.: Encryption using two-dimensional cellular automata with applications. Artificial intelligence and Security in Computing Systems, 203–215 (2003)
10. Wolfram, S.: Random sequence generation by cellular automata. Advances in Applied Mathematics 7(2), 163–169 (1986)
11. Kari, J.: Cryptosystems based on reversible cellular automata (manuscript, 1992)
12. Amoroso, S., Patt, Y.: Decision procedures for surjectivity and injectivity of parallel maps for tesselation structures. J. Comput. System Sci. 6, 448–464 (1972)
13. Kari, J.: Reversibility and surjectivity problems of cellular automata. J. Comput. System Sci. 48, 149–182 (1994)
14. Ashley, J.: Marker automorphisms of the one-sided d-shift. Ergodic Theory Dynam. Systems 10(2), 247–262 (1990)
15. Clarridge, A., Salomaa, K.: A cryptosystem based on the composition of reversible cellular automata. Technical Report 2008-549, Queen's University, School of Computing (2008)

Grammars Controlled by Special Petri Nets

Jürgen Dassow[1] and Sherzod Turaev[2]

[1] Otto-von-Guericke-Universität Magdeburg
PSF 4120, D-39016 Magdeburg, Germany
dassow@iws.cs.uni-magdeburg.de
[2] GRLMC, Universitat Rovira i Virgili
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
sherzod.turaev@urv.cat

Abstract. A Petri net controlled grammar is a context-free grammar with a control by a Petri net whose transitions are labeled with rules of the grammar or the empty string and the associated language consists of all terminal strings which can be derived in the grammar and the sequence of rules in a derivation is in the image of a successful occurrence of transitions of the net. We present some results on the generative capacities of such grammars that Petri nets are restricted to some known structural subclasses of Petri nets.

1 Introduction

A Petri net controlled grammar is a context-free grammar with a control by a Petri net whose transitions are labeled with rules of the grammar or the empty string and the associated language consists of all terminal strings which can be derived in the grammar and the sequence of rules in a derivation is in the image of a successful occurrence of transitions of the net. In [1] we investigated arbitrary Petri net controlled grammars in dependence on the type of labeling (a bijection, coding and weak coding) and on the use of final markings (a finite set of final markings and the set of all reachable markings).

In this paper we investigate grammars controlled by some special subclasses of Petri nets, which corresponds to investigations in grammars with controlled derivations where arbitrary regular sets are substituted by some special subregular sets, e.g. in case of regularly controlled grammars and tree controlled grammars, see [2] and [3].

The paper is organized as follows. In Section 2 we give some notions and definitions from the theories of formal languages and Petri nets. In Section 3 we introduce the concept of a control of derivations in context-free grammars by special Petri nets. In Section 4 we investigate the effect of labeling on the power of the introduced families of languages. In Section 5 we discuss the effect of different types of final markings on the generative power and give some characterizations by other regulated grammars.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 326–337, 2009.

2 Definitions

Throughout the paper, we assume that the reader is familiar with the basic concepts of the theories of formal languages and Petri nets; for details we refer to [4,5,6,7].

Let Σ be an alphabet. A *string* over Σ is a sequence of symbols from the alphabet. The set of all strings over the alphabet Σ is denoted by Σ^*. A subset L of Σ^* is called a *language*. A string $u = u_1u_2\cdots u_n,\ u_1, u_2, \ldots, u_n \in \Sigma$ is a *scattered substring* of $v \in \Sigma^*$ if $v = v_1u_1v_2\cdots u_nv_{n+1}$ for some $v_1, v_2, \ldots, v_{n+1} \in \Sigma^*$ with $n \geq 1$. The *length* of a string w is denoted by $|w|$, and the number of occurrences of a symbol a in a string w by $|w|_a$. The *empty* string is denoted by λ which is of length 0.

A string $u_1v_1u_2v_2\cdots u_nv_n$ is called a *shuffle* of strings $u = u_1u_2\cdots u_n$ and $v = v_1v_2\cdots v_n$ where $u_i, v_i \in \Sigma^*$, $1 \leq i \leq n$. A shuffle of u and v is *proper* if it is not a concatenation of u and v or v and u. A shuffle of $u_1, u_2, \ldots, u_n$, $n \geq 3$ is a shuffle of a shuffle of $u_1, u_2, \ldots, u_{n-1}$ and u_n. A shuffle of $u_1, u_2, \ldots, u_n$, $n \geq 2$, is *semi* if for some $1 \leq i < j \leq n$, $u_i = u_j$, then a proper shuffle of u_i and u_j is not its scattered substring, in words, a semi-shuffle string does not contain self-shuffled scattered substrings.

A *context-free grammar* is a quadruple $G = (V, \Sigma, S, R)$ where V and Σ are the disjoint finite sets of *nonterminal* and *terminal* symbols, respectively, $S \in V$ is the *start* symbol and the finite set $R \subset V \times (V \cup \Sigma)^*$ is the set of *(production) rules*. Usually, a rule (A, x) is written as $A \to x$. A rule of the form $A \to \lambda$ is called an *erasing rule*. The word $x \in (V \cup \Sigma)^+$ *directly derives* $y \in (V \cup \Sigma)^*$, written as $x \Rightarrow y$, iff there is a rule $r = A \to \alpha \in R$ such that $x = x_1Ax_2$ and $y = x_1\alpha x_2$. The reflexive and transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. A derivation using the sequence of rules $\pi = r_1r_2\cdots r_n$ is denoted by $\overset{\pi}{\Rightarrow}$ or $\overset{r_1r_2\cdots r_n}{\Longrightarrow}$. The *language* generated by G is defined by $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

A *regularly controlled grammar* is a quintuple $G = (V, \Sigma, S, R, K)$ where V, Σ, S, R are specified as in a context-free grammar and K is a regular set over R. The language generated by G consists of all words $w \in \Sigma^*$ such that there is a derivation $S \overset{r_1r_2\cdots r_n}{\Longrightarrow} w$ where $r_1r_2\cdots r_n \in K$.

A *matrix grammar* is a quadruple $G = (V, \Sigma, S, M)$ where V, Σ, S are defined as for a context-free grammar, M is a finite set of *matrices* which are finite strings over a set of context-free rules. The language generated by the grammar G consists of all strings $w \in \Sigma^*$ such that there is a derivation $S \overset{r_1r_2\cdots r_n}{\Longrightarrow} w$ where $r_1r_2\cdots r_n$ is a concatenation of some matrices $m_{i_1}, m_{i_2}, \ldots, m_{i_k} \in M$.

A *vector grammar* is a quadruple $G = (V, \Sigma, S, M)$ whose components are defined as for a matrix grammar. The language generated by the grammar G consists of all strings $w \in \Sigma^*$ such that there is a derivation $S \overset{r_1r_2\cdots r_n}{\Longrightarrow} w$ where $r_1r_2\cdots r_n$ is a shuffle of some matrices $m_{i_1}, m_{i_2}, \ldots, m_{i_k} \in M$.

A *semi-matrix grammar* is a quadruple $G = (V, \Sigma, S, M)$ whose components are defined as for a matrix grammar. The language generated by the grammar G consists of all strings $w \in \Sigma^*$ such that there is a derivation $S \overset{r_1r_2\cdots r_n}{\Longrightarrow} w$ where $r_1r_2\cdots r_n$ is a semi-shuffle of some matrices $m_{i_1}, m_{i_2}, \ldots, m_{i_k} \in M$.

A matrix (semi-matrix, vector) grammar G is called *without repetitions*, if each rule r occurs in $M = \{m_1, m_2, \dots, m_n\}$ only once, i.e., $|m_1 m_2 \cdots m_n|_r = 1$. For each matrix grammar, by adding chain rules, one can construct an equivalent matrix grammar without repetitions.

The families of languages generated by regularly controlled, matrix, vector and semi-matrix grammars (with erasing rules) are denoted by $\mathbf{rC}$, $\mathbf{MAT}$, $\mathbf{V}$ and $\mathbf{sMAT}$ ($\mathbf{rC}^\lambda$, $\mathbf{MAT}^\lambda$, $\mathbf{V}^\lambda$ and $\mathbf{sMAT}^\lambda$), respectively. It is known that $\mathbf{rC} = \mathbf{MAT} \subseteq \mathbf{V}$ and $\mathbf{rC}^\lambda = \mathbf{MAT}^\lambda = \mathbf{V}^\lambda = \mathbf{sMAT}^\lambda$ (see [8], [9]).

A *Petri net* (PN) is a construct $N = (P, T, F, \varphi)$ where P and T are disjoint finite sets of *places* and *transitions*, respectively, $F \subseteq (P \times T) \cup (T \times P)$ is the set of *directed arcs*, $\varphi : F \to \{1, 2, \dots\}$ is a *weight function*.

A Petri net can be represented by a bipartite directed graph with the node set $P \cup T$ where places are drawn as *circles*, transitions as *boxes* and arcs as *arrows* with labels $\varphi(p,t)$ or $\varphi(t,p)$. If $\varphi(p,t) = 1$ or $\varphi(t,p) = 1$, the label is omitted.

A mapping $\mu : P \to \{0, 1, 2, \dots\}$ is called a *marking*. For each place $p \in P$, $\mu(p)$ gives the number of *tokens* in p. Graphically, tokens are drawn as small solid *dots* inside circles. ${}^\bullet x = \{y \mid (y,x) \in F\}$ and $x^\bullet = \{y \mid (x,y) \in F\}$ are called *pre-* and *post-sets* of $x \in P \cup T$, respectively. For $X \subseteq P \cup T$, define ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$. The elements of ${}^\bullet t$ (${}^\bullet p$) are called *input* places (transitions) and the elements of $t^\bullet$ ($p^\bullet$) are called *output* places (transitions) of t (p).

A sequence of places and transitions $\rho = x_1 x_2 \cdots x_n$ is called a *path* if and only if no place or transition except x_1 and x_n appears more than once, and $x_{i+1} \in x_i^\bullet$ for all $1 \le i \le n-1$. A path $\rho = x_1 x_2 \cdots x_n$ is a *cycle* if $x_1 = x_n$. By P_ρ, T_ρ, F_ρ the sets of places, transitions, and arcs of a path ρ are denoted. The sequence of transitions in a path ρ is denoted by $tr(\rho)$.

A transition $t \in T$ is *enabled* by marking μ iff $\mu(p) \ge \varphi(p,t)$ for all $p \in P$. In this case t can *occur* (*fire*). Its occurrence transforms the marking μ into the marking μ' defined for each place $p \in P$ by $\mu'(p) = \mu(p) - \varphi(p,t) + \varphi(t,p)$. A finite sequence $t_1 t_2 \cdots t_k$ of transitions is called *an occurrence sequence* enabled at a marking μ if there are markings $\mu_1, \mu_2, \dots, \mu_k$ such that $\mu \xrightarrow{t_1} \mu_1 \xrightarrow{t_2} \dots \xrightarrow{t_k} \mu_k$. In short this sequence can be written as $\mu \xrightarrow{t_1 t_2 \cdots t_k} \mu_k$ or $\mu \xrightarrow{\nu} \mu_k$ where $\nu = t_1 t_2 \cdots t_k$. For each $1 \le i \le k$, the marking μ_i is called *reachable* from the marking μ.

A *marked* Petri net is a system $N = (P, T, F, \varphi, \iota)$ where (P, T, F, φ) is a Petri net, ι is the *initial marking*. Let M be a set of markings, which will be called *final* markings. An occurrence sequence ν of transitions is called *successful* for M if it is enabled at the initial marking ι and finished at a final marking τ of M.

An *ordinary net* (ON) is a Petri net $N = (P, T, F, \varphi, \iota)$ where $\varphi(x,y) = 1$ for all $(x,y) \in F$. We omit φ from the definition of an ordinary net.

We regard the following main structural subclasses of Petri nets.

A *state machine* (SM) is an ordinary Petri net such that $|{}^\bullet t| = |t^\bullet| = 1$ for all $t \in T$.

A *marked graph* (MG) is an ordinary Petri net such that $|{}^\bullet p| = |p^\bullet| = 1$ for all $p \in P$.

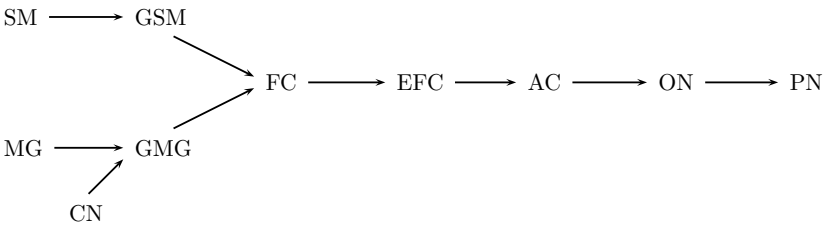


Fig. 1. The hierarchy of Petri net classes

A *generalized state machine* (GSM) is an ordinary Petri net such that $|{}^\bullet t| \leq 1$ and $|t^\bullet| \leq 1$ for all $t \in T$.

A *generalized marked graph* (GMG) is an ordinary Petri net such that $|{}^\bullet p| \leq 1$ and $|p^\bullet| \leq 1$ for all $p \in P$.

A *casual net* (CN) is a generalized marked graph each subgraph of which is not a cycle.

A *free-choice* net (FC) is an ordinary Petri net such that if $p_1^\bullet \cap p_2^\bullet \neq \emptyset$ then $|p_1^\bullet| = |p_2^\bullet| = 1$ for all $p_1, p_2 \in P$.

An *extended free-choice* net (EFC) is an ordinary Petri net such that if $p_1^\bullet \cap p_2^\bullet \neq \emptyset$ then $p_1^\bullet = p_2^\bullet$ for all $p_1, p_2 \in P$.

An *asymmetric choice net* (AC) is an ordinary Petri net such that if $p_1^\bullet \cap p_2^\bullet \neq \emptyset$ then $p_1^\bullet \subseteq p_2^\bullet$ or $p_1^\bullet \supseteq p_2^\bullet$ for all $p_1, p_2 \in P$.

The hierarchy of the introduced subclasses of Petri nets is shown in Fig. 1 where the arrows denote proper inclusions of the left families into the right families.

3 Grammars and Their Languages

We introduce the concept of control by special Petri nets.

Definition 1. *A Petri net controlled grammar (in short a PN controlled grammar) is a tuple $G = (V, \Sigma, S, R, N, \gamma, M)$ where V, Σ, S, R are defined as for a context-free grammar, $N = (P, T, F, \iota)$ is a Petri net, $\gamma : T \to R \cup \{\lambda\}$ is a transition labeling function and M is a set of final markings.*

If a Petri net is a (generalized) state machine, (generalized) marked graph, causal net, (extended) free-choice net, asymmetric choice net or ordinary net, then we call a *(generalized) state machine*, *(generalized) marked graph*, *causal net*, *(extended) free-choice net*, *asymmetric choice net* or *ordinary net* controlled grammar, respectively. We also use the common name a *special Petri net* (sPN) when we refer to each sub-class.

Definition 2. *The language generated by a PN controlled grammar G consists of all strings $w \in \Sigma^*$ such that there is a derivation $S \stackrel{\pi}{\Rightarrow} w \in \Sigma^*$, $\pi = r_1 r_2 \cdots r_k$, and a successful occurrence sequence $\nu = t_1 t_2 \cdots t_s$ for M such that $\pi = \gamma(\nu)$.*

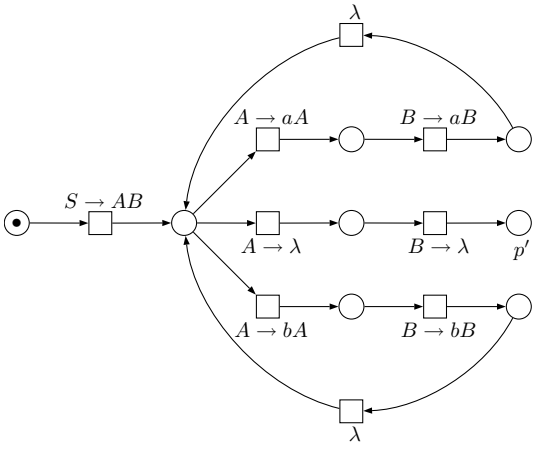


Fig. 2. A state machine N_1

(This definition uses the extended form of the transition labeling function $\gamma : T^* \to R^*$; this extension is done in the usual manner.)

Different labeling strategies and different definitions of the set of final markings result various types of Petri net controlled grammars. In this paper we consider the following types of Petri net controlled grammars.

A Petri net controlled grammar is called *free* (abbreviated by f), λ-*free* (abbreviated by $-\lambda$) or *extended* (abbreviated by λ) if γ is a bijection (the labels of the transitions are distinct and non-null), a coding (different transitions may be labeled with the same symbol) or a week coding (transitions may be labeled with λ), respectively.

A Petri net controlled grammar is of *r-type*, *t-type* or *g-type* if M is the set of all reachable markings, the finite set of markings or a set of any markings which is greater or equal to an element of a given finite set M_0 of markings, respectively.

We use the notation a (x, y)-*PN controlled grammar* where $x \in \{f, -\lambda, \lambda\}$ shows the type of a labeling function and $y \in \{r, t, g\}$ shows the type of a set of final markings.

We denote the families of languages generated by grammars controlled by state machines, generalized state machines, marked graphs, generalize marked graphs, causal nets, free-choice nets, extended free-choice nets, asymmetric nets, ordinary nets and petri nets (with erasing rules) by $\mathbf{SM}(x, y)$, $\mathbf{GSM}(x, y)$, $\mathbf{MG}(x, y)$, $\mathbf{GMG}(x, y)$, $\mathbf{CN}(x, y)$, $\mathbf{FC}(x, y)$, $\mathbf{EFC}(x, y)$, $\mathbf{AC}(x, y)$, $\mathbf{ON}(x, y)$, $\mathbf{PN}(x, y)$ ($\mathbf{SM}^\lambda(x, y)$, $\mathbf{GSM}^\lambda(x, y)$, $\mathbf{MG}^\lambda(x, y)$, $\mathbf{GMG}^\lambda(x, y)$, $\mathbf{CN}^\lambda(x, y)$, $\mathbf{FC}^\lambda(x, y)$, $\mathbf{EFC}^\lambda(x, y)$, $\mathbf{AC}^\lambda(x, y)$, $\mathbf{ON}^\lambda(x, y)$, $\mathbf{PN}^\lambda(x, y)$), respectively, where $x \in \{f, -\lambda, \lambda\}$ and $y \in \{r, t, g\}$.

We use bracket notation $\mathbf{Y}^{[\lambda]}$ for a language family $\mathbf{Y}$ in order to say that a statement holds both in case of with erasing rules and in case of without erasing rules.

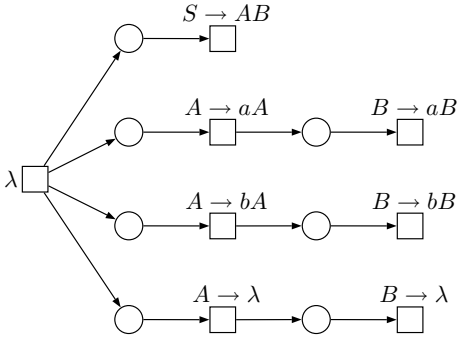


Fig. 3. A marked graph N_2

For $x \in \{f, -\lambda, \lambda\}$ and $y \in \{r, t, g\}$, the inclusion $\mathbf{Y}(x, y) \subseteq \mathbf{Y}^{\lambda}(x, y)$ is obvious where $\mathbf{Y} \in \{\mathbf{SM}, \mathbf{GSM}, \mathbf{MG}, \mathbf{GMG}, \mathbf{CN}, \mathbf{FC}, \mathbf{EFC}, \mathbf{AC}, \mathbf{ON}\}$.

Example 1. Let $G_1 = (\{S, A, B\}, \{a, b\}, S, R, N_1, \gamma_1, M_1)$ be a state machine controlled grammar where $R = \{S \to AB,\ A \to aA|bA|\lambda,\ B \to aB|bB|\lambda\}$, N_1 which is illustrated in Fig. 2 is a state machine and $M_1 = \{\mu\}$ where $\mu(p') = 1$ and $\mu(p) = 0$ for all $p \in P - \{p'\}$, then $L(G_1) = \{ww \mid w \in \{a, b\}^*\}$.

Example 2. Let $G_2 = (\{S, A, B\}, \{a, b\}, S, R, N_2, \gamma_2, M_2)$ be a marked graph controlled grammar where R is the same as in Example 1, a marked graph N_2 is illustrated in Fig. 3 and $M_1 = \{\mu\}$ where $\mu(p) = 0$ for all $p \in P$. Then $L(G_2) = \{ww' \mid w \in \{a, b\}^*$ and $w' \in \mathrm{Perm}(w)\}$.

4 Results: Labeling Strategies

In this section we investigate the effect of the labeling of transitions on the generative capacities of the introduced families of languages.

From the definition, the next statement follows immediately.

Lemma 1. *For* $\mathbf{Y} \in \{\mathbf{SM}, \mathbf{GSM}, \mathbf{MG}, \mathbf{GMG}, \mathbf{CN}, \mathbf{FC}, \mathbf{EFC}, \mathbf{AC}, \mathbf{ON}\}$ *and* $y \in \{r, t, g\}$,

$$\mathbf{Y}^{[\lambda]}(f, y) \subseteq \mathbf{Y}^{[\lambda]}(-\lambda, y) \subseteq \mathbf{Y}^{[\lambda]}(\lambda, y).$$

Further, we show that the reverse inclusions also hold.

For each sPN, one can easily construct a net of the same type in which the transitions have different labels, by "splitting" each transition into two, i.e., by replacing a transition t with label $A \to \alpha$ by new transitions t', t'' with labels $A \to A'$, $A' \to \alpha$, respectively, where t' receives all incoming arcs of t and t'' receives all outgoing arcs of t, and a new place p_t from transition t' and to transition t''.

Lemma 2. *For* $\mathbf{Y} \in \{\mathbf{SM}, \mathbf{GSM}, \mathbf{MG}, \mathbf{GMG}, \mathbf{CN}, \mathbf{FC}, \mathbf{EFC}, \mathbf{AC}, \mathbf{ON}\}$ *and* $y \in \{r, t, g\}$,

$$\mathbf{Y}^{[\lambda]}(-\lambda, y) \subseteq \mathbf{Y}^{[\lambda]}(f, y).$$

For each (λ, y)-sPN controlled grammar, if we label each λ-transition with $X \to X$, start each derivation with $S' \to SX$ and erase X with rule $X \to \lambda$ at the end of the derivation, then we get the same derivation in a $(-\lambda, y)$-sPN controlled grammar, i.e.,

Lemma 3. *For* $\mathbf{Y} \in \{\mathbf{SM}, \mathbf{GSM}, \mathbf{MG}, \mathbf{GMG}, \mathbf{CN}, \mathbf{FC}, \mathbf{EFC}, \mathbf{AC}, \mathbf{ON}\}$ *and* $y \in \{r, t, g\}$,

$$\mathbf{Y}^{\lambda}(\lambda, y) \subseteq \mathbf{Y}^{\lambda}(-\lambda, y).$$

It is known that by structure state machines and finite automata are the same. We can remove λ-transitions of a state machine by using an analogous method to λ-transition removal operation in finite automata.

Lemma 4. *For* $y \in \{r, t, g\}$ *and* $\mathbf{Y} \in \{\mathbf{SM}, \mathbf{GSM}\}$, $\mathbf{Y}(\lambda, y) \subseteq \mathbf{Y}(-\lambda, y)$.

Lemma 5. *For* $x \in \{f, -\lambda, \lambda\}$ *and* $y \in \{r, t, g\}$, $\mathbf{GMG}^{[\lambda]}(x, y) \subseteq \mathbf{MG}^{[\lambda]}(x, y)$.

Proof. Let $G = (V, \Sigma, S, R, N, \gamma, M)$ be a (x, y)-GMG controlled grammar (with or without erasing rules) where $N = (P, T, F, \iota)$ is a generalized marked graph. Let $P_{\emptyset}^{-} = \{p \in P \mid {}^{\bullet}p = \emptyset\}$ and $P_{\emptyset}^{+} = \{p \in P \mid p^{\bullet} = \emptyset\}$. Without loss of generality we can assume that $P_{\emptyset}^{-} \cap P_{\emptyset}^{+} = \emptyset$ (if place $p \in P$ is isolated, i.e., $|{}^{\bullet}p| = |p^{\bullet}| = 0$, it can be eliminated since isolated places do not effect any derivation of the grammar).

Let $Q^{-} = \{q_p \mid p \in P_{\emptyset}^{-}\}$ and $Q^{+} = \{q_p \mid p \in P_{\emptyset}^{+}\}$ be the sets of new places, $T^{-} = \{t_p \mid p \in P_{\emptyset}^{-}\}$ and $T^{+} = \{t_p \mid p \in P_{\emptyset}^{+}\}$ be the sets of new transitions and $F^{-} = \{(t_p, q_p), (q_p, t_p), (t_p, p) \mid p \in P_{\emptyset}^{-}\}$ and $F^{+} = \{(p, t_p), (t_p, q_p), (q_p, t_p) \mid p \in P_{\emptyset}^{+}\}$ be the sets of new arcs.

We construct a MG $N' = (P \cup Q^{-} \cup Q^{+}, T \cup T^{-} \cup T^{+}, F \cup F^{-} \cup F^{+}, \iota')$ where $\iota'(p) = \iota(p)$ if $p \in P$ and $\iota'(p) = 0$ if $p \in Q^{-} \cup Q^{+}$.

We set $V' = V \cup \{B\}$ and $R' = R \cup \{B \to B\}$ where B is a new nonterminal symbol, and define a MG controlled grammar $G' = (V', \Sigma, S, R', N', \gamma', M')$ where the labeling function γ' is defined by $\gamma'(t) = \gamma(t)$ if $t \in T$ and $\gamma'(t) = B \to B$ if $t \in T^{-} \cup T^{+}$. For each $\tau' \in M'$, $\tau'(p) = \tau(p)$ if $p \in P$ and $\tau'(p) = 0$ if $p \in Q^{-} \cup Q^{+}$.

By construction of N', any transition $t \in T^{-} \cup T^{+}$ never occurs and the production rule $B \to B$ is never applied in any derivation of G'. Thus it is not difficult to see that $L(G) = L(G')$. □

We state the following lemma without proof.

Lemma 6. *For* $y \in \{r, t, g\}$ *and* $\mathbf{Y} \in \{\mathbf{MG}, \mathbf{CN}\}$, $\mathbf{Y}(\lambda, y) \subseteq \mathbf{Y}(-\lambda, y)$.

Using the same arguments of the proof of Lemma 7 in [1] we can show that the next statement holds.

Lemma 7. $\mathbf{Y}(\lambda, y) \subseteq \mathbf{Y}(-\lambda, y)$ *for* $\mathbf{Y} \in \{\mathbf{EFC}, \mathbf{AC}, \mathbf{ON}\}$ *and* $y \in \{r, t, g\}$.

Lemma 8. $\mathbf{PN}^{[\lambda]}(\lambda, y) \subseteq \mathbf{FC}^{[\lambda]}(\lambda, y)$ *for* $y \in \{r, t, g\}$.

Proof. Let $G = (V, \Sigma, S, R, N, \gamma, M)$ be a Petri net controlled grammar (with or without erasing rules) where $N = (P, T, F, \varphi, \iota)$. For each arc $(p,t) \in F$, we introduce new places $p_i[p,t]$, new transitions $t_i[p,t]$ and new arcs $(p, t_i[p,t])$, $(t_i[p,t], p_i[p,t])$, $(p_i[p,t], t)$ whose weights are 1's, $1 \le i \le \varphi(p,t)$, and for each arc $(t,p) \in F$, we introduce new places $p_j[t,p]$, new transitions $t_j[t,p]$ and new arcs $(t, p_j[t,p])$, $(p_j[t,p], t_j[t,p])$, $(t_j[t,p], p)$ whose weights are 1's, $1 \le j \le \varphi(t,p)$. Let

$$\begin{aligned}
P_F =& \{p_i[p,t] \mid (p,t) \in F, 1 \le i \le \varphi(p,t)\} \cup \{p_j[t,p] \mid (t,p) \in F, 1 \le j \le \varphi(t,p)\},\\
T_F =& \{t_i[p,t] \mid (p,t) \in F, 1 \le i \le \varphi(p,t)\} \cup \{t_j[t,p] \mid (t,p) \in F, 1 \le j \le \varphi(t,p)\},\\
F' =& \{(p, t_i[p,t]), (t_i[p,t], p_i[p,t]), (p_i[p,t], t) \mid (p,t) \in F, 1 \le i \le \varphi(p,t)\}\\
&\cup \{(t, p_j[t,p]), (p_j[t,p], t_j[t,p]), (t_j[t,p], p) \mid (t,p) \in F, 1 \le j \le \varphi(t,p)\}.
\end{aligned}$$

We construct a net $N' = (P \cup P_F, T \cup T_F, F', \iota')$ where the initial marking ι' is defined by $\iota'(p) = \iota(p)$ for all $p \in P$ and $\iota'(p) = 0$ for all $p \in P_F$.

Let ${}^\bullet t = \{p_1, p_2, \ldots, p_k\}$ for a transition $t \in T$ in N. Then for this transition in N' we have ${}^\bullet t = \bigcup_{i=1}^{k} \{p_j[p_i,t] \mid 1 \le j \le \varphi(p_i,t)\}$ and $(p_j[p_i,t])^\bullet = \{t\}$ for all $1 \le i \le k$ and $1 \le j \le \varphi(p_i,t)$. It follows that N' is a free-choice net.

We define a free-choice net controlled grammar $G' = (V, \Sigma, S, R, N', \gamma', M')$ where the components V, Σ, S, R are defined as for the grammar G, the free-choice net N' is constructed above. We set $\gamma'(t) = \gamma(t)$ if $t \in T$ and $\gamma'(t) = \lambda$ if $t \in T_F$; for each $\tau' \in M'$, $\tau'(p) = \tau(p)$ if $p \in P$, and for $p \in P_F$, $\tau'(p) = 0$ if $y \in \{g, t\}$, otherwise $0 \le \tau'(p) \le \tau'(p')$ where $p' \in {}^\bullet({}^\bullet p)$.

Let $D : S \xRightarrow{r_1 r_2 \cdots r_m} w \in \Sigma^*$ be a derivation in G. Then there is a successful occurrence sequence of transitions $\nu = t_1 t_2 \cdots t_n$ for M in N such that $\gamma(\nu) = r_1 r_2 \cdots r_m$. We replace ν by $\nu' = \nu_1 t_1 \nu_2 \cdots \nu_n t_n$ in N' where for all $1 \le i \le n$, $\nu_i \in \mathrm{Perm}({}^\bullet({}^\bullet t_i))$ where ${}^\bullet({}^\bullet t_i) = \{t_j[p_{i_l}, t_i] \mid 1 \le j \le \varphi(p_{i_l}, t_i), 1 \le i \le n, 1 \le l \le s\}$.

In order to fire each t_i, $1 \le i \le n$, in N', we need to fire all transitions of ${}^\bullet({}^\bullet t_i)$ at least once, therefore, ν' is successful for M' and $r_1 r_2 \cdots r_m = \gamma'(\nu')$, i.e., D is a derivation in G'.

Let $t_1 t_2 \cdots t_n$ be a successful occurrence sequence for M'. By construction, each occurrence of t_i, $1 \le i \le n$, needs at least one occurrence of all transitions of ${}^\bullet({}^\bullet t_i)$. Without loss of generality we can assume that $\nu = \nu_1^\lambda \sigma_1^\lambda t_1 \nu_2^\lambda \cdots \sigma_n^\lambda t_n \nu_{n+1}^\lambda$ where $\sigma_i^\lambda = \prod_{l=1}^{s} \prod_{j=1}^{\varphi(p_{i_l}, t_i)} t_j[p_{i_l}, t_i], 1 \le i \le n$, and $\nu_i^\lambda \in T_F^*$, $1 \le i \le n+1$.

We replace $\prod_{l=1}^{s} \prod_{j=1}^{\varphi(p_{i_l}, t_i)} t_j[p_{i_l}, t_i] t_i$ by t_i, $1 \le i \le n$, and erase ν_i^λ, $1 \le i \le n+1$. The obtained occurrence sequence $\nu' = t_1 t_2 \cdots t_n$ is successful for M in N. Then a derivation $S \xRightarrow{r_1 r_2 \cdots r_n} w \in \Sigma^*$ in G' where $r_1 r_2 \cdots r_n = \gamma'(\nu)$ is also a derivation in G and $r_1 r_2 \cdots r_n = \gamma'(\nu')$. □

The immediate consequence of this lemma is

Corollary 1.

$$\mathbf{Y}^{[\lambda]}(\lambda, y) \subseteq \mathbf{FC}^{[\lambda]}(\lambda, y)$$

where $\mathbf{Y} \in \{\mathbf{EFC}, \mathbf{AC}, \mathbf{ON}\}$ *and* $y \in \{r, g, t\}$.

Lemma 9. *For* $y \in \{r, t, g\}$, $\mathbf{FC}(\lambda, y) \subseteq \mathbf{FC}(-\lambda, y)$.

From the presented lemmas above, we can conclude that the labeling strategies of transitions of special Petri nets do not effect on the generative powers of the families of languages generated by grammars controlled by these nets.

Theorem 1. *For* $\mathbf{Y} \in \{\mathbf{SM}, \mathbf{GSM}, \mathbf{MG}, \mathbf{GMG}, \mathbf{CN}, \mathbf{FC}, \mathbf{EFC}, \mathbf{AC}, \mathbf{ON}\}$,

$$\mathbf{Y}^{[\lambda]}(f, y) = \mathbf{Y}^{[\lambda]}(-\lambda, y) = \mathbf{Y}^{[\lambda]}(\lambda, y) \text{ where } y \in \{r, t, g\}.$$

5 Results: Final Markings

In this section, we give some characterizations of the classes of languages generated by sPN controlled grammars by other classes of regulated languages.

From the structural properties of special Petri nets and Lemmas 5, 8, the next statement follows immediately

Theorem 2. *For* $\mathbf{Y} \in \{\mathbf{FC}, \mathbf{EFC}, \mathbf{AC}, \mathbf{ON}\}$ *and* $x \in \{f, -\lambda, \lambda\}$, $y \in \{r, g, t\}$,

$$\mathbf{SM}(x, y) \subseteq \mathbf{GSM}(x, y) \subseteq \mathbf{Y}(x, y) \subseteq \mathbf{Y}^{\lambda}(x, y),$$
$$\mathbf{CN}(x, y) \subseteq \mathbf{MG}(x, y) = \mathbf{GMG}(x, y) \subseteq \mathbf{Y}(x, y) \subseteq \mathbf{Y}^{\lambda}(x, y).$$

Lemma 10. $\mathbf{SM}^{[\lambda]}(\lambda, r) \subseteq \mathbf{SM}^{[\lambda]}(\lambda, t)$.

Proof. Let $G = (V, \Sigma, S, R, N, \gamma, M)$ be a state machine controlled grammar (with or without erasing rules) where $N = (P, T, F, \iota)$.

Since the firing of a transition in a state machine moves one token from the input place to the output place, the number of tokens in the net remains the same in any firing of a transition. It follows that the set M of all reachable markings is finite, i.e.,

$$|M| \leq \binom{n+k-1}{k-1}$$

where $n = \sum_{p \in P} \iota(p)$ and $k = |P|$ ($\binom{n+k-1}{k-1}$ *is the number of solutions in non-negative integers to the equation* $x_1 + x_2 + \cdots + x_k = n$, see [10]). □

From Lemma 10 the next statements follow immediately

Corollary 2. $\mathbf{SM}^{[\lambda]}(\lambda, r) \subseteq \mathbf{SM}^{[\lambda]}(\lambda, g)$ *and* $\mathbf{SM}^{[\lambda]}(\lambda, g) \subseteq \mathbf{SM}^{[\lambda]}(\lambda, t)$.

We state the following lemma without proof.

Lemma 11. $\mathbf{SM}^{[\lambda]}(\lambda, t) \subseteq \mathbf{SM}^{[\lambda]}(\lambda, r)$.

Lemma 12. $\mathbf{MAT}^{[\lambda]} \subseteq \mathbf{SM}^{[\lambda]}(f, t)$.

Proof. Let $G = (V, \Sigma, S, M)$ be a matrix grammar (with or without erasing rules) and $M = \{m_1, m_2, \ldots, m_n\}$ where $m_i = r_{i1} r_{i2} \cdots r_{ik(i)}$, $1 \leq i \leq n$. Without loss of generality we can assume that G is without repetitions. Let

$R = \{r_{ij} \mid 1 \le i \le n, 1 \le j \le k(i)\}$. We define a (f,t)-SM controlled grammar $G' = (V, \Sigma, S, R, N, \gamma, \{\mu\})$ where the sets of places, transitions and arcs of the a SM $N = (P, T, F, \iota)$ are defined by $P = \{p_0\} \cup \{p_{ij} \mid 1 \le i \le n, 1 \le j \le k(i)-1\}$, $T = \{t_{ij} \mid 1 \le i \le n, 1 \le j \le k(i)\}$ and

$$\begin{aligned} F = &\{(p_0, t_{i1}), (t_{ik(i)}, p_0) \mid 1 \le i \le n\} \cup \{(p_{ik(i)-1}, t_{ik(i)}) \mid 1 \le i \le n\} \\ &\cup \{(t_{ij}, p_{ij}) \mid 1 \le i \le n, 1 \le j \le k(i)-1\}. \end{aligned}$$

The initial marking is defined by $\iota(p_0) = 1$, and $\iota(p) = 0$ for all $P - \{p_0\}$.

The bijection $\gamma : T \to R$ is defined by $\gamma(t_{ij}) = r_{ij}$, $1 \le i \le n, 1 \le j \le k(i)$ and the final marking μ is the same as the initial marking ι.

Let $S = w_0 \overset{m_{i_1}}{\Longrightarrow} w_1 \overset{m_{i_2}}{\Longrightarrow} \cdots \overset{m_{i_l}}{\Longrightarrow} w_l = w \in \Sigma^*$ be a derivation in G, where $m_{i_j} \in M$, $1 \le j \le l$, and $w_{j-1} \overset{m_{i_j}}{\Longrightarrow} w_j :\ w_{j-1} \overset{r_{i_j 1} r_{i_j 2} \cdots r_{i_j k(i_j)}}{\Longrightarrow} w_j$. By definition of γ, $\gamma^{-1}(m_{i_j}) = \sigma_j$ where $\sigma_j = t_{i_j 1} t_{i_j 2} \cdots t_{i_j k(i_j)}$ for all $1 \le j \le l$. Then the occurrence sequence of transitions $\iota \xrightarrow{\sigma_1 \sigma_2 \cdots \sigma_l} \iota$ is a successful for $\{\mu\}$. Therefore, $S \overset{m_{i_1} m_{i_2} \cdots m_{i_l}}{\Longrightarrow} w_l \in \Sigma^*$ is a derivation in G'.

The inverse inclusion can also be shown using the same arguments. □

Lemma 13. *For $y \in \{r, g, t\}$, $\mathbf{SM}^{[\lambda]}(\lambda, y) \subseteq \mathbf{rC}^{[\lambda]}$.*

Proof. Let $G = (V, \Sigma, S, R, N, \gamma, M)$ be a SM controlled grammar (with or without erasing rules) where $N = (P, T, F, \iota)$. We construct a (deterministic) finite automaton $\mathcal{A}$ whose states are the markings of the net N (since the set of all reachable markings of a state machine is finite, it can be considered as a set of states) and there is an arc from state μ to state μ' with label t iff marking μ' is obtained from marking μ by firing transition t. The initial marking is considered as the initial state and the set of final markings M as a set of final states.

Formally, $\mathcal{A} = (M', T, \iota, \delta, M)$ where M' is the set of all reachable markings of the net N and the state-transition function $\delta : M' \times T \to M'$ is defined by $\delta(\mu, t) = \mu'$ iff $\mu \xrightarrow{t} \mu'$. It is not difficult to see that $\sigma = t_1 t_2 \cdots t_n \in L(\mathcal{A})$ iff σ is a successful occurrence sequence of transitions of N. Therefore, $L(G) = L(G')$ where $G' = (V, \Sigma, S, R, L(\mathcal{A}))$ is a regularly controlled grammar. □

For each vector grammar, if we define an ordinary net as a union of disjoint pathes of the form $\rho = t_1 p_1 t_2 p_2 \cdots p_{n-1} t_n$ for each matrix $m = r_1 r_2 \cdots r_n$ where t_i labeled with r_i, $1 \le i \le n$, then it is not difficult to see that an grammar controlled by this net generates the same language as the vector grammar.

Lemma 14. $\mathbf{V}^{[\lambda]} \subseteq \mathbf{MG}^{[\lambda]}(f,t) \cap \mathbf{CN}^{[\lambda]}(f,t) \cap \mathbf{GSM}^{[\lambda]}(f,t)$.

For each semi-matrix grammar, if an ordinary net is defined as a union of disjoint cycles of the form $\rho = p_1 t_1 p_2 t_2 \cdots p_n t_n p_1$ for each matrix $m = r_1 r_2 \cdots r_n$ where t_i labeled with r_i, $1 \le i \le n$, then it is not difficult to see that an grammar controlled by this net generates the same language as the semi-matrix grammar.

Lemma 15. $\mathbf{sMAT}^{[\lambda]} \subseteq \mathbf{SM}^{[\lambda]}(f,t) \cap \mathbf{MG}^{[\lambda]}(f,t)$.

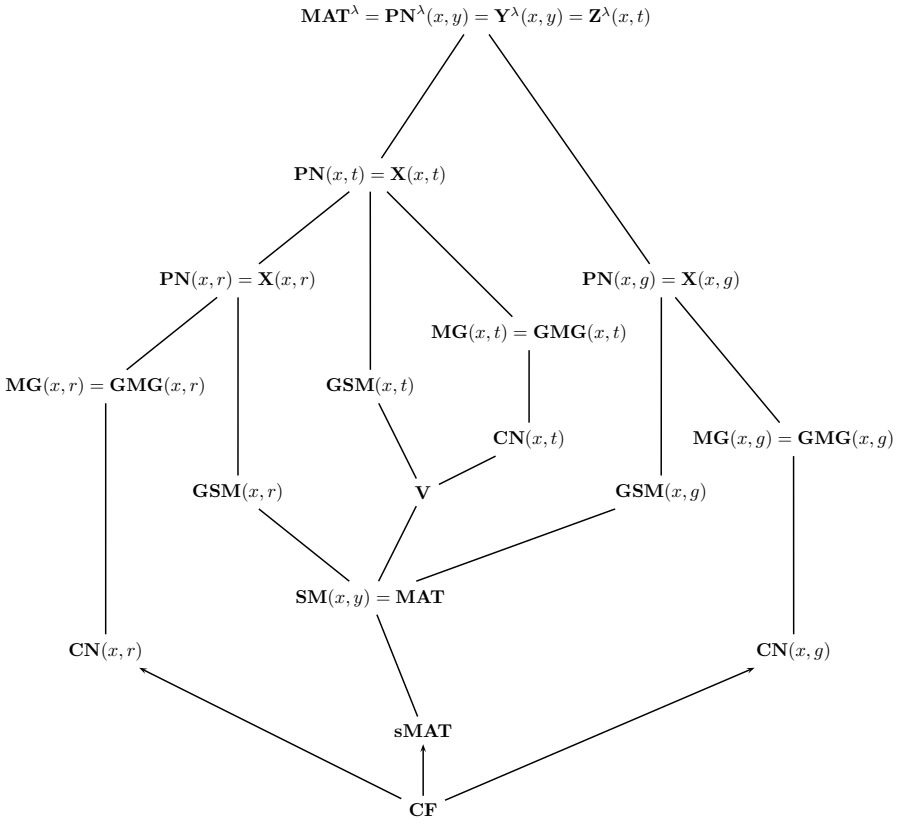


Fig. 4. The hierarchy of language families generated by Petri net controlled grammars

Now we summarize our results in the following theorem.

Theorem 3. *The relations in Figure 4 hold where* $x \in \{f, -\lambda, \lambda\}$, $y \in \{r, g, t\}$, $\mathbf{X} \in \{\mathbf{FC}, \mathbf{EFC}, \mathbf{AC}, \mathbf{ON}\}$, $\mathbf{Y} \in \{\mathbf{SM}, \mathbf{GSM}, \mathbf{FC}, \mathbf{EFC}, \mathbf{AC}, \mathbf{ON}\}$ *and* $\mathbf{Z} \in \{\mathbf{MG}, \mathbf{GMG}, \mathbf{CN}\}$*; the lines (arrows) denote (proper) inclusions of the lower families into the upper families.*

Proof. The relations of the language families generated by sPN controlled grammars follow from Theorem 2. From Theorem 2.1.2 in [8], Theorem 12 in [9] and above presented lemmas we get $\mathbf{sMAT} \subseteq \mathbf{SM}(x,y) = \mathbf{MAT} \subseteq \mathbf{V} \subseteq \mathbf{CN}(x,t) \subseteq \mathbf{MG}(x,t) = \mathbf{GMG}(x,t)$ and $\mathbf{V} \subseteq \mathbf{GSM}(x,t)$.

The inclusion $\mathbf{X}(x,r) \subseteq \mathbf{X}(x,t)$, $\mathbf{X} \in \{\mathbf{FC}, \mathbf{EFC}, \mathbf{AC}, \mathbf{ON}\}$, follows from Theorem 13 in [1] and Lemma 8. Again from Theorem 13 in [1] we have $\mathbf{PN}^\lambda(x,r) = \mathbf{MAT}^\lambda \subseteq \mathbf{PN}^\lambda(x,t)$. If we define erasing matrices in the proof of Lemma 11 in [1] for each $\tau \in M$ as

$$m_\tau = (B \to \lambda, \underbrace{\bar{p}_1 \to \lambda, \ldots, \bar{p}_1 \to \lambda}_{\tau(p_1)}, \ldots, \underbrace{\bar{p}_n \to \lambda, \ldots, \bar{p}_n \to \lambda}_{\tau(p_n)})$$

where $P = \{p_1, p_2, \ldots, p_n\}$, then we also get $\mathbf{PN}^{\lambda}(x,t) \subseteq \mathbf{MAT}^{\lambda}$. If we consider erasing matrices of the form $m_{\lambda} = (\bar{p} \to \lambda)$ for each $p \in P$ together with the matrices defined above then it is easy to see that $\mathbf{PN}^{\lambda}(x,g) \subseteq \mathbf{MAT}^{\lambda}$.

The equalities $\mathbf{SM}^{\lambda}(x,y) = \mathbf{GSM}^{\lambda}(x,y) = \mathbf{MG}^{\lambda}(x,t) = \mathbf{GMG}^{\lambda}(x,t) = \mathbf{CN}^{\lambda}(x,t) = \mathbf{FC}^{\lambda}(x,y) = \mathbf{EFC}^{\lambda}(x,y) = \mathbf{AC}^{\lambda}(x,y) = \mathbf{ON}^{\lambda}(x,y) = \mathbf{PN}^{\lambda}(x,y) = \mathbf{MAT}^{\lambda}$ follows from the fact that $\mathbf{sMAT}^{\lambda} = \mathbf{V}^{\lambda} = \mathbf{MAT}^{\lambda}$ (Theorem 2.1.2, [8] and Theorem 12, [9] and the above-presented lemmas). □

References

1. Dassow, J., Turaev, S.: Arbitrary petri net controlled grammars. In: Enguix, G., Jiménez-López, M. (eds.) The Second International Workshop on Non-classical Formal Languages in Linguistics, Forling 2008, Tarragona, Spain, pp. 27–39 (2008)
2. Dassow, J.: Subregularly controlled derivations: Context-free case. Rostock. Math. Kolloq. 34, 61–70 (1988)
3. Dassow, J., Truthe, B.: Subregularly tree controlled grammars and languages. In: Csuhaj-Varjú, E., Esik, Z. (eds.) The 12th Inernational Conference AFL 2008, Balatonfured, Hungary, pp. 158–169 (2008)
4. Baumgarten, B.: Petri-Netze. Grundlagen und Anwendungen. Wissensschaftverlag, Mannheim (1990)
5. Desel, J., Esparsa, J.: Free-Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, vol. 10. Cambridge University Press, Cambridge (1995)
6. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. I–III. Springer, Berlin (1996)
7. Reisig, W., Rozenberg, G. (eds.):Lectures on Petri Nets I: Basic Models. LNCS, vol. 1491. Springer, Heidelberg (1998)
8. Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. Springer, Berlin (1989)
9. Turaev, S.: Semi-matrix grammars. In: The Second Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, MEMICS 2006, Mikulov, Czech Republic, pp. 245–252 (2006)
10. Graham, R., Lovász, M.G. (eds.): Handbook of Combinatorics, vol. I–II. Elsevier(North-Holland), Amsterdam and MIT Press, Cambridge (1996)

Nested Counters in Bit-Parallel String Matching

Kimmo Fredriksson[1] and Szymon Grabowski[2]

[1] Department of Computer Science, University of Kuopio,
P.O. Box 1627, 70211 Kuopio, Finland
fredriks@cs.uku.fi
[2] Computer Engineering Department, Technical University of Łódź,
Al. Politechniki 11, 90–924 Łódź, Poland
sgrabow@kis.p.lodz.pl

Abstract. Many algorithms, e.g. in the field of string matching, are based on handling many counters, which can be performed in parallel, even on a sequential machine, using bit-parallelism. The recently presented technique of nested counters (*Matryoshka counters*) [1] is to handle small counters most of the time, and refer to larger counters periodically, when the small counters may get full, to prevent overflow. In this work, we present several non-trivial applications of Matryoshka counters in string matching algorithms, improving their worst- or average-case time complexities. The set of problems comprises (δ, α)-matching, matching with k insertions, episode matching, and matching under Levenshtein distance.

1 Introduction

A word RAM is a random-access machine with unit-cost operations for operands of w bits, and having instruction set similar to modern computers. An especially practical variant of word RAM is *transdichotomous RAM* [2]. This model assumes that $w = \Omega(\log(n))$, where n is the "input size" (or simply $w \geq \log_2(n)$; we also sometimes distinguish between a weaker assumption, that $w = \Theta(\log(n))$, and the more general case). Note that the word RAM model allows e.g. to sort n integers in $o(n \log n)$ time, which is impossible in the comparison model.

Bit-parallelism [3] is now an established and highly successful algorithmic technique, useful especially in the field of string matching. Basically, it makes use of wide machine words (CPU registers) to parallelize the work of other algorithms, e.g., filling the matrix in a dynamic programming algorithm or simulating a non-deterministic automaton.

Most interesting string matching problems can be classified as approximate string matching [4] of some sort. There is a plethora of approximate matching models, with applications in natural language processing, computational biology, music information retrieval, image processing and other areas. Many approximate matching problems can be stated like that. Given a text $T = t_0 \dots t_{n-1}$, a pattern $P = p_0 \dots p_{m-1}$, over some alphabet Σ, and a threshold k, we want to find all the text positions where the pattern matches the text with at most k errors. How the error measure is calculated constitutes the actual problem.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 338–349, 2009.

Bit-parallel solutions to approximate string matching problems often deal with *counters*. The counters are bit-fields storing the total errors for individual states of the problem, e.g., for every pattern prefix. Many such counters, each of size e.g. $O(\log(k))$ bits, are updated in parallel, which makes the respective algorithms efficient in practice.

In a recent work [1], we showed a simple technique to decrease the worst-case time complexity of one of the best known bit-parallel algorithms based on counters, Shift-Add [3], from $O(n\lceil m\log(k)/w\rceil)$ to $O(n\lceil m/w\rceil)$. The underlying observation was that the counter values grow by at most one per read text character, hence most of the time the algorithm could update much smaller counters, and only when they can overflow, at periodical moments, their content could be added (and then flushed) to a high-level counters, occupying more bits. This idea was generalized to many counter levels, and dubbed *Matryoshka counters*, to reflect their nested nature.

The current work presents several non-trivial applications of Matryoshka counters in string matching algorithms, improving their time complexities in the worst or average case.

2 Preliminaries

Let the pattern $P = p_0p_1p_2\ldots p_{m-1}$ and the text $T = t_0t_1t_2\ldots t_{n-1}$ be strings over alphabet $\Sigma = \{0, 1, \ldots, \sigma - 1\}$. The pattern has an exact occurrence in some text position j, if $p_i = t_{j-m+1+i}$ for $i = 0\ldots m-1$. If $p_i \neq t_{j-m+1+i}$ for at most k positions, then the pattern has an approximate occurrence with at most k mismatches. The number of mismatches is called *Hamming distance*. We want to report all text positions j where the Hamming distance is at most k.

To present our algorithms, some extra notation is needed. We number the bits from the least significant (right-most) bit (0) to the most significant (left-most) bit $(w-1)$. Bit-wise operations are denoted like in C language: & is bit-wise `and`, | is `or`, ^ is `xor`, $\sim$ negates all bits, << and >> are shift to left and to right, with zero padding. Exponentiation denotes bit repetition, as in $(1(10)^2)^2 = 1101011010$. In general, all formulas of the above form will be (run-time) constants that can be precomputed. The notation $V_{[i]\ell}$ denotes the ith ℓ-bit field of the bit-vector V.

Shift-Add algorithm. To be able to describe our new algorithms we need to briefly cover some previous work. Shift-Add [3] is a bit-parallel algorithm for approximate searching under Hamming distance. It reserves a *counter* of $\ell = \lceil \log_2(k+1)\rceil + 1$ bits for each pattern character in a bit-vector D of length $m\ell$ bits. This bit-vector denotes the search state: the ith counter tells the number of mismatches for the pattern prefix $p_0\ldots p_i$ for some text substring $t_{j-i}\ldots t_j$. If the $(m-1)$th counter is at most k at any time, i.e. $D_{[m-1]\ell} \leq k$, then we know that the pattern occurs with at most k mismatches in the current text position j. The preprocessing algorithm builds an array B of bit-vectors. More precisely, we set $B[c]_{[i]\ell} = 0$ iff $p_i = c$, else 1. Then we can accumulate the mismatches as $D \leftarrow (D << \ell) + B[t_j]$. I.e. the shift operation moves all counters at position i to

position $i+1$, and effectively clears the first counter. The $+ B[t_j]$ operation then adds 0 or 1 to each counter, depending on whether the corresponding pattern characters match t_j. Note that the number of mismatches can be as large as m, i.e., the counters in D can overflow. The solution is to store the highest bits of the fields in a separate computer word o, and keep the corresponding bits cleared in D. Shift-Add works in $O(n\lceil m \log(k)/w \rceil)$ time in the worst case.

Counter-splitting. In [1] it was shown how the number of bits for Shift-Add can be reduced. To this end, we use two levels of counters. The top level is as in plain Shift-Add, i.e. we use $\ell = O(\log(k))$ bits. For the bottom level we use only $\ell' = \log_2(\log_2(k+1)+1)$ bits. The basic idea is then to use a bit-vector D' of $m\ell'$ bits, and accumulate the mismatches as before. However, these counters may overflow every $2^{\ell'}$ steps. We therefore add D' to D at every $2^{\ell'}-1$ steps, and clear the counters in D'. The counters in D have $\ell = \lceil \log_2(k+2^{\ell'}) \rceil + 1 = O(\log(k))$ bits each. The result is that updating D' takes only $O(\lceil m \log\log(k)/w \rceil)$ worst case time per text character, and updating D takes only $O(\lceil m \log(k)/w \rceil / 2^{\ell'}) = O(m/w)$ amortized worst case time. The total time is then dominated by computing the D' vectors, leading to $O(n\lceil m \log\log(k)/w \rceil)$ total time.

Now adding the two sets of counters can be done without causing an overflow, but the problem is how to add them in parallel. The difficulty is that the counters have different number of bits, and hence are unaligned. The vector D' must therefore be expanded so that we insert $\ell - \ell'$ zero bits between all counters prior to the addition, i.e. we must obtain a bit-vector x, so that $x_{[i]\ell} = D'_{[i]\ell'}$. Then we need to effectively add the counters in D and D' as $D + x$, i.e. $D + Expand(D')$. The function $Expand(\cdot)$ can be computed in $o(1)$ amortized time [1].

Note that we cannot shift the vector D at each step as this would cost $O(\lceil m\ell/w \rceil)$ time. Instead, we shift it only each $2^{\ell'}-1$ steps in one shot prior to adding the two counter sets: $D \leftarrow D << (2^{\ell'}-1)\ell$. As in plain Shift-Add, we take care not to overflow the counters (see [1] for details).

The final issue is the detection of the occurrences. At each step j, we just add $D'_{[m-1]\ell'}$ and $D_{[m-i]\ell}$, in the ith bottom level iteration ($i = 1 \ldots 2^{\ell'}-1$). This constitutes the true sum of mismatches for the whole pattern at text position j. If this sum is at most k, we report an occurrence. This takes only a constant time since we only add up two counters, one from each of the two vectors (the whole counter sets are added only each $2^{\ell'}-1$ steps). The vector D is not shifted at each step, but we simulate the shift by selecting the $(m-i)$th field when detecting the possible occurrences. Summing up, we have $O(n\lceil m \log\log(k)/w \rceil)$ worst case time algorithm for string matching under Hamming distance. The above scheme can be improved by using more counter levels. We call these *Matryoshka counters*, to reflect their nested nature. The basic ideas remain the same, but the some details become more involved, see [1] for details. The end result is an algorithm with total worst case time of $O(n\lceil m/w \rceil)$.

Contracting counters. In what follows, we will need the inverse of $Expand(\cdot)$, i.e. we need to compute $x'_{[i]\ell'} \leftarrow y_{[i]\ell}$ for all i efficiently. We call this function $Contract(\cdot)$. Note that this is not possible in general, as the value of $y_{[i]\ell}$ may not

fit into ℓ' bits. However, in our application we have a guarantee that ℓ' bits will suffice. In general, we may assume that we want to compute $x'_{[i]\ell'} \leftarrow y_{[i]\ell} \;\&\; 1^{\ell'}$ for all i. It is easy to see that this can be computed in parallel just by inverting and doing in opposite order all the steps required for expanding the counters (or by using precomputed tables, which makes the task trivial). Hence the time bound remains also the same, including the amortized $o(1)$ time, as we will be doing this operation only in the companion of *Expand*.

Space complexities. In all our algorithms, the most space consuming structure used is the array of bit-vectors B. The preprocessing cost, in space and time, of B is bounded by $O(m\sigma)$, for all the algorithms. In some cases (see Sec. 4), we may need a look-up table of size e.g. $O(\sqrt{n})$ words. For large σ we can either (1) in $O(n+\sigma)$ time and $O(\sigma)$ space remap T and P to a new alphabet of size $O(m)$, or (2) if $O(\sigma)$ is too large, or the actual values of the symbols are of importance, as e.g. with the δ-relaxation (see Sec. 3), we can consider the resulting $O(m)$ equivalence classes for the alphabet symbols and still map the alphabet to $O(m)$ symbols in $O(n \log m)$ (or e.g. $O(n \log_w(m))$ [5]) time and $O(m)$ space.

3 (δ, α)-Matching

In (δ, α)-matching the pattern matches the text substring $t_{j_0} t_{j_1} t_{j_2} \ldots t_{j_{m-1}}$ if $|p_i - t_{j_i}| \leq \delta$, and $j_i < j_{i+1}$ and $j_{i+1} - j_i \leq \alpha + 1$. In other words, there can be α non-matching text symbols between each matching pattern symbol. There are many efficient solutions to this problem, see e.g. [6] for some recent results and review of the problem. The first bit-parallel algorithm [7] solves the problem in $O(n\lceil m\alpha/w \rceil)$ worst case time. This algorithm is based on simulating non-deterministic finite automata. A different solution [8], loosely based on Shift-Add, solves the problem in $O(n\lceil m \log(\alpha)/w \rceil)$ worst case time. Below we review this approach and improve it to take only $O(n\lceil m \log\log(\alpha)/w \rceil)$ time.

At a high level, the algorithm can be seen as a combination of Shift-And and Shift-Add algorithms [3]. The 'automaton' has two kinds of states: Shift-And and Shift-Add ones. The Shift-And states (vector D) keep track of the pattern symbols, while the Shift-Add states (vector C) keep track of the gap length between the symbols. The result is a systolic array rather than automaton; a high level description of a building block for symbol p_i is shown in Fig. 1. The final array is obtained by concatenating one building block for each pattern symbol. The building blocks are the counters. Only $\ell = \lceil \log_2(\alpha + 1) \rceil + 1$ bits are reserved for each counter. The value $2^{\ell-1} - (\alpha + 1)$ is used to initialize the counters, i.e. to represent the value 0. This means that the highest bit (ℓth bit) of the counter becomes 1 when the counter has reached a value $\alpha + 1$, i.e. the gap cannot be extended anymore. Hence, line 1 of the algorithm in Fig. 1 can be computed bit-parallelly as

$$C \leftarrow C + ((\sim C >> (\ell - 1)) \;\&\; (0^{\ell-1}1)^m). \tag{1}$$

That is, we negate and select the highest bit of each counter (shifted to the low bit positions), and add the result to the original counters. If a counter value is

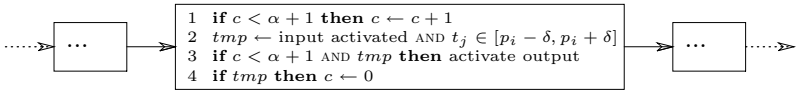


Fig. 1. A building block for a systolic array detecting δ-matches with α-bounded gaps

less than $\alpha + 1$, then the highest bit position is not activated, and hence the counter gets incremented by one. If the bit was activated, we effectively add 0.

To detect the δ-matching characters we need to preprocess a table B, so that $B[c]$ has $i\ell$th bit set to 1, iff $|p_i - c| \leq \delta$. We can then use a Shift-And step:

$$tmp \leftarrow ((D << \ell) \mid 1) \ \& \ B[t_i], \tag{2}$$

where we have reserved ℓ bits per character in D as well. Only the lowest bit of each field has any significance, the rest are only for aligning D and C appropriately. The reason is that a state in D may be activated also if the corresponding gap counter has not exceeded $\alpha+1$. In other words, if the highest bit of a counter in C is not activated (the gap condition is not violated), then the corresponding bit in D should be activated:

$$D \leftarrow tmp \mid ((\sim C >> (\ell - 1)) \ \& \ (0^{\ell-1}1)^m) \tag{3}$$

The only remaining difficulty to solve is how to reinitialize (parallelly) some subset of the counters to zero, i.e. how to implement line 4 in Fig. 1. The bit-vector tmp has value 1 in every field position that survived the Shift-And step, i.e. in every field position that needs to be initialized in C. Then

$$C \leftarrow (C \ \& \sim(tmp \times 1^\ell)) \mid (tmp \times (2^{\ell-1} - (\alpha + 1))) \tag{4}$$

first clears the corresponding counter fields, and then copies the initial value $2^{\ell-1} - (\alpha + 1)$ to all the cleared fields. This completes the algorithm. Clearly, it runs in $O(n\lceil m \log(\alpha)/w \rceil)$ worst case time.

3.1 Two-Level Solution

Let us now improve this algorithm. We use the two-level variant. Basically, $\ell' = O(\log\log(\alpha))$ bits per counter is enough for the bottom level. However, the solution is somewhat more complicated than for the previous algorithm. The reason is that we need to know for *each step* of the algorithm if *any* of the counters have reached the value $\alpha + 1$ or not, since this information is used to compute the next value of the counters at each step. I.e. it is not enough to detect this for the last counter, which was easy. Again, we use $\ell = \lceil \log_2(\alpha + 2^{\ell'}) \rceil + 1$, just as in the Shift-Add algorithm (now k is just called α). However, for the bottom level we add one additional bit per field. This new highest bit will be used to signal whether the corresponding counter has exceeded the value $\alpha + 1$. The two counter levels are added after each $2^{\ell'-1} - 1$ steps, so the highest bits of the bottom level counters never get activated if the counters started from zero.

The above base-line algorithm is used as is for the bottom level counters (we just replace ℓ with ℓ' everywhere), with the following modification: the initial values copied to the counters are 0. The bottom and top level counters are denoted as C' and C, respectively.

The first modification needed is handling the zeroing of the counters. If a bottom level counter gets zeroed, it should be done for the corresponding top level counter as well. But we cannot afford doing it in every step, since the cost would be too high. Instead, we introduce new bit-vector Z, which is initialized to all zeroes after each top level update. For each bottom level step we record the counters that should be zeroed for the top level as well:

$$Z \leftarrow Z \mid tmp, \tag{5}$$

where tmp is as in Eq. (2). In other words, Z gets bit 1 to every "counter" field that is zeroed. This is then used in each top level step to clear the corresponding fields of the top level counter. This is done as in Eq. (4):

$$Z \leftarrow Expand(Z) \tag{6}$$

$$C \leftarrow (C \ \& \sim((Z << \ell) - Z)) \mid (Z \times (2^{\ell-1} - (\alpha + 1))). \tag{7}$$

Note that we expanded Z first. Adding the two counter sets is straightforward; C' is expanded and added to C, taking care (with the standard technique) not causing an overflow:

$$C \leftarrow ((C \ \& \sim((10^{\ell-1})^m)) + Expand(C')) \mid (C \ \& \ (10^{\ell-1})^m). \tag{8}$$

After adding the two counter levels, we must bring some information from the top level back to the bottom level. Consider the top level counter $C_{[i]\ell}$. Three cases can occur:

1. $C_{[i]\ell} > \alpha+1$ (the highest bit of the counter is set): the counter has overflowed. In this case we set $C'_{[i]\ell'} = 2^{\ell'-1}$, i.e. activate the overflow bit of the bottom level counter.
2. $C_{[i]\ell} \leq \alpha + 1 - (2^{\ell'-1} - 1)$: the counter cannot overflow in the next $2^{\ell'-1} - 1$ steps, and we can safely set $C'_{[i]\ell'}$ to 0.
3. $\alpha + 1 \geq C_{[i]\ell} > \alpha + 1 - (2^{\ell'-1} - 1)$: the counter may or may not overflow in the next $2^{\ell'-1} - 1$ steps. In other words, if the counter $C'_{[i]\ell'}$ has the value v, where $v > \alpha + 1 - C_{[i]\ell}$, the sum of the two counters has overflowed. Let $v = \alpha+1-C_{[i]\ell}$. By the assumption of this case, $v < 2^{\ell'-1}$. Let $u = 2^{\ell'-1} - v$. If the counter $C'_{[i]\ell'}$ is initialized to u, the highest bit will be activated as soon as α-gap condition is violated. To keep the true sum of the counter levels correct, we also subtract u from the top level counter, i.e. set $C_{[i]\ell} \leftarrow C_{[i]\ell} - u$.

The implementation of the cases 1 and 2 are easy. The case 2 is handled implicitly, i.e. all fields not touched by case 1 or 3 are left to zero. The first case is simple. The highest bits are extracted from C, and shifted to the ℓ'-th positions. The

result can be contracted to fit into the bottom level counters, activating the corresponding highest bits:

$$C' \leftarrow Contract((C \ \& \ (10^{\ell-1})^m) >> (\ell - \ell')). \tag{9}$$

The case 3 is slightly trickier. First, those ℓ bit fields of C that have not the highest bit set are selected, in vector x:

$$o \leftarrow C \ \& \ (10^{\ell-1})^m \tag{10}$$
$$x \leftarrow C \ \& \ \sim((o << 1) - (o >> (\ell - 1))). \tag{11}$$

We then add $2^{\ell'-1} - 1$ to every field of x, again select the highest bits of the result, signaling the fields that may reach $\alpha + 1$, and form a new mask selecting these fields:

$$y \leftarrow (x + (2^{\ell'-1} - 1) \times (0^{\ell-1}1)^m) \ \& \ (10^{\ell-1})^m \tag{12}$$
$$o \leftarrow (y << 1) - (y >> (\ell - 1)) \tag{13}$$

Then, the value of v can be computed as described above. Note that for C counters the value of zero is represented as $2^{\ell-1} - (\alpha + 1)$, which must be taken into account:

$$v \leftarrow (((\alpha + 1 + 2^{\ell-1} - (\alpha + 1)) \times (0^{\ell-1}1)^m) \ \& \ o) - (C \ \& \ o). \tag{14}$$

However, this simplifies to: $v \leftarrow ((10^{\ell-1})^m \ \& \ o) - (C \ \& \ o)$. Similarly, we compute the value for u (note that now zero is represented as 0):

$$u \leftarrow ((2^{\ell'-1} \times (0^{\ell-1}1)^m) \ \& \ o) - v, \tag{15}$$

which again simplifies to: $u \leftarrow ((0^{\ell-\ell'}10^{\ell'-1})^m \ \& \ o) - v$. Finally, the values are subtracted from C, and added to C', so that $C + Expand(C')$ will be correctly maintained:

$$C \leftarrow C - u \tag{16}$$
$$C' \leftarrow C' \mid Contract(u) \tag{17}$$

Following the Shift-Add analysis, it is easy to find the $O(n\lceil m \log\log(\alpha)/w \rceil)$ time complexity. Note that we can easily replace all the multiplications in the search code with precomputed masks and bitwise operations, and thus the analysis is valid even in AC^0 RAM model (as detailed in [1] and Sec. 2, $Expand()$ and $Contract()$ can be implemented efficiently without multiplications).

4 Intrusion Detection and Episode Matching

A close relative to α-matching is searching allowing k insertions of symbols into the pattern. In other words, we want to find all minimal length text substrings t, such that $id(P, t) \leq k$, where $id(P, t)$ is the minimum number of symbols

inserted to P, to convert it to t. It follows that if P is a subsequence of t, then $id(P,t) = |t| - |P| = |t| - m$, and ∞ otherwise, and that $m \leq |t| \leq m + k$ if t matches P with at most k insertions. This matching model has important applications in intrusion detection [9]. The problem can be solved using dynamic programming [9]. We define a vector C of counters: $C_i = id(p_{0\ldots i}, t_{h\ldots j})$, for $i = 0 \ldots m$ and some h. The initial values are $C_0 = 0$, and $C_{i>0} = \infty$. (The value ∞ can be represented as any value $> k$ in practical implementation.) Therefore, the goal is to report all j such that $C_m \leq k$. The computation of the new values of C, given the old values C^o, is based on the following recurrence:

$$C_i = C^o_{i-1}, \text{if } p_i = t_j, \text{and } C^o_i + 1 \text{ otherwise.} \tag{18}$$

The obvious implementation of the recurrence leads to $O(mn)$ worst case time. It was then shown by Kuri et al. [9] how to compute C_i using bit-parallelism, which resulted in an $O(n\lceil m \log(k)/w \rceil)$ worst case time algorithm. We briefly present the Kuri et al. algorithm as this is the starting point of our solution.

The C_i counters are packed into machine words. Only error counts up to $k+1$ are interesting, so any C_i value above k could be replaced by $k+1$, and a similar effect is achieved using overflow bits, a technique known from Shift-Add. In this way, the counters occupy $\ell = \lceil \log_2(k+1) \rceil$ bits each. A table B storing σ bit-vectors of length $m(\ell+1)$ is built in the preprocessing. Each $(\ell+1)$-bit field of $B[c]$ is set to 01^ℓ if $c = p_i$, and $0^{\ell+1}$ otherwise. Note that the highest bit in each field is 0. Also the state vector D has $m(\ell+1)$ bits, initialized to 0s. If the counters could use $O(\log(k))$ bits, the update formula (invoked once per text character) could be simply

$$D^{new} = (B[t_j] \;\&\; (D << (\ell+1))) \mid (\sim B[t_j] \;\&\; (D + (0^\ell 1)^m)), \tag{19}$$

but the real algorithm is somewhat more complicated due to handling the counter overflows in the usual way, see [9] for details. Let us only note for the formula above that each field of $B[t_j]$ selects between $(D << (\ell+1))$ operation, which corresponds to $C_i \leftarrow C^o_{i-1}$ assignment in the plain dynamic programming formula, and $(D + (0^\ell 1)^m)$, which replaces $C_i \leftarrow C^o_i + 1$ assignment.

It might seem that nested counters could be used for this algorithm just as easily as with Shift-Add, but there is actually a new problem. As seen in Eq. 18, the new values C_i depend on the old values of C in less "predictable" way than it was in Shift-Add. In Shift-Add the counter values are simply shifted left (i.e., to the next position) with each text character (and then their counts possibly increased by 1), while here they depend on a condition. Let us assume a two-level counter scheme. The manipulations on counters should be done both in the bottom level and the top level. The top level updates should be done infrequently, and here is where the problem lies, as it seems difficult to delay such operations and then perform a bulk update in constant time. We found a compromise solution though.

Our algorithm gives an improvement over the Kuri et al. algorithm if $\log(k) = \omega(\log(w))$. This may seem quite restrictive but for the intrusion detection

problem large values of k (exceeding m) are quite typical. The basic observation is that during the inner loop the *set of distinct values* in the top level counters is never extended, as the counter values are simply copied from one to another ($C_i \leftarrow C_{i-1}$ operations). So, we do not need to know their actual values, only we need to distinguish those m counters somehow. To this end, just before the inner loop we label the top level counters. Fortunately, we do not need to give them truly unique labels (which would imply $\log(m)$ bits per label) but only choose from a smallest set of labels which prevents from losing identity of any counter during the inner loops. Since the copy operations always involve only adjacent counters, it is enough to assign label $i \mod 2^{\ell'}$ to a counter at position i. In this way, we need ℓ' bits per label. Now, for every text character also the upper level may change, but all the copy operations on labels are performed in parallel, with $O(m\ell'/w)$ time per character.

When the inner loop is over, we need to get back the true top level counters, to increase their counts with values from the bottom level. This requires remapping with the labels mentioned above, reflecting the actual label arrangement. A brute-force rearrangement takes $O(m)$ time, after which we we can update the top level counters with the respective counts from the bottom level, in $O(m\ell/w)$ time. The total time spent on bottom level operations is $O(n\lceil m\ell'/w\rceil)$. The total time spent for the top level is $O(n\lceil m\ell'/w\rceil + n/2^{\ell'}(m + \lceil m\ell/w\rceil))$. The sum of the above is optimized for $\ell' = \log(w)$, which gives $O(n\lceil m\log(w)/w\rceil)$ overall worst-case time complexity. Finally, assuming that $w = \Theta(\log(n))$, we can improve the brute-force counter rearrangement to take just $O(m/w)$ amortized time per text character, by using look-up tables. In this case we can use $\ell' = \Theta(\log\log(k))$, and $\ell = \Theta(\log(k))$, achieving $O(n\lceil m\log\log(k)/\log(n)\rceil)$ total time.

We note that for the opposite scenario, i.e., for small k, a theoretical $O(nk)$-time algorithm may be of lower complexity. We mean an application of the classic technique of Landau and Vishkin [10], where they build a suffix tree with LCA (lowest common ancestor) support over the concatenated sequence $P\#T$ (# is a unique separator), in linear time, and then, for every text character, jump between matching subsequences of P in constant time using LCA queries, hence finding a match or resolving a mismatch in $O(k)$ time per text position. Translating to our problem, after finding each pair of equal substrings, the position in the pattern is shifted by one, while in the text the position is shifted by two, i.e., a single (mismatching) character is skipped.

A similar problem to matching with k-insertions is *episode matching*, which can be stated like that: Find the shortest text substring(s) that contain P as a subsequence. Using our technique, for $k = n - m$, and keeping track of the minimum values, immediately gives $O(n\lceil m\log(w)/w\rceil)$ time complexity as above. This is not always better than the best known algorithms for this problem [11], working in $O(nm/\log(m))$ and $O(n + s + nm\log\log(s)/\log(s/m))$ time, using $O(s)$ additional space, but our algorithm dominates over the former result if m is small enough, namely if $\log(m) = o(w/\log(w))$, and either uses less time or less space than the latter algorithm.

5 Improved ABNDM

We would like to point out that Matryoshka counters can be useful not only for improving the worst cases. To this end, note that in case of a long pattern, a typical implementation of a bit-parallel algorithm searches only for its prefix, such that fits a single computer word, and whenever the prefix is found, the pattern suffix is verified, e.g., with brute-force. For example, this strategy can be used for Shift-Add if $m\ell > w$. For small k and typical texts this results in an $O(n)$ average time algorithm. The same idea can be used together with the technique presented in this paper. Although we do not expect our algorithm for Hamming distance to be competitive in practice, we note that this trick is now usable for larger k values, when we can search for longer prefixes (the ℓ parameter is constant in our algorithm, with no dependence on k), decreasing thus the number of verifications and false positives. The same applies to our (δ, γ)-matching and (δ, α)-matching algorithms, and in general to all the forward-matching algorithms.

However, some algorithms scan the text backwards, using windows of m characters. The text windows are only partially examined on average, and then the window is shifted to the right, in the best case as much as m characters (the maximum shift depending on the actual problem). In this case, cutting the pattern to $\lfloor w/\ell \rfloor$ characters limits the maximum shift to $\lfloor w/\ell \rfloor$ as well. Using more computer words will not improve the overall complexity. Hence using our techniques can improve the average case as well, as we can use longer patterns and obtain longer shifts.

Consider now the Levenshtein distance. In this case we are interested in finding all occurrences of the pattern permitting up to k edit operations, that is, insertions, deletions or substitutions of characters. The average case lower bound of the problem is $\Omega(n(k+\log_\sigma(m))/m)$, and an algorithm with matching upper bound was obtained in [12].

We will improve a bit-parallel algorithm, ABNDM (Approximate Backward Nondeterministic DAWG Matching) [13] that achieves $O(\lceil \frac{m}{w} \rceil \cdot n(k+\log(m))/m)$ average time. Assuming $m = O(w)$, this is optimal up to a $O(\log(\sigma))$ factor and truly optimal if $k = \Omega(\log(m))$ as well. The analysis holds for $k/m < 1/2 - O(1/\sqrt{\sigma})$. We show that using our two-level counter splitting we can obtain $O(\lceil \frac{m}{w} \rceil \cdot n(k + \log\log(m) + \log_\sigma(m))/m)$ average time, optimal if $k = \Omega(\log\log(m))$ or $\log\sigma = \Omega(\log(m)/\log\log(m))$.

The ABNDM algorithm is based on combining the well-known BNDM algorithm [7] and (modified) Myers' bit-parallel dynamic programming algorithm [14]. ABNDM works with text windows of $m-k$ characters. For each window position i, it computes two bit-vectors (among others), V_i^+ and V_i^-, of length m bits. These define an integer vector $C_{i,j}$, whose values are defined to be

$$C_{i,j} \leftarrow C_{i-1,j} + ((V_i^+ >> j) \;\&\; 1) - ((V_i^- >> j) \;\&\; 1). \tag{20}$$

However, the values of $C_{i,j}$ can grow large, and cannot (all) be efficiently maintained explicitly. Still, one of the key ingredients of ABNDM boils down to detecting as soon as possible the situation when all the values in C_i have exceeded k. The solution presented in [13] uses *bit-parallel witnesses*. A witness is some explicitly maintained value of C_i, needing only $\ell = O(\log(m))$ bits. Hence a single

bit-vector of w bits can pack $O(w/\log(m))$ witnesses. These are distributed uniformly, i.e. every ℓ-th value of C_i is explicitly maintained (and $C_{i,m}$ is always maintained). The values can be easily updated in parallel, by masking the V^+ and V^- vectors appropriately. The particularities of the problem guarantee that:

1. No value can be negative.
2. If *all* values of C_i have reached a certain threshold, then they cannot anymore be decreased below that threshold, i.e. every value of C_{i+1} is at least that threshold.
3. The values of two neighboring witnesses can differ by at most $\lfloor \ell/2 \rfloor$;

The two last properties can be used to bound the actual values of C_i: if all witnesses have exceeded $k + \lfloor \ell/2 \rfloor$, then all values of C_i have surely exceeded k. The first two properties can be used to obtain efficient bit-parallel algorithm, also in our case. From the analysis point of view, this is effectively the same as searching with $k' = k + O(\ell)$ errors, and the time bound becomes $O(n(k + \log(m) + \log_\sigma(m))/m)$, for $m = O(w)$.

The $O(\log(m))$ term can be relaxed by using more witnesses, and we can afford using more witnesses by using more counter levels. The solution is similar to that of used for (δ, α)-matching. The main difference is that the counter (witness) values can also *decrease*. Let us again consider the two level counter splitting. For the bottom level we use $\ell' = \Theta(\log\log(m))$ bits per counter, and hence $\lfloor w/\ell' \rfloor$ witnesses. Obviously, the top level needs $\ell = \Theta(\log(m))$ bits per counter, and $\Theta(w\ell/\ell')$ in total. More precisely, we use $\ell' = \lceil \log(\log(m+1)+1) \rceil + 2$ bits per counter in the bottom level, and $\ell = \lceil \log_2(m + 2^{\ell'-1}) \rceil + 1$ for the top level. We note that these could be improved a bit (see the formulas in [13]), but not asymptotically. Consider a bottom level (witness) counter $W'_{[i]\ell'}$ and a top level counter $W_{[i]\ell}$. Call $k' = k + \lfloor \ell'/2 \rfloor$. Then we can proceed as follows:

1. Initialize $W'_{[i]\ell'}$ to 0. This is safe *at first*, since no counter can have a negative value. Initialize $W_{[i]\ell}$ to $2^{\ell-1} - (k'+1)$, which is used to represent 0.
2. Bottom level phase: update the counter $W'_{[i]\ell'}$. This is repeated $2^{\ell-2} - 1$ times. Check if the highest bit of $W'_{[i]\ell'}$ is activated, for all i. If so, terminate the text window scan.
3. Top level phase: Compute $x = W'_{[i]\ell'} - (2^{\ell'-2} - 1)$. If $x > 0$, add x to $W_{[i]\ell}$, and subtract x from $W'_{[i]\ell'}$ (i.e. initialize it to $2^{\ell'-2} - 1$). Otherwise, do nothing. This ensures that the counter cannot underflow (nor overflow) in the next bottom level phase.
4. Adjust $W'_{[i]\ell'}$ and $W_{[i]\ell}$, as described shortly.
5. Go to step 2.

The above scheme correctly maintains the counter values, taking care no underflow or overflow can happen. However, in step 2 we check the highest bits of the bottom level counters, but these never get activated (as step 3 prevents that), unless step 4 adjusts the values appropriately. Step 4 is similar to the cases 1–3 in Sec. 3.1, i.e. the bottom level counters are initialized differently, if the sum of

the two counter levels have exceeded k', or are $2^{\ell'-2}-1$ steps away from doing so. The technique is precisely the same as before. Note that even if the highest bit of $W'_{[i]\ell'}$ is activated, the counter still cannot overflow in $2^{\ell'-2}-1$ steps.

Given the tools we have developed, all the steps can be easily performed bit-parallelly. The actual implementation is similar to that of (δ,α)-matching, hence we do not give the details. The analysis is the same as for the original algorithm, the only difference is that the threshold k' is now $k+\lfloor \ell'/2 \rfloor$, instead of $k+\lfloor \ell/2 \rfloor$. Hence the algorithm runs in $O(n(k+\log\log(m)+\log_\sigma(m))/m)$ average time for $m = O(w)$.

Acknowledgments

We thank an anonymous referee for insightful comments helping to improve the original paper version.

References

1. Grabowski, S., Fredriksson, K.: Bit-parallel string matching under Hamming distance in $O(n\lceil m/w \rceil)$ worst case time. IPL 105(5), 182–187 (2008)
2. Pătraşcu, M.: Predecessor search. In: Kao, M.Y. (ed.) Encyclopedia of Algorithms. Springer, Heidelberg (2008)
3. Baeza-Yates, R.A., Gonnet, G.H.: A new approach to text searching. Communications of the ACM 35(10), 74–82 (1992)
4. Navarro, G.: A guided tour to approximate string matching. ACM Computing Surveys 33(1), 31–88 (2001)
5. Fredman, M.L., Willard, D.E.: Surpassing the information theoretic bound with fusion trees. J. Comput. System Sci. 47, 424–436 (1993)
6. Fredriksson, K., Grabowski, S.: Efficient algorithms for pattern matching with general gaps, character classes and transposition invariance. Information Retrieval 11(4), 335–357 (2008)
7. Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings – Practical online search algorithms for texts and biological sequences, 280 pages. Cambridge University Press, Cambridge (2002)
8. Fredriksson, K., Grabowski, S.: Efficient bit-parallel algorithms for (δ,α)-matching. In: Àlvarez, C., Serna, M. (eds.) WEA 2006. LNCS, vol. 4007, pp. 170–181. Springer, Heidelberg (2006)
9. Kuri, J., Navarro, G., Mé, L.: Fast multipattern search algorithms for intrusion detection. Fundamenta Informaticae 56(1–2), 23–49 (2003)
10. Landau, G.M., Vishkin, U.: Efficient string matching with k mismatches. Theoretical Computer Science 43(2-3), 239–249 (1986)
11. Das, G., Fleischer, R., Gąsieniec, L., Gunopulos, D., Kärkkäinen, J.: Gąsieniec, L., Gunopulos, D., Kärkkäinen, J.: Episode matching. In: Hein, J., Apostolico, A. (eds.) CPM 1997. LNCS, vol. 1264, pp. 12–27. Springer, Heidelberg (1997)
12. Chang, W., Marr, T.: Approximate string matching with local similarity. In: Crochemore, M., Gusfield, D. (eds.) CPM 1994. LNCS, vol. 807, pp. 259–273. Springer, Heidelberg (1994)
13. Hyyrö, H., Navarro, G.: Bit-parallel witnesses and their applications to approximate string matching. Algorithmica 41(3), 203–231 (2005)
14. Myers, G.: A fast bit-vector algorithm for approximate string matching based on dynamic programming. Journal of the ACM 46(3), 395–415 (1999)

Bounded Delay and Concurrency for Earliest Query Answering

Olivier Gauwin[1,2,3], Joachim Niehren[1,3], and Sophie Tison[2,3]

[1] INRIA, Lille
[2] University of Lille 1
[3] Mostrare project, INRIA & LIFL (CNRS UMR8022)

Abstract. Earliest query answering is needed for streaming XML processing with optimal memory management. We study the feasibility of earliest query answering for node selection queries. Tractable queries are distinguished by a bounded number of concurrently alive answer candidates at every time point, and a bounded delay for node selection. We show that both properties are decidable in polynomial time for queries defined by deterministic automata for unranked trees. Our results are obtained by reduction to the bounded valuedness problem for recognizable relations between unranked trees.

1 Introduction

Streaming algorithms are relevant for XML databases and data exchange, whenever large data collections that cannot be stored in main memory are to be processed. Instead data is communicated over streams and processed incrementally. Recently, XML streaming algorithms were proposed for schema validation [1] (membership in tree languages), one-pass typing [2] (annotating nodes of trees by types), and query answering [3,4,5].

The space complexity of streaming algorithms for answering node selection queries in XML trees depends on the size of the call stack (bounded by the depth of the tree) and on the number of concurrently alive answer candidates that are kept in main memory at every time point [6]. The purpose of earliest query answering (EQA) is to minimize the second number. Selection and unselection of answer candidates need to be decided as early as possible, so that selected candidates can be output and unselected candidates discarded as early as possible. In both cases they are removed from main memory.

EQA is the objective of various streaming algorithms for Forward XPath and its fragments [6,7,8], and has been studied for automata defined queries too [9,10]. In the latter paper, it is shown how to obtain a correct EQA algorithm for a fragment of Forward XPath under schema assumptions, by a P-time translation to deterministic streaming tree automata (dSTAs) [11], or equivalently deterministic nested word [12], visibly pushdown [13] or pushdown forest automata [14].

Whether EQA is tractable depends on two properties of the considered query Q and schema S, both of which are independent of the concrete algorithm. The first restriction is *bounded delay of selection*, which requires a bound for all trees $t \in S$ on the number of events between the first visit of a selected node $\pi \in Q(t)$

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 350–361, 2009.

and the earliest event that permits its selection. This limits the waiting time for the answer. The second restriction is *bounded concurrency of alive candidates*, which imposes a bound on the number of concurrently alive answer candidates for Q wrt. S for all trees of S at all time points. This limits the maximal number of candidates that need to be memoized simultaneously by every EQA algorithm.

In this paper, we show that bounded delay and bounded concurrency are decidable in P-time for n-ary queries and schemas defined by deterministic automata for unranked trees. Our result holds for dSTAs, as well as for bottom-up deterministic tree automata that operate on binary encodings of unranked trees [15], either firstchild-nextsibling based or Curried [16]. When restricting databases to words instead of trees, decision procedures for bounded delay and concurrency for queries defined by dFAs can be obtained quite easily by reduction to bounded ambiguity of nFAs. The algorithm for words, however, cannot be lifted to trees in any straightforward manner. In order to solve this problem, we propose another solution by reduction to the bounded valuedness problem of recognizable relations between unranked trees. As we show, this problem can be reduced to bounded valuedness of bottom-up tree transducers, which can be decided in P-time (Theorem 2.8 of [17]). All reductions are in P-time since we start from deterministic automata. *Omitted proofs can be found in the long version.*

2 Queries in Words

We recall definitions of schemas and queries for tuples of positions in words by dFAs, and the concept of bounded ambiguity for nFAs.

Words, Positions, and Events. Let an alphabet Σ be a finite set with elements ranged over by a, b, c and $\mathbb{N}$ the set of natural numbers $n \geq 1$. We will consider words $w \in \Sigma^+$ as databases. The set of all words Σ^* is closed under concatenation ww' and contains the empty word ϵ. The set of positions of a word $w \in \Sigma^*$ is $pos(w) = \{1, \ldots, |w|\}$ where $|w|$ is the number of letters of w. For all $w \in \Sigma^+$ and positions $\pi \in pos(w)$ we define $lab^w(\pi) \in \Sigma$ to be the π-th letter of word w and say that position π is labeled by $lab^w(\pi)$.

One-way finite automata process words letter by letter from the left to the right, equally to streaming algorithms for words. We define the set of events for a word $w \in \Sigma^+$ by adding the start event 0 to the set of all positions $eve(w) = \{0\} \cup pos(w)$. For all events $e \in eve(w)$, we define $w^{\leq e} \in \Sigma^*$ to be the prefix of w with exactly e letters. We say that two words coincide until event e if $w^{\leq e} = w'^{\leq e}$ and write $eq_e(w, w')$ in this case.

Queries and Schemas. An *n-ary query* Q selects a set of n-tuples of positions for every word $w \in \Sigma^+$. It is a function which maps words w to sets of tuples of positions $Q(w) \subseteq pos(w)^n$. A *schema* $S \subseteq \Sigma^+$ restricts the set of permitted databases. We say that a word $w \in \Sigma^+$ satisfies a schema S if and only if $w \in S$.

Automata, Ambiguity, and Determinism. A *finite automaton* (nFA) over Σ is a tuple $A = (stat, init, rul, fin)$ where $init, fin \subseteq stat$ are finite sets and $rul \subseteq stat^2 \times (\Sigma \cup \{\epsilon\})$ contains rules that we write as $q \xrightarrow{a} q'$ or $q \xrightarrow{\epsilon} q'$

where $q, q' \in \mathit{stat}$ and $a \in \Sigma$. Whenever necessary, we will index the components of A by A. Let the size of A count all states and rules, *i.e.*, $|A| = |\mathit{stat}_A| + |\mathit{rul}_A|$.

A run of A on a word w is a function $r : \mathit{eve}(w) \to \mathit{stat}_A$ so that $r(0) \in \mathit{init}_A$ and $r(\pi{-}1) \xrightarrow{\epsilon^*} \xrightarrow{a} \xrightarrow{\epsilon^*} r(\pi)$ is justified by rul for all $\pi \in \mathit{pos}(w)$ with $a = \mathit{lab}^w(\pi)$. A run is successful if $r(|w|) \in \mathit{fin}_A$. The language $L(A) \subseteq \Sigma^*$ is the set of all words that permit a successful run by A. An nFA is called *productive*, if all its states are used in some successful run. This is the case if all states are reachable from some initial state, and if for all states, some final state can be reached.

The *ambiguity* $\mathit{amb}_A(w)$ is the number of successful runs of A on w. The ambiguity of A is *k-bounded* if $\mathit{amb}_A(w) \leq k$ for all $w \in \Sigma^*$. It is *bounded*, if it is bounded by some k. An nFA is *deterministic* or a dFA if it has at most one initial state and at most one rule for all left hand sides, including letters. Clearly the ambiguity of dFAs is 1-bounded.

Stearns and Hunt [18] present a P-time algorithm for deciding k-bounded ambiguity of nFAs. Let us write $p \xrightarrow{w} q$ *by* A if there exists a run of $A[\mathit{init} = \{p\}]$ (the automaton obtained from A by setting its initial states to $\{p\}$) on w that ends in q. Weber and Seidl [19] show that an nFA A has unbounded ambiguity iff there exists a word $w \in \Sigma^+$ and distinct states $p \neq q$ such that $p \xrightarrow{w} p$, $p \xrightarrow{w} q$, $q \xrightarrow{w} q$ by A. This can be tested in $O(|A|^3)$ as shown very recently by [20].

Canonicity and Types. Let $\mathbb{B} = \{0, 1\}$ be the set of Booleans. For words $w \in \Sigma^*$ and tuples $\tau = (\pi_1, \ldots, \pi_n) \in \mathit{pos}(w)^n$, let $w * \tau$ be the annotated word in $(\Sigma \times \mathbb{B}^n)^*$ obtained from w by relabeling all positions $\pi \in \mathit{pos}(w)$ to $(\mathit{lab}^w(\pi), b_1, \ldots, b_n)$, where $b_i = 1$ if $\pi = \pi_i$ and $b_i = 0$ otherwise. The *canonical language of an n-ary query* Q is $\mathit{can}_Q = \{w * \tau \mid \tau \in Q(w)\}$. The type of a word $v \in (\mathbb{B}^n)^*$ is an element of $(\mathbb{N} \cup \{0\})^n$ defined by $\sum_{\pi \in \mathit{pos}(v)} \mathit{lab}^v(\pi)$. The type of a word in $(\Sigma \times \mathbb{B}^n)^*$ is the type of its projection in $(\mathbb{B}^n)^*$. All words over $\Sigma \times \mathbb{B}^n$ of type 1^n have the form $w * \tau$, and vice versa. An nFA A over $\Sigma \times \mathbb{B}^n$ is called *canonical* if all words of $L(A)$ have type 1^n. For productive canonical dFAs A, every state $q \in \mathit{stat}$ has a unique type in $\mathbb{B}^n$, which is the type of all words with runs by A ending in q (e.g., Lemma 3 of [21]).

3 Earliest Query Answering for Words

We recall the framework of EQA for XML databases from [10], but restricted to the case of words, and show for queries defined by dFAs that bounded concurrency and delay can be reduced in P-time to bounded ambiguity for dFAs.

An EQA algorithm decides selection and failure of answer candidates at every time point (without knowing the rest of the stream). This way, it needs to keep in main memory only alive candidates, which are neither safe for selection nor failure. As an example, consider the monadic query Q that selects all positions in words w that are labeled by a and followed by bb. When applied to $w = \mathit{aabbabbcabab}$, this query returns $Q(w) = \{(2), (5)\}$. A streaming algorithm can enumerate these answers by using a sliding window of length 3. Position 1 for instance can be refused when having seen the labels of positions $1, 2$, while position 2 can be selected when having seen the labels of positions $2, 3, 4$.

Schema assumptions are relevant to EQA since restricting the remainder of the stream. The schema $(a|b)^*c(ab)^*$, for instance, excludes all positions from $Q(w)$ that are on the right of the c letter in w. This allows to exclude positions $8, \ldots, 12$ to belong to the answer set $Q(w)$ immediately at the respective position.

Earliest Selection and Bounded Delay. The delay of a selected position is the number of subsequent events before selection can be safely decided. More formally, let Q be an n-ary query in words $w \in \Sigma^+$ satisfying a schema $S \subseteq \Sigma^+$. We define a relation $sel_Q^S(w)$ that links tuples $\tau \in pos(w)^n$ to events $e \in eve(w)$ that are sufficient for selection, *i.e.*, where τ will be selected in all possible continuations of the stream beyond e. Only those continuations are allowed, which extend the current prefix of the word to a member of S:

$$(\tau, e) \in sel_Q^S(w) \;\Leftrightarrow\; \tau \in \{1, \ldots, e\}^n \wedge \forall w' \in S.\ eq_e(w, w') \Rightarrow \tau \in Q(w')$$

Note that the initial event 0 may be sufficient to select the empty tuple () in Boolean queries where $n = 0$, while it is never sufficient for selection if $n \geq 1$ since otherwise $\tau \notin \{1, \ldots, e\}^n$.

Let $latest((\pi_1, \ldots, \pi_n)) = max_{1 \leq i \leq n} \pi_i$ be the latest position of the tuple. The delay of an n-ary query Q for a tuple $\tau \in pos(w)$ is the number of events e following $latest(\tau)$ such that e is insufficient for selection, *i.e.*, $(\tau, e) \notin sel_Q^S(w)$.

$$delay_Q^S(w, \tau) = |\{e \in eve(w) \mid\ latest(\tau) \leq e,\ (\tau, e) \notin sel_Q^S(w)\}|$$

Query Q with schema S has *k-bounded delay* if $delay_Q^S(w, \tau) \leq k$ for all $w \in S$ and $\tau \in Q(w)$. It has bounded delay if it has k-bounded delay for some $k \geq 0$. Having bounded delay means that every EQA algorithm will output selected tuples a constant time after completion.

Deciding Bounded Delay. We start with the case without schemas. Let A be a canonical and productive dFA over $\Sigma \times \mathbb{B}^n$ and Q_A the n-ary query that it defines. We call a state $q \in stat_A$ of type 1^n *safe for selection* if $(\Sigma \times \{0\}^n)^* \subseteq L(A[init = \{q\}])$. Since A is canonical and deterministic, this is the case if and only if all states reachable from q are final and have outgoing transitions for all letters in $\Sigma \times \{0\}^n$. Thus, the set of states of A that are safe for selection can be computed in time $O(|A| + |\Sigma| + n)$.

Lemma 1. *If the run r of a canonical dFA A on $w * \tau$ exists then it satisfies for all $e \in pos(w)$ that $r(e)$ is safe for selection if and only if $(\tau, e) \in sel_{Q_A}(w)$.*

We define an nFA $D(A)$ such that $amb_{D(A)}(w * \tau) = delay_{Q_A}(w, \tau)$ for all $\tau \in Q_A(w)$. $D(A)$ has the same states as A except for one additional state *ok*, which is the only final state of $D(A)$. All transitions of A are preserved by $D(A)$. In addition to simulating A, automaton $D(A)$ has ϵ transitions into the state *ok* from all states q of type 1^n of A that are unsafe for selection. State *ok* has transitions into itself for all letters in $\Sigma \times \{0\}^n$.

Proposition 1. *For $w \in \Sigma^*$, $\tau \in Q_A(w)$: $delay_{Q_A}(w, \tau) = amb_{D(A)}(w * \tau)$.*

Proof. Consider a run of $D(A)$ on a canonical word $w * \tau$. The only ambiguity of $D(A)$ is introduced by the ϵ-transitions, by which to exit from A at positions

between the last component of τ (included) and the earliest event that is safe for the selection of τ. The number of such positions is precisely $delay_{Q_A}(w, \tau)$.

Theorem 1. *Bounded delay for queries Q_A and schemas $L(B)$ defined by dFAs A, B can be decided in time $O(|A| \cdot |B|)$, and k-bounded delay in P-time.*

Proof. We sketch the proof for bounded delay without schemas, where $L(B) = \Sigma^*$. By Proposition 1, it is sufficient to decide whether $D(A)$ has bounded ambiguity. By Weber and Seidl's characterization, this holds if the subautomaton of A containing only unsafe states for selection of type 1^n, has no loop. Acyclicity of this subautomaton can be tested in time $O(|A|)$.

Earliest Failure and Bounded Concurrency. The space complexity of EQA algorithms depends on the concurrency of a query, which is the maximal number of concurrently alive answer candidates at every time point [6,7], since these are to be kept in main memory. In order to formalize this for n-ary queries, we have to deal with *partial* answer candidates for a given word w. We fix a constant $\bullet$ that represents unknown components, and define partial tuples τ of positions until $e \in pos(w)$ as members of $(\{1, \ldots, e\} \uplus \{\bullet\})^n$. So far, we have only studied *complete* answer candidates, which do not contain any unknown component. We write $compl(\tau, w, e)$ for the set of complete candidates, in which all unknown components of τ have been instantiated with nodes $\pi \in pos(w)$ such that $e \leq \pi$. Given a query Q, schema S, and word $w \in S$, we call a partial candidate τ *failed at event* $e \in eve(w)$, if no completion of τ by nodes in the future of e is selected by Q.

$$(\tau, e) \in fail^S_Q(w) \Leftrightarrow \begin{cases} \tau \in (\{1, \ldots, e\} \uplus \{\bullet\})^n \wedge \\ \forall w' \in S.\ eq_e(w, w') \Rightarrow \forall \tau' \in compl(\tau, w', e).\ \tau' \notin Q(w') \end{cases}$$

A partial candidate $\tau \in (\{1, \ldots, e\} \cup \{\bullet\})^n$ is *alive* at e if it is neither failed nor selected at e. The concurrency of a query schema pair on a word $w \in \Sigma^+$ at position $e \in eve(w)$ is the number of alive candidates at time point e, so that e is neither sufficient for selection or failure:

$$(\tau, e) \in alive^S_Q(w) \Leftrightarrow\ (\tau, e) \notin fail^S_Q(w) \text{ and } (\tau, e) \notin sel^S_Q(w)$$
$$concur^S_Q(w, e) = |\{\tau \in (\{1, \ldots, e\} \cup \{\bullet\})^n \ \mid\ (\tau, e) \in alive^S_Q(w)\}|$$

We say that the concurrency of a query schema pair is bounded if there exists $k \geq 0$ such that $concur^S_Q(w, e) \leq k$ for all words $w \in S$ and $e \in pos(w)$.

Deciding Bounded Concurrency. We start with queries without schemas. So let A be a canonical productive dFA over $\Sigma \times \mathbb{B}^n$ and Q_A the query it defines. Recall that all states of A have a unique type $v \in \mathbb{B}^n$. We call a state q of type v *safe for failure*, if no final states can be reached from q by words of complementary type $1^n - v$. By canonicity, this is the case if no final states can be reached from q at all. We can thus compute the set of safe states for failure in time $O(|A|)$.

Lemma 2. *If the unique run r of a canonical dFA A on some $w * \tau$ exists, then all $e \in pos(w)$ satisfy that $r(e)$ is safe for failure if and only if $(\tau, e) \in fail_{Q_A}(w)$.*

We define an nFA $C(A)$ such that $amb_{C(A)}(w * e) = concur_Q(w, e)$ for all $e \in pos(w)$. The situation is a little different than for $D(A)$ since $C(A)$ runs on words

annotated by events rather than tuples. So the alphabet of $C(A)$ is $\Sigma \times \mathbb{B}$. The nFA $C(A)$ guesses all partial candidates with positions until e, tests whether they are alive at e, and accepts in this case and only this case.

Proposition 2. *For all $e \in pos(w)$: $concur_{Q_A}(w, e) = amb_{C(A)}(w * e)$.*

Theorem 2. *Bounded and k-bounded concurrency for queries and schemas defined by canonical dFAs can be decided in P-time for any fixed $k \geq 0$.*

4 Earliest Query Answering for Unranked Trees

We extend EQA from words to unranked trees. We then lift our P-time decision results to tree automata for unranked trees, and argue why the proofs for words cannot be lifted in any straightforward manner.

The set of *unranked trees* T_Σ is the least set that contains all (k+1)-tuples $a(t_1, \ldots, t_k)$ where $k \geq 0$, $a \in \Sigma$ and $t_1, \ldots, t_k \in T_\Sigma$. Positions of words correspond to nodes of trees, defined by $nod(a(t_1, \ldots, t_k)) = \{\epsilon\} \cup \{i\pi \mid \pi \in nod(t_i)\}$. The word $w = abaca$, for instance, can be encoded by the tree $t = r(a, b, a, c, a)$, where $r \in \Sigma$ is an arbitrary symbol. Note that $nod(t) = \{\epsilon\} \cup pos(w)$. Queries Q in unranked trees select tuples of nodes $Q(t) \subseteq nod(t)^n$ for all trees $t \in T_\Sigma$. Events are produced by preorder traversals:

$$eve(t) = \{\texttt{start}\} \cup (\{\texttt{open}, \texttt{close}\} \times nod(t))$$

There is an initial event `start` and an opening and a closing event per node. Let $\leq$ be the total order on $eve(t)$ induced by preorder traversals over trees $t \in T_\Sigma$, and let $\texttt{pred}(e)$ be the immediate predecessor of event $e \in eve(t) - \{\texttt{start}\}$. For all $e \in eve(t) - \{\texttt{start}\}$, we define the prefix $t^{\leq e}$ of t to be the tree which contains the part of t with all nodes opened before e, *i.e.*, $nod(t^{\leq e}) = \{\pi \in nod(t) \mid (\texttt{open}, \pi) \leq e\}$, and $lab^{t^{\leq e}}(\pi) = lab^t(\pi)$ for all $\pi \in nod(t^{\leq e})$. Note that $t^{\leq(\texttt{close},\pi)}$ contains all proper descendants of π in t, while $t^{\leq(\texttt{open},\pi)}$ does not. As before, we can define $eq_e(t, t')$ by $e = \texttt{start}$ or $t^{\leq e} = t'^{\leq e}$. The notion $t * \tau \in T_{\Sigma \times \mathbb{B}^n}$ extends straightforwardly from words to trees. The canonical language of an n-ary query Q thus has type $can_Q \subseteq T_{\Sigma \times \mathbb{B}^n}$. The definitions of sel^S_Q, $fail^S_Q$, $delay^S_Q$ and $concur^S_Q$ extend literally, except that the set $\{1, \ldots, e\}$ needs to be replaced by $nod(t^{\leq e})$.

Tree automata for unranked trees are often obtained from standard tree automata for binary trees. A binary signature is a finite set $\Gamma = \Gamma_2 \uplus \Gamma_0$ with constants in Γ_0 and binary function symbols in Γ_2. The set of binary trees T^{bin}_Γ is the least set containing all $c \in \Gamma_0$ and triples $f(t_1, t_2)$ where $f \in \Gamma_2$ and $t_1, t_2 \in T^{\text{bin}}_\Gamma$. A tree automaton (TA) for binary trees in T^{bin}_Γ is a tuple $A = (stat, fin, rul)$ consisting of finite sets $fin \subseteq stat$ and a set $rul \subseteq \cup_{i \in \{0,2\}} stat^{i+1} \times \Gamma_i$, that we denote as $f(q_1, q_2) \to q$ and $c \to q$ where $q_1, q_2, q \in stat$, $f \in \Gamma_2$ and $c \in \Gamma_0$. A run of A on $t \in T^{\text{bin}}_\Gamma$ is a function $r : nod(t) \to stat$ such that $f(r(\pi 1), r(\pi 2)) \to r(\pi)$ belongs to rul_A for all nodes π of t with $lab^t(\pi) = f \in \Gamma_2$, and $r(\pi) \to c$ in rul_A for all nodes π of t with $lab^t(\pi) = c \in \Gamma_0$. The language $L^{\text{bin}}(A)$ is the set

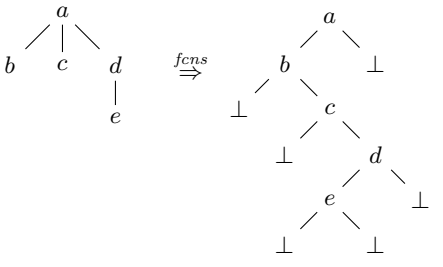


Fig. 1. Binary encoding

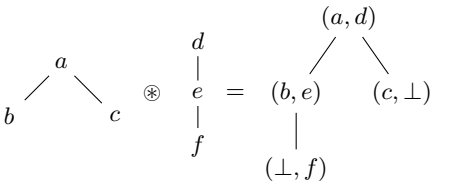


Fig. 2. Example for overlays

of all binary trees over Γ that permit a successful run by A, where $r(\epsilon) \in \mathit{fin}$. A (bottom-up) deterministic TA (dTA) is a TA of which no two rules have the same left hand side.

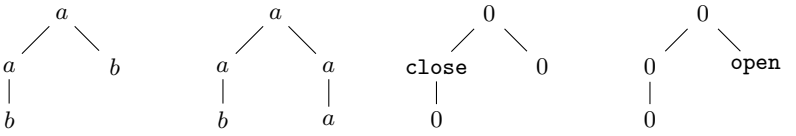
We can encode unranked trees $t \in T_\Sigma$ into binary trees by applying Rabin's firstchild-nextsibling encoding $\mathit{fcns} : T_\Sigma \to T^{\mathrm{bin}}_{\Sigma_\perp}$ where $\Sigma_\perp = \Sigma \uplus \{\perp\}$. The definition is recalled by example in Fig. 1. A TA over $T_{\Sigma_\perp}$ defines the language of unranked trees $L(A) = \{t \in T_\Sigma \mid \mathit{fcns}(t) \in L^{\mathrm{bin}}(A)\}$.

Operationally, however, dTAs fail to operate in streaming manner on unranked trees, so that the previous decision algorithms cannot be lifted to queries defined by dTAs. Streaming tree automata (STAs) [11] operate in the proper order. They are a reformulation of nested word automata [12,13] and shown equivalent to pushdown forest automata [14]. Deterministic STAs (dSTAs) can perform one-pass typing for extended DTDs with restrained competition [2] as well as EQA [10] for queries defined by dSTAs. Furthermore, deterministic stepwise tree automata [16] can be converted in dSTAs in linear time.

Proposition 3 (Closure properties). *The classes of TAs (wrt. the fcns encoding) and STAs permit determinization, and recognize the same languages of unranked trees modulo P-time automata translations (not preserving determinism). Recognizable languages are closed under Boolean operations, projection and cylindrification. All corresponding operations on TAs (resp. STAs) can be performed in P-time and preserve determinism except for projection.*

Even with STAs, it remains difficult to lift our P-time algorithms for words to trees, since the notion of safe states becomes more complex. Given a canonical dSTA A for query Q_A, one can define another dSTA $E(A)$ for which appropriate notions of safe states wrt. Q_A exist [10]. The size of $E(A)$, however, may grow exponentially in $|A|$. Therefore, we cannot use $E(A)$ to construct polynomially sized counterparts of $D(A)$ and $C(A)$ in the case of unranked trees. Nevertheless:

Theorem 3 (Main). *Bounded delay is decidable in P-time for n-ary queries and schemas in unranked trees defined by dTAs (wrt. the fcns or Curried encoding) or dSTAs, where n may be variable. Bounded concurrency is decidable in P-time for fixed n. For fixed k and n, k-bounded delay and concurrency are decidable in NP-time.*



(a) Tree t (b) Tree s (c) Tree s_e (d) Tree $s_{e'}$

Fig. 3. $(t, s, s_e) \in Eq$ but $(t, s, s_{e'}) \notin Eq$

Our proof will be based on the powerful notion of recognizable relations between unranked trees (see [15] for ranked trees). Bounded delay and concurrency are reduced to bounded valuedness of recognizable relations, which in turn is reduced to bounded valuedness of tree transducers for binary trees [17].

5 Recognizable Relations between Unranked Trees

We extend the theory of recognizable relations from ranked to unranked trees. We show that FO-formulas over recognizable relations with n free variables define recognizable relations between n unranked trees (so that satisfiability is decidable), and that bounded valuedness of recognizable relations can be decided in P-time by reduction to bounded valuedness of tree transducers (for binary trees).

Recognizable Relations. In this section, we assume an arbitrary class of tree automata, that satisfy the properties of STAs in Proposition 3. This includes TAs modulo the *fcns* encoding, STAs, and stepwise tree automata [16] (but not deterministic hedge automata with dFAs for horizontal languages [15]).

The *overlay* of k unranked trees $t_i \in T_{\Sigma^i}$ is the unranked tree $t_1 \circledast \ldots \circledast t_k$ in $T_{\Sigma_\perp^1 \times \ldots \times \Sigma_\perp^k}$ obtained by superposing these k trees top-down and left-to-right; the $\perp$ symbol represent missing children where the structures of the trees differ. This is illustrated in Fig. 2. Overlays of ranked trees can be obtained this way too [15], except that overlayed symbols need to inherit the maximal arity. A k-ary relation R between unranked trees is *recognizable* iff the language of its overlays $ovl(R) = \{t_1 \circledast \ldots \circledast t_k \mid (t_1, \ldots, t_k) \in R\}$ is recognizable by a tree automaton. We say that R is recognized by the automaton A if $ovl(R) = L(A)$. We also say that R can be computed in time k if an automaton recognizing R can be computed in time k.

The prime example is the relation $Eq \subseteq T_\Sigma \times T_\Sigma \times T_{\{0,\texttt{open},\texttt{close}\}}$. Here, we map event $e = (\alpha, \pi) \in eve(t)$ to trees $ren^e(t) \in T_{\{0,\texttt{open},\texttt{close}\}}$ obtained by relabeling t, such that π is relabeled to α and all other nodes to 0. We then define $Eq = \{(t, s, ren^e(t)) \mid eq_e(t, s)\}$. See Fig. 3 for an example.

Lemma 3. *Given a signature Σ, a deterministic automaton recognizing relation $Eq \subseteq T_\Sigma \times T_\Sigma \times T_{\{0,\texttt{open},\texttt{close}\}}$ can be computed in time $O(|\Sigma|^2)$.*

An STA recognizing $ovl(Eq)$ with $O(1)$ states is easy to define. It can be converted into a TA by Proposition 3 and from there to a deterministic automaton of the class under consideration by assumption. The resulting automaton still has $O(1)$ states, and thus an overall size of $O(|\Sigma|)$, if we assume in addition a function ψ such that $|A| \leq |\Sigma| \cdot \psi(|stat_A|)$ for all automata A with signature Σ.

FO Logic. Let Ω be a collection of unranked disjoint signatures and $\Re$ a set of recognizable relations between unranked trees, so that each relation $R \in \Re$ has a type $R \subseteq T_{\Sigma_1^R} \times \ldots \times T_{\Sigma_{ar(R)}^R}$ where $\Sigma_1^R, \ldots, \Sigma_{ar(R)}^R \in \Omega$ and $ar(R) \geq 0$. We fix an infinite set of variables V ranging over unranked trees. A FO formula over recognizable relations in $\Re$ and signatures in Ω has the abstract syntax:

$$\phi \ ::= \ R(X_1, \ldots, X_{ar(R)}) \mid \phi \wedge \phi' \mid \neg\phi \mid \exists X \in T_\Sigma.\ \phi$$

where $R \in \Re$, $X_1, \ldots, X_{ar(R)} \in V$, and $\Sigma \in \Omega$. We assume that all formulas are well-typed, *i.e.*, that the types of variables are compatible with those of the relations in which they appear. A formula ϕ with m free variables $X_1, \ldots, X_m$ of types $T_{\Sigma_1}, \ldots, T_{\Sigma_m}$ defines an m-ary relation $R_{\phi(X_1,\ldots,X_m)} \subseteq T_{\Sigma_1} \times \ldots \times T_{\Sigma_m}$. The closure properties of tree automata ensure that all such relations are recognizable. Let $FO_\exists[\Re]$ be the set of formulas where existential quantifiers are restricted to occur outermost (and Ω is the set of signatures appearing in the types of relations in $\Re$). The size $|\phi|$ is the number of nodes of ϕ.

Proposition 4. *Let $\Re$ be a finite set of relations recognized by deterministic automata $\{A_R\}_{R\in\Re}$. Then there exists a polynomial p such that for all formulas ϕ in $FO_\exists[\Re]$, an automaton recognizing the relation defined by ϕ can be computed in time $p(|\phi|, (|A_R|)_{R\in\Re})$.*

Bounded Valuedness. Let $R \subseteq T_{\Sigma_1} \times T_{\Sigma_2}$ be a recognizable binary relation. For every $s_1 \in T_{\Sigma_1}$, the number $\#R(s_1) = |\{s_2 \mid (s_1, s_2) \in R\}|$ counts the trees in T_{Σ_2} in relation to it. The *valuedness* of R is the maximal such number $val(R) = \max_{s\in T_{\Sigma_1}} \#R(s)$. We call R *bounded* if $val(R) \leq k$ for some $k \geq 0$.

Lemma 4. *A relation R between unranked trees is recognizable iff the corresponding relation between binary trees $fcns(R)$ is, and $val(fcns(R)) = val(R)$.*

Theorem 4. *For every automaton A recognizing a binary relation R between unranked trees, i.e., $L(A) = ovl(R)$:*

1. *$val(R) < \infty$ can be decided in P-time in $|A|$.*
2. *$val(R) < k$ (for a fixed k) can be decided in NP-time in $|A|$.*

Proof. It is sufficient to prove the theorem for TAs over binary trees by Lemma 4. We start with recognizable relabeling relations, and lift the result to recognizable relations in the long version of the paper. A relabeling relation $R \subseteq T_{\Sigma^1} \times \ldots \times T_{\Sigma^n}$ is a relation between trees of the same structure, *i.e.*, whenever $(s_1, \ldots, s_n) \in R$ then $nod(s_1) = \ldots = nod(s_n)$. In other words, the overlays in $ovl(R)$ do not contain any place holder $\perp$.

So let $R \subseteq T_\Sigma \times T_\Delta$ be a relabeling relation for binary signatures, and A a TA for trees in $T_{\Sigma\times\Delta}$ that recognizes R, *i.e.*, $L^{bin}(A) = ovl(R)$. We transform A into a bottom-up tree transducer T for defining the relation R as in [22]. The rules of T are infered as follows where x_1, x_2 are variables:

$$\frac{(f,g)(p_1,p_2) \to p \in rul(A)}{f(p_1(x_1), p_2(x_2)) \to p(g(x_1,x_2)) \in rul(T)} \qquad \frac{(a,b) \to p \in rul(A)}{a \to p(b) \in rul(T)}$$

This transducer T has the same valuedness as R. Theorem 2.8 of [22] shows that it can be decided in polynomial time whether T is finite-valued, *i.e.*, whether R is bounded. Concerning k-valuedness, it can be decided in non-deterministic polynomial time according to Theorem 2.2 of [22].

The polynomials for testing bounded valuedness of tree transducers are much higher than for testing bounded ambiguity for tree automata [23].

Using the above constructions and Theorem 2.7 of [22], we can build an algorithm for computing the exact value of $val(R)$, if it exists. We can proceed by dichotomy, starting from $2^{2^{P(|A|)}}$, for a fixed polynomial P.

From Proposition 4, we get in P-time a non-deterministic automaton recognizing a relation defined by an $FO_\exists[\Re]$ formula, and then apply Theorem 4:

Corollary 1. *Let $\Re$ be a finite set of relations and A_R deterministic automata recognizing $R \in \Re$. Then there exists a polynomial p such that for formulas ϕ in $FO_\exists[\Re]$, the bounded valuedness $val(R_\phi) < \infty$ of the relation R_ϕ defined by ϕ can be decided in time $p(|\phi|, (|A_R|)_{R\in\Re})$.*

6 Deciding Bounded Delay and Concurrency

We prove the main Theorem 3 on deciding bounded delay and concurrency by reduction to Corollary 1 on recognizable relations.

Let Q be an n-ary query for trees in T_Σ and $S \subseteq T_\Sigma$ a schema. We define a relation $Can_Q = \{(t, ren^\tau(s)) \mid t * \tau \in can_Q \ \wedge \ Eq(t, s, ren^{latest(\tau)}(t))\}$, where $ren^\tau(s)$ is the projection of $s*\tau$ to $\mathbb{B}^n$. The relation $Bef = \{(t, ren^\tau(t), ren^e(t)) \mid \tau \in nod(t^{\leq e})^n\}$ is recognizable by a dTA of size $O(2^n)$, so we cannot use this relation for P-time algorithms without fixing n. By using the relation $Bef\&Can_Q = \{(t, s_\tau, s_e) \mid Can_Q(t, s_\tau)$ and $Bef(t, s_\tau, s_e)\}$, the problem can sometimes be circumvented. Given a deterministic automaton defining Q (it can be a TA on *fcns* encoding, a stepwise tree automaton or an STA), one can construct an automaton of polynomial size recognizing the relation $Bef\&Can_Q$.

Our objective is to define the formulas $delay_Q^S$ and $concur_Q^S$ in the logic $FO_\exists(Eq, Can_Q, S, Bef, Bef\&Can_Q)$, preferably without using Bef. We start with defining relation $Sel_Q^S = \{(t, ren^\tau(t), ren^e(t)) \mid (\tau, e) \in sel_Q^S(t)\}$ by an FO formula $Sel_Q^S(X_t, X_\tau, X_e)$ with three free variables:

$$Sel_Q^S(X_t, X_\tau, X_e) \ =_{df} \ S(X_t) \ \wedge \ Bef(X_t, X_\tau, X_e) \\ \wedge \forall X'_t \in T_\Sigma.\ (S(X'_t) \ \wedge \ Eq(X_t, X'_t, X_e)) \ \Rightarrow \ Can_Q(X'_t, X_\tau)$$

Given automata defining Q and schema S, we can thus define an automaton recognizing $Sel_Q^S(X_t, X_\tau, X_e)$. This yields an algorithm for deciding judgments $(\tau, e) \in sel_Q^S(t)$. It may be unefficient, though, since the automaton obtained this way may be huge, given that formula $Sel_Q^S(X_t, X_\tau, X_e)$ uses full FO-logic of recognizable relations without restriction to some $FO_\exists$.

Bounded Delay. We define the relation $Delay_Q^S = \{(t, ren^\tau(t), ren^e(t)) \mid e \in delay_Q^S(t, \tau)\}$ by the following formula of $FO_\exists(Eq, Bef\&Can_Q, S)$:

$$Delay^S_Q(X_t, X_\tau, X_e) \;=_{df}\; \exists X'_t \in T_\Sigma.\; S(X_t) \;\wedge\; Bef\&Can_Q(X_t, X_\tau, X_e) \\ \wedge\; S(X'_t) \;\wedge\; Eq(X_t, X'_t, X_e) \;\wedge\; \neg Can_Q(X'_t, X_\tau)$$

All base relations can be defined by deterministic automata of polynomial size when leaving n variable (since we don't need relaton *Bef* here). Given deterministic automata A and B defining query Q and schema $S = L(B)$, we can thus define a possibly nondeterministic automaton recognizing $Delay^S_Q(X_t, X_\tau, X_e)$ in P-time from A and B. Let $2Delay^S_Q = \{(t \circledast s_\tau, s_e) \mid Delay^S_Q(t, s_\tau, s_e)\}$. Both relations are recognized by the same automaton. This relation exactly captures the delay: $val(2Delay^S_Q) = |delay^S_Q|$. Applying Corollary 1 to $2Delay^S_Q$ proves that bounded delay is decidable in P-time.

Bounded Concurrency. For concurrency, we proceed in a similar manner. The relation $Alive^S_Q = \{(t, ren^\tau(t), ren^e(t)) \mid \tau \in alive^S_Q(t, e)\}$ can be defined by the following formula of $FO_\exists$:

$$Alive^S_Q(X_t,X_e,X_\tau) = \exists X'_t{\in}T_\Sigma.\exists X''_t{\in}T_\Sigma.\exists X'_\tau{\in}T_{\mathbb{B}^n}.\exists X''_\tau{\in}T_{\mathbb{B}^n}.\; S(X'_t){\wedge}S(X''_t) \\ \wedge\; Can_Q(X'_t, X'_\tau) \;\wedge\; Eq_\Sigma(X_t, X'_t, X_e) \wedge Eq_{\mathbb{B}^n}(X_\tau, X'_\tau, X_e) \;\wedge\; Bef_\bullet(X_\tau, X_e) \\ \wedge\; \neg Can_Q(X''_t, X''_\tau) \;\wedge\; Eq_\Sigma(X_t, X''_t, X_e) \;\wedge\; Eq_{\mathbb{B}^n}(X_\tau, X''_\tau, X_e) \;\wedge\; C_{\mathbb{B}^n}(X''_\tau)$$

Here, $Bef_\bullet$ is like *Bef* but for partial tuples, and $C_{\mathbb{B}^n} \subseteq T_{\mathbb{B}^n}$ is the set of trees of type 1^n. Let $2Alive^S_Q$ be the binary version of $Alive^S_Q$, then $val(2Alive^S_Q) = |concur^S_Q|$. Automata for $C_{\mathbb{B}^n}$ and $Eq_{\mathbb{B}^n}$ are necessarily of size $O(2^n)$, which cannot be avoided by embedding them inside other relation like $\neg Can_Q$. But for a fixed n, all these automata can be computed in P-time, so that Corollary 1 applied to $2Alive^S_Q$ proves that bounded concurrency can be decided in P-time.

Conclusion. In this paper we proved that bounded delay and (for fixed n) bounded concurrency are both computable in P-time, for queries defined by dSTAs. This was obtained by studying some properties of recognizable relations on unranked trees, and combining them with prior results on the valuedness of tree transducers [17]. Considering the P-time translation of a fragment of XPath to dSTAs proposed in [10], we get the same complexity results for this fragment of XPath.

Some questions are left open in the present paper. For fixed k and n, deciding k-boundedness for delay and concurrency for n-ary queries defined by dSTAs is known to be in NP-time. However, NP-hardness is still open. We also chose to define the delay for selection from the time point where the tuple gets complete. An alternative could be to define the ith delay, that starts when i components of the tuple are filled.

Acknowledgments. This work was partially supported by the Enumeration project ANR-07-blanc.

References

1. Segoufin, L., Vianu, V.: Validating streaming XML documents. In: ACM PODS, pp. 53–64 (2002)
2. Martens, W., Neven, F., Schwentick, T., Bex, G.J.: Expressiveness and complexity of XML schema. ACM Transactions of Database Systems 31(3), 770–813 (2006)

3. Gupta, A.K., Suciu, D.: Stream processing of XPath queries with predicates. In: ACM SIGMOD, pp. 419–430 (2003)
4. Fernandez, M., Michiels, P., Siméon, J., Stark, M.: XQuery streaming á la carte. In: 23nd International Conference on Data Engineering, pp. 256–265 (2007)
5. Benedikt, M., Jeffrey, A.: Efficient and expressive tree filters. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 461–472. Springer, Heidelberg (2007)
6. Bar-Yossef, Z., Fontoura, M., Josifovski, V.: Buffering in query evaluation over XML streams. In: ACM PODS, pp. 216–227 (2005)
7. Olteanu, D.: SPEX: Streamed and progressive evaluation of XPath. IEEE Trans. on Know. Data Eng. 19(7), 934–949 (2007)
8. Gou, G., Chirkova, R.: Efficient algorithms for evaluating XPath over streams. In: ACM SIGMOD, pp. 269–280 (2007)
9. Berlea, A.: Online evaluation of regular tree queries. Nordic Journal of Computing 13(4), 1–26 (2006)
10. Gauwin, O., Niehren, J., Tison, S.: Earliest query answering for deterministic streaming tree automata and a fragment of XPath (2009)
11. Gauwin, O., Niehren, J., Roos, Y.: Streaming tree automata. Information Processing Letters 109(1), 13–17 (2008)
12. Alur, R.: Marrying words and trees. In: ACM PODS, pp. 233–242 (2007)
13. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: 36th ACM Symposium on Theory of Computing, pp. 202–211 (2004)
14. Neumann, A., Seidl, H.: Locating matches of tree patterns in forests. In: Arvind, V., Ramanujam, R. (eds.) FST TCS 1998. LNCS, vol. 1530, pp. 134–146. Springer, Heidelberg (1998)
15. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: (1997) (revised October 12th, 2007), http://tata.gforge.inria.fr
16. Carme, J., Niehren, J., Tommasi, M.: Querying unranked trees with stepwise tree automata. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 105–118. Springer, Heidelberg (2004)
17. Seidl, H.: Single-valuedness of tree transducers is decidable in polynomial time. Theoretical Computer Science 106, 135–181 (1992)
18. Stearns, R.E., Hunt III, H.B.: On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. SIAM Journal on Computing 14(3), 598–611 (1985)
19. Weber, A., Seidl, H.: On the degree of ambiguity of finite automata. In: Wiedermann, J., Gruska, J., Rovan, B. (eds.) MFCS 1986. LNCS, vol. 233, pp. 620–629. Springer, Heidelberg (1986)
20. Allauzen, C., Mohri, M., Rastogi, A.: General algorithms for testing the ambiguity of finite automata. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 108–120. Springer, Heidelberg (2008)
21. Carme, J., Lemay, A., Niehren, J.: Learning node selecting tree transducer from completely annotated examples. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 91–102. Springer, Heidelberg (2004)
22. Seidl, H.: Ambiguity, valuedness and costs, Habilitation Thesis (1992)
23. Sakarovitch, J., de Souza, R.: Decidability of bounded valuedness for transducers. In: Mathematical Foundations of Computer Science (2008)

Learning by Erasing in Dynamic Epistemic Logic

Nina Gierasimczuk*

Institute for Logic, Language, and Computation, University of Amsterdam
Institute of Philosophy, University of Warsaw
n.gierasimczuk@uva.nl

Abstract. This work provides a comparison of learning by erasing [1] and iterated epistemic update [2] as analyzed in dynamic epistemic logic (see e.g. [3]). We show that finite identification can be modelled in dynamic epistemic logic and that the elimination process of learning by erasing can be seen as iterated belief-revision modelled in dynamic doxastic logic.

1 Introduction

There have been many formal attempts to grasp the phenomenon of epistemic change. In this paper we will discuss two of them. On the one hand we have the formal learning theory (LT) framework (see e.g. [4]), with its direct implications for analysis of scientific discovery, on the other — belief-revision theory in its interrelation with dynamic epistemic logic (DEL). In learning theory, the classical framework of identification in the limit [5] was motivated mostly by the problem of language acquisition. It turned out to be very useful for modelling the process of grammar inference, and found numerous applications in the area of syntax. Initially the idea of identification was unappreciated in semantic considerations, but eventually also this direction has started to be developed resulting in applications to the acquisition of semantics of natural language [6,7,8] as well as in modelling the process of scientific inquiry [9]. The serious step towards involving more semantics was coupled with the design of model-theoretic learning [10] and its application to belief-revision theory [11].

Other, very prominent directions that explicitly involve notions of knowledge and belief have been developed in the area of epistemology. First, a precise language to discuss epistemic states of agents has been established in [12]. After that the need of formalizing dynamics of knowledge emerged. The belief-revision AGM framework [13] constitutes an attempt to talk about the dynamics of epistemic states. Belief-revision policies thus explained have been successfully modelled in dynamic epistemic logic (see [3]) and in the above-mentioned model-theoretic learning [11].

In the present paper we show how those two important traditions, LT and DEL, can be merged. We explain this connection by joining iterated epistemic

* The author is a receiver of the Foundation for Polish Science Award for Young Researchers (START Programme 2008).

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 362–373, 2009.

update as modelled in DEL with a special case of learning in the limit — learning by erasing [14,1].

We will proceed according to the following plan. First we explain the ideas of dynamic epistemic logic (DEL) from a strictly semantic point of view. We will also mention an important modification of DEL, namely dynamic doxastic logic (DDL). As we will see the part 'dynamic' in those names refers to the fact that those logics include operators which modify models. With respect to those modifications we discuss the notions of epistemic and doxastic update. In particular, we focus on public announcement as a special case. Then we leave this logical subject and move to briefly recall the basics of formal learning theory in its set-theoretical version. After that the definition of learning by erasing is provided. The last two parts present a way to model finite identification in DEL and learning by erasing in DDL.

2 Dynamic Epistemic Logic – Semantic Perspective

In general, dynamic epistemic logic has been introduced to formalize knowledge change. In this section basic notions of DEL will be provided. The definitions are based on [15]. Let us take *Atom* to be a set of atomic propositions and A — a set of agents.

Definition 1 (Epistemic Model). *Epistemic model M is a triple $\langle W, \{\sim_i\}_{i\in A}, V\rangle$, where W is a set of possible worlds, for each $i \in A$, $\sim_i \subseteq W \times W$ is an indistinguishability relation and $V : Atom \to \wp(W)$ is a valuation.*

Intuitively speaking, M formalizes the epistemic situation of all agents from A. The indistinguishability relation models their uncertainty about which of the possible worlds is the actual one.

Definition 2 (Event Model). *An event model E is a triple $\langle S, \{\rightarrow_i\}_{i\in A}, pre\rangle$, where S is a set of worlds, for each $i \in A$, $\rightarrow_i \subseteq S \times S$, and $pre : S \to Atom$ is a pre-condition function which indicates what* pre-condition *a world has to satisfy to enable the event to take place.*

Event model describes the epistemic content of the event. Relation $\rightarrow_i$ directly corresponds to the indistinguishability relation $\sim_i$ of epistemic model.

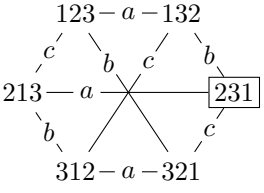
Definition 3 (Product Update). *Let $M = \langle W, \{\sim_i\}_{i\in A}, V\rangle$ and $E = \langle S, \{\rightarrow_i\}_{i\in A}, pre\rangle$. The product update $M \otimes E$ is the epistemic model $M' = \langle W', \{\sim'_i\}_{i\in A}, V'\rangle$ such that:*

- $W' = \{(w, s) | w \in W, s \in S$ *and* $M, w \models pre(s)\}$,
- $(w, s) \sim_i (w', s')$ *iff* $w \sim_i w'$ *and* $s \rightarrow_i s'$,
- $V'((w, s)) = V(w)$.

Definition 4 (Public Announcement [16]). *The public announcement of a formula φ is the event model $E_\varphi = \langle S, \{\rightarrow_i\}_{i\in A}, pre\rangle$, such that $S = \{e\}$ and for each $i \in A$, $e \rightarrow_i e$ and $pre(e) = \varphi$.*

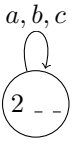
The major result of updating an epistemic model M with public announcement of φ is a submodel of M containing only the states that satisfy φ.

Example 1. Let us take the set of agents $A = \{a \text{ (Anne)}, b \text{ (Bob)}, c \text{ (Carl)}\}$ and the deck of cards consisting of: 1, 2, 3. Each person gets one card. We can represent the situation after dealing as a triple xyz, where x, y, z are cards and the first position in the triple assigns the value to a (Anne), second to b (Bob), etc. For instance, 231 means that Anne has 2, Bob has 3 and Carl has 1. All possible situations after a deal are: 123, 132, 213, 231, 312, 321. We assume that all the players are witnessing the fact of dealing but they do not know the distribution of the cards. The epistemic model M of this situation is illustrated in the figure.

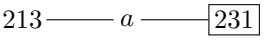


Let us then assume that as a result the actual world is 231. Obviously each player's knowledge does not allow certainty about which is the actual world. In the model the uncertainty of the agent x about the worlds w and w' is symbolized by the following: $w \sim_x w'$ (in the Figure this relation is depicted by two states being joined by a line labeled with x).

Let us now assume that Anne shows her card to all the players publicly, i.e., all the players see her card and all of them know that all of them see it. This event is modelled by $E = (S, \{\rightarrow_i\}_{i \in A}, pre)$, where $S = \{s\}$, for each $x \in A$, $s \rightarrow_x s$ and $pre(s) = 2\_\_$ ('Anne has 2').



The public announcement of 'Anne has 2' results in the epistemic situation, which can be presented as $M' = M \otimes E$ (depicted below).



Event E is an example of a public announcement, in this case: 'Anne has 2'. In dynamic epistemic logic the public announcement of φ is represented by '$!\varphi$' and corresponds to the elimination of all those possible worlds that do not satisfy φ. In other words, public announcement works as relativization of the model to those worlds that satisfy the content of the announcement.

3 Dynamic Doxastic Logic

The objective of dynamic doxastic logic (DDL) is to formalize the notion of belief change. This is usually done by introducing preference relations over the possible worlds. Each agent has his own preference relation. Belief of agent a is determined by the set of his most preferred states.

Definition 5 (Epistemic Plausibility Model). *Let Atom be a set of atomic propositions and A — a set of agents. Epistemic plausibility model E is a quadruple:* $\langle W, \{\sim_i\}_{i\in A}, \{\leq_i\}_{i\in A}, V\rangle$, *where W is a set of possible worlds, for each* $i \in A$, $\sim_i \subseteq W \times W$ *is an indistinguishability relation,* $\leq_i \subseteq W \times W$ *is a preference relation and* $V : Atom \to \wp(W)$ *is a valuation.*

Definition 6 (Plausibility Event Model). *An event model E is a quadruple:* $\langle S, \{\rightarrow_i\}_{i\in A}, \{\preceq_i\}_{i\in A}, pre\rangle$, *where S is a set of worlds, for each* $i \in A$, $\rightarrow_i \subseteq S \times S$, $\preceq_i \subseteq S \times S$ *and* $pre : S \to Atom$ *is a pre-condition function.*

For completeness' sake we add the definition of priority update.

Definition 7 (Priority Update). *The priority update works analogously to the epistemic update. The additional condition is for the* $\leq_i$ *relation:*

- *for* $w \in W$ *and* $s \in S$, $(w, s) \leq'_i (w', s')$ *iff* $s \prec_i s'$, *or* $s \simeq_i s'$ *and* $w \leq_i w'$, *where* $s \simeq_i s'$ *iff* $s \preceq_i s'$ *and* $s' \preceq_i s$.

4 Learning Theory

4.1 Identification in the Limit

Learning theory is concerned with the process of inductive inference [5]. We can think of it as of a game between Scientist and Nature. In the beginning we have a class of possible worlds together with a class of hypotheses (possible descriptions of worlds). Different hypotheses may describe the same world. We assume that both Scientist and Nature know what all the possibilities are, i.e., they both have access to the initial classes. Nature chooses one of those possible worlds to be the actual one. Scientist has to guess which it is. Scientist receives information about the world in an inductive manner. The stream of data is infinite and contains only and all the elements from the chosen reality. Each time Scientist receives a piece of information he answers with one of the hypotheses from the initial class. We say that Scientist identifies Nature's choice in the limit if after some finite number of guesses his answers stabilize on a correct hypothesis. Moreover, to discuss more general identifiability, we require that the same is true for all the possible worlds from the initial class, i.e., regardless of which element from the class is chosen by Nature to be true, Scientist can identify it in the limit on the basis of data about the actual world.

To formalize this simple setting we need to make the notion of *stream of data* clear. In learning theory such streams are often called 'environments'[1].

[1] We are concerned here only with sequences of positive information (texts).

Let us consider E — the set of all computably enumerable sets. Let $C \subseteq E$ be some class of c.e. sets. For each S in C we consider Turing machines h_n which generate S and in such a case we say that n is an index of S. The Turing machines will function as the conjectures that Scientist makes. It is well-known that each S has infinitely many indices. Let us take I_S to be the set of all indices of the set S, i.e, $I_S = \{n | h_n \text{ generates } S\}$.

Definition 8 (Environment). *By environment of S, ε, we mean any infinite sequence of elements from S such that:*

1. *ε enumerates all the elements from S;*
2. *ε enumerates only the elements from S;*
3. *ε allows repetitions.*

Definition 9 (Notation). *We will use the following notation:*

- *ε_n is the n-th element of ε;*
- *$\varepsilon|n$ is a sequence $(\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{n-1})$;*
- *SEQ denotes the set of all finite initial segments of all environments;*
- *$set(\varepsilon)$ is a set of elements that occur in ε;*
- *h_n will refer to a hypothesis, i.e., a finite description of a set, a Turing machine generating S;*
- *L is a learning function — a map from finite data sequences to indices of hypotheses, $L : SEQ \to I_{H_C}$.*

The structure of the identifiability in the limit can be formulated by the following chain of definitions:

Definition 10 (Identification in the limit, LIM). *We say that a learning function L:*

1. *identifies $S \in C$ in the limit on ε iff there is a number k, such that for co-finitely many m, $L(\varepsilon|m) = k$ and $k \in I_S$;*
2. *identifies $S \in C$ in the limit iff it identifies S in the limit on every ε for S;*
3. *identifies C in the limit iff it identifies in the limit every $S \in C$.*

The notion of identifiability can be strengthened in various ways. One radical case is to introduce a finiteness condition for identification.

Definition 11 (Finite identification, FIN). *We say that a learning function L:*

1. *finitely identifies $S \in C$ on ε iff, when successively fed ε, at some point L outputs a single k, such that $k \in I_S$, and stops;*
2. *finitely identifies $S \in C$ iff it finitely identifies S on every ε for S;*
3. *finitely identifies C iff it finitely identifies every $S \in C$.*

4.2 Learning by Erasing

Learning by erasing [1,17] is an epistemologically intuitive modification of identification in the limit. Very often the cognitive process of converging to a correct conclusion consists of eliminating those possibilities that are falsified during the inductive inquiry. Accordingly, in the formal model the outputs of the learning function are negative, i.e., the function each time eliminates a hypothesis, instead of explicitly guessing one that is supposed to be correct. A special case of learning by erasing is co-learning [14]. The set $S \in C$ is co-learnable iff there is a function which stabilizes by eliminating all indices from I_{H_C} except just one from I_S. The difference between this approach and the usual identification is in the interpretation of the positive guess of the learning function. In learning by erasing there is always some ordering of the initial hypothesis space. This allows to interpret the actual positive guess of the learning-by-erasing function to be the least hypothesis (in a given ordering) not yet eliminated.

Let us give now the two definitions that explain the notion of learning by erasing.

Definition 12 (Function Stabilization). *In learning by erasing we say that a function stabilizes to number k on environment ε if and only if for co-finitely many $n \in \mathbb{N}$:*

$$k = min\{\mathbb{N} - \{L(\varepsilon|0), \ldots, L(\varepsilon|n)\}\}.$$

Definition 13 (Learning by Erasing, E-learning). *We say that a learning function, L:*

1. *learns $S \in C$ by erasing on ε iff L stabilizes to k on ε and $k \in I_S$;*
2. *learns $S \in C$ by erasing iff it learns by erasing S from every ε for S;*
3. *learns C by erasing iff it learns by erasing every $S \in C$.*

A variety of additional conditions for learning can be defined. Let us mention the following conditions on e-learning function L [1].

1. L erases all but one, correct hypothesis (co-learning, e-ALL);
2. L erases only hypotheses that are incorrect (e-SUB);
3. L erases exactly all hypotheses that are incorrect (e-EQ);
4. L erases all hypotheses that are incorrect but may also erase some that are correct (e-SUPER);

Let us cite two theorems [1] that establish the relationships between various types of learning: e-learning, finite identifiability and identifiability in the limit.

Theorem 1. *FIN* $\subset$ *e-EQ* $\subset$ *e-SUB* $\subset$ *LIM*

Theorem 2. e-ALL, e-SUPER = *LIM*

5 Finite Identification in DEL

The word 'learning' is used in epistemology to cover a variety of epistemic processes. One of them is the epistemic update in the form of one-step *learning that* φ, followed by a direct modification of the set of beliefs, as we have seen in sections 2 and 3. In the learning-theoretic setting the incoming information is of a different nature than the actual thing being learned. This feature has an important consequence for modelling learning in DEL. We are forced to provide two-sorted models, with one sort for pieces of incoming information and another for the hypotheses. To establish a bridge between those two different ontologies we treat a hypothesis as the set of events that it predicts, e.g., if we take a hypothesis h to be 'There are all natural numbers except 3' it predicts that the environment will enumerate all the natural numbers except 3.

The possible worlds in our epistemic model are identified with hypotheses. Unlike in the classical DEL approach, the event models are announcements of data corresponding to elements of the sets being learned, and not hypotheses themselves.

A further difference is in the number of agents. In sections 2 and 3 we provided definitions for multi-agent epistemic cases. Although science as well as learning seem to be at least a two-player game, in the present paper we are concerned only with the role of Scientist (Learner). By implication, we assume Nature (Teacher) to be an objective machine that makes an arbitrary choice and gives out random data, she does not have any particular strategy, is neither helping the learner, nor obstructing his attempts to identify a correct hypothesis. We recognize the possibility and potential of analyzing two or more agents in the contexts of inductive inference. However, for the sake of simplicity our DEL and DDL models are going to account only for one agent.

Let us again fix C to be a class of sets, and for each $S_n \in C$ we consider h_n to be a hypothesis that describes S_n. In learning by erasing we can take the initial epistemic model to represent the background knowledge of Scientist together with his uncertainty about which world is the actual one. Let us take the initial epistemic frame to be

$$M = \langle H_C, \sim \rangle ,$$

where H_C is a possibly infinite[2] set of worlds (hypotheses that are considered possible) and $\sim \subseteq H_C \times H_C$ is an uncertainty relation for Scientist. Since we assume that the initial hypothesis space is arbitrary, we also do not require any particular preference of the scientist over H_C. Hence, we take the relation $\sim$ to be a universal, equivalence binary relation over H_C. The initial epistemic state of the Scientist is depicted in Figure 1. This model corresponds to the starting point of the scientific discovery process. Each world represents a hypothesis from the initial set determined by the background knowledge. In the beginning Scientist considers all of them possible. The model also reflects the fact that Scientist is given the class of hypotheses H_C. In other words he knows what the alternatives are.

[2] We can effectively deal with the epistemic update and identification in infinite domains by using special enumeration strategies (for explanation and examples see [18]).

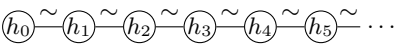


Fig. 1. Initial epistemic model

Next, Nature decides on some state of the world by choosing one possibility from C. Let us assume that as a result h_3 correctly describes the chosen world. Then, she decides on some particular environment ε, of the elements from the world. We picture this enumeration in Figure 2 below.

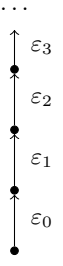


Fig. 2. Environment ε consistent with h_3

The sequence ε is successively given to Scientist. Let us focus now on the first step of the procedure. We have the uncertainty range of Scientist, it runs through the whole set of hypotheses H_C. A piece of data ε_0 is given to Scientist. This fact can be represented by the event model $E_0 = \langle \{e\}, \rightarrow, pre \rangle$, where $e \rightarrow e$ and $pre(e) = \varepsilon_0$ (see Figure 3).

Fig. 3. Event model E_0 of the announcement of ε_0

Scientist, when confronted with the announcement of ε_0 updates his epistemic state accordingly. We will represent the process formally by the product update $M \otimes E_0$. The result of the product update is again an epistemic model $M' = \langle {H_C}', \sim' \rangle$, where:

1. ${H_C}' = \{(h_n, e) | h_n \in H_C \ \& \ pre(e) \in S_n)\}$;
2. $\sim' = \sim | {H_C}'$.

We use here event models similar in spirit to those of public announcement [16]. They consist in only one state with a pre-condition determined by the piece of data that is given. In Figure 4 Scientist's confrontation with ε_0 is depicted.

Scientist tests each hypothesis with ε_0. If a hypothesis is consistent with it, it remains as a possibility, if it is not consistent, it is eliminated (see figure 5). Let us assume that ε_0 is not consistent with h_2.

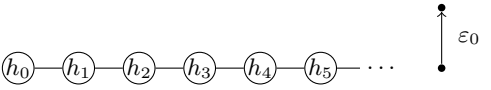


Fig. 4. Confrontation with data

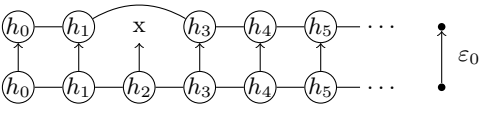


Fig. 5. Epistemic update

This epistemic update can be iterated infinitely many times along ε resulting in an infinite sequence of models which according to the lines of DEL can be called ε-generated epistemic model (see e.g. [15]).

Definition 14 (Generated Epistemic Model). *The generated epistemic model $(M)^{\varepsilon}$, with $\varepsilon = \varepsilon_0, \varepsilon_1, \varepsilon_2, \ldots$, is the result of epistemic update $M \otimes E_0 \otimes E_1 \otimes E_2 \otimes \ldots$, where for each n, the event E_n corresponds to the announcement of ε_n.*

Let us now see a simple example of finite identification of a single hypothesis.

Example 2. Let us take $H_C = \{h_0, h_1, h_2\}$, such that $h_n = \{0, ..., n\}$. Nature makes her choice regarding what the world is like. We assume that as a result h_2 holds. Then, Nature chooses an enumeration $\varepsilon = 0, 1, 0, 2, 1, \ldots$. After the first piece of data, 0, the uncertainty range of Scientist includes the whole H_C. After the second, 1, Scientist eliminates h_0 since it does not contain the event 1 and now he hesitates between h_1 and h_2. The third piece, 0, does not change anything, however the next one, 2, eliminates h_1. Uncertainty is eliminated as well. He knows that the only hypothesis that can be true is h_2. Therefore, we can say that he learned it conclusively, with certainty.

The above discussion suggests the following thesis.

Thesis 1. *Finite identifiability can be modelled within the DEL framework, using:*

- *epistemic states for hypotheses;*
- *infinite sequences of announcements for environments;*
- *epistemic update for the progress in eliminating uncertainty over hypothesis space.*

Scientist succeeds in finite identification of S from ε if and only if there is a finite initial segment of ε, $\varepsilon|n$, such that the domain of the $\varepsilon|n$-generated model contains only one hypothesis h_k and $k \in I_S$. In other words, there is a finite step of the iterated epistemic update along ε, that eliminates Scientist's uncertainty.

6 Learning by Erasing in DDL

From Scientist's point of view the process of learning has a few components that are very important in logical modelling. The first is of course the current conjecture — a hypothesis that is considered appropriate in a given step of the procedure. The second is the set of those hypotheses that were used in the past and have already been discarded. The third part is the set of hypotheses that are still considered to be possible, but for some reasons less probable than the chosen one.

Let us consider the following example of a learning scenario, in which the uncertainty is never eliminated.

Example 3. As you probably observed, in the Example 2 Scientist was very lucky. Let us assume for a moment that nature had chosen h_1, and had fixed the enumeration $\varepsilon = 0, 1, 0, 1, 1, 1, 1, \ldots$ In this case Scientist's uncertainty can never be eliminated.[3]

This example indicates that the central element of the identification in the limit model is the unavoidable presence of uncertainty. The limiting framework allows however the introduction of some kind of *operational* knowledge, which is uncertainty-proof.

To model the algorithmic nature of the learning process that includes actual guess and other not-yet-eliminated possibilities, we enrich the epistemic model with some preference relation $\leq$: $H_C \times H_C$. The relation $\leq$ represents some preference over the set of hypotheses, e.g., if Scientist is an occamist, the preference would be defined according to the simplicity of hypotheses. In the initial epistemic state the uncertainty of the scientist again ranges over all of H_C. This time however the class is ordered and Scientist current belief is the most preferred hypothesis. Therefore, we consider the initial epistemic state of Scientist to be:

$$M = \langle H_C, \sim, \leq \rangle .$$

The procedure of erasing hypotheses that are inconsistent with successively incoming data is the same as in the previous section. This time however let us introduce the current-guess state which is interpreted as the actual guess of the Scientist. It is always the one that is most preferred — the smallest one according to $\leq$. In doxastic logic a set of most preferred hypotheses is almost invariably interpreted as the one that the agent *believes* in. Let us go back to Example 2, where Nature chose a world consistent with h_1. After seeing 1 and eliminating h_0, Scientist's attention focuses on h_1, then h_1 is his current belief. It is the most preferred hypothesis, and as such it can be reiterated as long as it is consistent with ε. In this particular case, since Nature chose a world consistent with h_1,

[3] As we are interested here in learning by erasing, we assume a suitable underlying ordering of hypothesis space. In this case it is: h_0, h_1, h_2. However, note that this type of identification is not order-independent. If the initial ordering was: h_0, h_2, h_1, then Scientist would not stabilize on the correct hypothesis.

it will never be contradicted, so Scientist will always be uncertain between h_1 and h_2. However, his preference directs him to believe in the correct hypothesis, without him being aware of the correctness. Therefore, we claim the following.

Thesis 2. *Learning by erasing can be modelled within the DDL framework, using:*

- *epistemic states for hypotheses;*
- *infinite sequences of announcements for environments;*
- *epistemic update for the progress in eliminating uncertainty over the hypothesis space;*
- *preference relation for the underlying hypothesis space;*
- *in each step of the procedure, the most preferred hypothesis for the actual positive guess of the learning function.*

Scientist learns S by erasing from ε if and only if there is n such that for every $m > n$, the most preferred state of the domain of the $\varepsilon|m$-generated epistemic model is h_k, and $k \in I_S$.

7 Conclusions and Further Work

In this paper we argued that the process of inductive inference can be modelled in dynamic epistemic logic and dynamic doxastic logic. To support our claim we provided a translation of the components of learning into a two-sorted semantics for DEL and DDL. In particular, we see DEL as an appropriate framework to analyze the notion of finite identifiability. Learning by erasing, a special case of identifiability in the limit, is based on the existence of an underlying ordering of hypothesis space. Therefore, in logical modelling it requires adding to the epistemic model a preference relation over possible worlds. This indicates that it should be formalized in DDL, where the preference relation is a standard element of any model.

The above-presented conceptual work has many implications and possible continuations. After establishing a correspondence on the semantic level, it is possible to formulate axioms of epistemic logic for inductive inference. We find this project promising and potentially fruitful for both DEL and LT. Moreover, modal analysis of the process of learning can be continued in the following directions:

- formulating LT theorems as validities in epistemic and temporal logic;
- analyzing the inductive inference process in game-theoretical terms, and discussing strategies for learning and teaching;
- studying the notion of non-introspective operational knowledge and uncertainty that are involved in the process of inductive inference;
- comparing formal learning theory and belief-revision theory in a systematic way.

References

1. Lange, S., Wiehagen, R., Zeugmann, T.: Learning by erasing. In: Arikawa, S., Sharma, A.K. (eds.) ALT 1996. LNCS (LNAI), vol. 1160, pp. 228–241. Springer, Heidelberg (1996)
2. Baltag, A., Moss, L.: Logics for epistemic programs. Synthese 139(2), 165–224 (2004)
3. van Ditmarsch, H., van der Hoek, W., Kooi, B.: Dynamic Epistemic Logic. Springer, Heidelberg (2007)
4. Jain, S., Osherson, D., Royer, J.S., Sharma, A.: Systems that Learn. MIT Press, Cambridge (1999)
5. Gold, E.: Language identification in the limit. Information and Control 10, 447–474 (1967)
6. Tiede, H.J.: Identifiability in the limit of context-free generalized quantifiers. Journal of Language and Computation 1, 93–102 (1999)
7. Costa Florêntio, C.: Learning generalized quantifiers. In: Proc. 7th ESSLLI Student Session (2002)
8. Gierasimczuk, N.: The problem of learning the semantics of quantifiers. In: ten Cate, B.D., Zeevat, H.W. (eds.) TbiLLC 2005. LNCS (LNAI), vol. 4363, pp. 117–126. Springer, Heidelberg (2007)
9. Kelly, K.: The Logic of Reliable Inquiry. Oxford University Press, Oxford (1996)
10. Osherson, D., de Jongh, D., Martin, E., Weinstein, S.: Formal learning theory. In: van Benthem, J., Ter Meulen, A. (eds.) Handbook of Logic and Language, pp. 737–775. MIT Press, Cambridge (1997)
11. Martin, E., Osherson, D.: Elements of Scientific Inquiry. MIT Press, Cambridge (1998)
12. Hintikka, J.: Knowledge and Belief. An Introduction to the Logic of the Two Notions. Cornell University Press (1962)
13. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the logic of theory change: Partial meet contraction and revision functions. The Journal of Symbolic Logic 50(2), 510–530 (1985)
14. Freivalds, R., Zeugmann, T.: Co–learning of recursive languages from positive data. In: Bjorner, D., Broy, M., Pottosin, I.V. (eds.) PSI 1996. LNCS, vol. 1181, pp. 122–133. Springer, Heidelberg (1996)
15. van Benthem, J., Gerbrandy, J., Pacuit, E.: Merging frameworks for interaction: DEL and ETL. In: Proc. TARK 2007, pp. 72–81 (2007)
16. Batlag, A., Moss, L.S., Solecki, S.: The logic of public announcements and common knowledge and private suspicions. In: Proc. TARK 1998, pp. 43–56 (1998)
17. Freivalds, R., Karpinski, M., Smith, C., Wiehagen, R.: Learning by the process of elimination. Information and Computation 176(1), 37–50 (2002)
18. Gierasimczuk, N.: Identification through inductive verification. Application to monotone quantifiers. In: Bosch, P., Gabelaia, D., Lang, J. (eds.) TbiLLC 2007. LNCS, vol. 5422, pp. 193–205. Springer, Heidelberg (2009)

The Fault Tolerance of NP-Hard Problems

Christian Glaßer[1], A. Pavan[2,⋆], and Stephen Travers[1]

[1] Julius-Maximilians-Universität Würzburg, Germany
{glasser,travers}@informatik.uni-wuerzburg.de
[2] Iowa State University, USA
pavan@cs.iastate.edu

Abstract. We study the effects of faulty data on NP-hard sets. We consider hard sets for several polynomial time reductions, add corrupt data and then analyze whether the resulting sets are still hard for NP. We explain that our results are related to a weakened deterministic variant of the notion of program self-correction by Blum, Luby, and Rubinfeld. Among other results, we use the Left-Set technique to prove that m-complete sets for NP are nonadaptively weakly deterministically self-correctable while btt-complete sets for NP are weakly deterministically self-correctable. Our results can also be applied to the study of Yesha's p-closeness. In particular, we strengthen a result by Ogiwara and Fu.

1 Introduction

Even small amounts of faulty data can obscure reasonable information. For instance, by filling more and more whitespaces of a printed text with arbitrary letters, it can become quite difficult to understand the original meaning of the text.

The same holds true for NP-complete sets. Take for instance SAT, the set of all satisfiable formulas. By adding *false positives* to SAT, i.e., some unsatisfiable formulas, we can actually lose information: If we overdo it, we end up with $\mathrm{SAT} \cup \overline{\mathrm{SAT}} = \Sigma^*$, and by this definitely lose NP-completeness. But how much false positive data can NP-hard sets handle, i.e., how many false positives can we add such that the resulting set stays NP-hard? Alternatively, how much effort is needed to extract the original information?

In this paper, we investigate how polynomial time reductions can cope with false positives. More precisely, we consider NP-hard sets for several polynomial time reductions and add false positives to the sets.

Moreover, we study the effects of more general kinds of faulty data. We investigate how polynomial time reductions can handle combinations of both, false positives and *false negatives*. This relates our research to the notion of *program self-correction* which was introduced by Blum, Luby, and Rubinfeld [1]. That notion addresses a fundamental question regarding software reliability: Can one increase the reliability of existing software without understanding the way it

⋆ Research supported in part by NSF grant CCF-0430807.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 374–385, 2009.

works? More precisely, let P be a program that is designed to solve a problem L. However, we do not know whether P is correct. Is it possible to write an auxiliary program M that uses P such that if P errs only on a small fraction of the inputs, then with high probability M corrects the errors made by P? So M has to find the right answer with high probability by calling P on several inputs.

Our investigations of the consequences of faulty data are related to a considerably weakened *deterministic* variant of self-correction. In this case, the error probability of the polynomial-time wrapping machine M must be 0, i.e., M must achieve certainty about the question of whether the input belongs to L. However, we only require M to correct very few errors (i.e., $p(n)$ errors for some polynomial p). For probabilistic self-correction however, a probabilistic polynomial-time corrector must be able to correct up to $2^n/p(n)$ errors for some polynomial p. We prove that

- The symmetric difference of m-hard sets and sparse sets is always tt-hard. This implies that m-complete sets for NP are *nonadaptively weakly deterministically self-correctable.*
- The symmetric difference of btt-hard sets and arbitrary sparse sets is always T-hard. This implies that btt-complete sets are *weakly deterministically self-correctable.*
- The union of dtt-hard sets and arbitrary sparse sets is always T-hard.

These results show that $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard, $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard, and $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard sets do not become too easy when false positives are added (as they stay NP-hard with respect to more general reducibilities). On the other hand, we show that unless P = NP, there exist sparse sets S_1, S_2 such that $\mathrm{SAT} \cup S_1$ is not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for NP, and $\mathrm{SAT} \cup S_2$ is not $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for NP.

Furthermore, we explain that one of our results about btt-reducibility is related to the notion of *p-closeness* which was introduced by Yesha [2]. We show that no $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard set for NP is p-close to P, unless P = NP. This strengthens a result by Ogiwara [3] and Fu [4] who proved that no $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard set for NP is p-close to P, unless P = NP.

2 Preliminaries

We recall basic notions. Σ denotes a finite alphabet with at least two letters, Σ^* denotes the set of all words, and $|w|$ denotes the length of a word w. For $n \geq 0$, Σ^n denotes the set of all words of length n. A set $A \subseteq \Sigma^*$ is *nontrivial* if $A \neq \emptyset$ and $A \neq \Sigma^*$. A *tally* set is a subset of 0^*. The census function of a set S is defined as $\mathrm{census}_S(n) \stackrel{\mathrm{df}}{=} |S \cap \Sigma^n|$. A set S is *sparse* if there exists a polynomial p such that for all $n \geq 0$, $\mathrm{census}_S(n) \leq p(n)$. The symmetric difference of sets A and B is defined as $A \triangle B = (A - B) \cup (B - A)$.

The language accepted by a machine M is denoted by $L(M)$. The characteristic function of a set A is denoted by c_A. $\overline{L}$ denotes the complement of a language L and co$\mathcal{C}$ denotes the class of complements of languages in $\mathcal{C}$. FP denotes the class of functions computable in deterministic polynomial time.

We recall standard polynomial-time reducibilities [5]. A set B *many-one-reduces* to a set C (*m-reduces* for short; in notation $B \leq^{\mathrm{p}}_{\mathrm{m}} C$) if there exists a total, polynomial-time-computable function f such that for all strings x, $x \in B \Leftrightarrow f(x) \in C$.

A set B *Turing-reduces* to a set C (*T-reduces* for short; in notation $B \leq^{\mathrm{p}}_{\mathrm{T}} C$) if there exists a deterministic polynomial-time-bounded oracle Turing machine M such that for all strings x, $x \in B \Leftrightarrow M$ with C as oracle accepts the input x.

A set B *truth-table-reduces* to a set C (*tt-reduces* for short; in notation $B \leq^{\mathrm{p}}_{\mathrm{tt}} C$) if there exists a deterministic polynomial-time-bounded oracle Turing machine M that queries nonadaptively such that for all strings x,
$x \in B \Leftrightarrow M$ with C as oracle accepts the input x.

A set B *disjunctively truth-table-reduces to a set* C (dtt-reduces for short; in notation $B \leq^{\mathrm{p}}_{\mathrm{dtt}} C$) if there exists a total, polynomial-time-computable function $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$ such that for all strings x, $x \in B \Leftrightarrow f(x) \cap C \neq \emptyset$.

A set B *conjunctively truth-table-reduces to a set* C (ctt-reduces for short; in notation $B \leq^{\mathrm{p}}_{\mathrm{ctt}} C$) if there exists a total, polynomial-time-computable function $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$ such that for all strings x, $x \in B \Leftrightarrow f(x) \subseteq C$.

A set B *bounded truth-table-reduces to a set* C (btt-reduces for short; in notation $B \leq^{\mathrm{p}}_{\mathrm{btt}} C$) if there exists a $k \geq 1$, a k-ary Boolean function α, and $g_1, \ldots, g_k \in \mathrm{FP}$ such that for all x $x \in B \Leftrightarrow \alpha(c_C(g_1(x)), c_C(g_2(x)), \ldots, c_C(g_k(x))) = 1$.

A set B is *many-one-hard* (*m-hard* for short) for a complexity class $\mathcal{C}$ if every $B \in \mathcal{C}$ m-reduces to B. If additionally $B \in \mathcal{C}$, then we say that B is *many-one-complete* (*m-complete* for short) for $\mathcal{C}$. Similarly, we define hardness and completeness for other reducibilities. We use the term $\mathcal{C}$-complete as an abbreviation for m-complete for $\mathcal{C}$.

A set L is *paddable* [6] if there exists $f(\cdot,\cdot)$, a polynomial-time computable, injective polynomial-time invertible function such that for all x and y, $x \in L \iff f(x,y) \in L$.

2.1 Weak Deterministic Self-correction

We introduce the notion of weak deterministic self-correction which is a deterministic variant of (probabilistic) self-correction [1]. The prefix *weak* indicates that our notion of deterministic self-correction does not necessarily imply probabilistic self-correction in the sense of Blum, Luby, and Rubinfeld [1]. The difference is as follows: For weak deterministic self-correction, we require that a sparse amount of errors can be corrected by a deterministic polynomial-time corrector. For probabilistic self-correction however, a probabilistic polynomial-time corrector must be able to correct up to $2^n/p(n)$ errors for some polynomial p.

Definition 1. *L is* weakly deterministically self-correctable *if for every polynomial q there exists a polynomial-time machine M such that $L \leq^{\mathrm{p}}_{\mathrm{T}} P$ via M whenever the census of $L \triangle P$ is bounded by q. If M queries nonadaptively, then L is* nonadaptively weakly deterministically self-correctable.

The set P in the definition formalizes a program for L that errs on at most $q(n)$ inputs of length n. So L is weakly deterministically self-correctable if there

exists an auxiliary machine M that corrects *all* programs that err on at most $q(n)$ inputs of length n. The next theorem shows that such a universal M surprisingly exists already if the single programs can be corrected with possibly different machines. This establishes the connection between weak deterministic self-correction and the robustness against false positives.

Theorem 1. *L is weakly deterministically self-correctable $\Leftrightarrow$ $L \leq_{\mathrm{T}}^{\mathrm{p}} L \triangle S$ for all sparse S.*

Proof

$\Rightarrow$: This is a direct consequence of Definition 1.

$\Leftarrow$: Assume that L is not weakly deterministically self-correctable. So there exists a polynomial q such that

$$\forall \text{polynomial-time machine } M,\ \exists T \subseteq \Sigma^* \ [\text{census}_T \leq q \text{ and } L \neq L(M^{L \triangle T})]. \tag{1}$$

We construct a sparse S such that $L \not\leq_{\mathrm{T}}^{\mathrm{p}} L \triangle S$. The construction is stagewise where in step i we construct a finite set S_i such that $S_1 \subseteq S_2 \subseteq \cdots$ and $S \stackrel{df}{=} \bigcup_{i \geq 1} S_i$.

Let $M_1, M_2, \ldots$ be an enumeration of all deterministic, polynomial-time Turing machines such that M_i runs in time $n^i + i$. Let $S_0 = \emptyset$. For $i \geq 1$, the set S_i is constructed as follows:

Choose n large enough such that $S_{i-1} \subseteq \Sigma^{<n}$ and changing the oracle with respect to words of length $\geq n$ will not affect the computations that were simulated in earlier steps. Choose a finite $T_i \subseteq \Sigma^{\geq n}$ and an $x_i \in \Sigma^*$ such that $\text{census}_{T_i} \leq q$ and

$$x_i \in L \Leftrightarrow x_i \notin L(M_i^{L \triangle (S_{i-1} \cup T_i)}). \tag{2}$$

Let $S_i \stackrel{df}{=} S_{i-1} \cup T_i$. We argue that the choice of T_i is possible. If not, then for all finite $T_i \subseteq \Sigma^{\geq n}$ where $\text{census}_{T_i} \leq q$ and all $x_i \in \Sigma^*$ it holds that

$$x_i \in L \Leftrightarrow x_i \in L(M_i^{L \triangle (S_{i-1} \cup T_i)}).$$

Let M be the polynomial-time machine obtained from M_i when queries of length $< n$ are answered according to $(L \triangle S_{i-1}) \cap \Sigma^{<n}$ (which is a finite set). So for all T where $\text{census}_T \leq q$ and all $x_i \in \Sigma^*$ it holds that

$$\begin{aligned} x_i \in L(M^{L \triangle T}) \Leftrightarrow x_i \in L(M_i^{L \triangle (S_{i-1} \cup (T \cap \Sigma^{\geq n}))}) &\Leftrightarrow x_i \in L(M_i^{L \triangle (S_{i-1} \cup T')}) \\ &\Leftrightarrow x_i \in L, \end{aligned}$$

where $T' = T \cap \Sigma^{\geq n} \cap \Sigma^{\leq |x_i|^i + i}$. Hence $L = L(M^{L \triangle T})$ for all T where $\text{census}_T \leq q$. So M contradicts (1). It follows that the choice of T_i is possible and hence also the construction of S.

The equivalence (2) makes sure that $\forall i \geq 1 \ [x_i \in L \Leftrightarrow x_i \notin L(M_i^{L \triangle S})]$ and hence $L \not\leq_{\mathrm{T}}^{\mathrm{p}} L \triangle S$. □

Corollary 1. *L is nonadaptively weakly det. self-correctable $\Leftrightarrow$ $L \leq_{\mathrm{tt}}^{\mathrm{p}} L \triangle S$ for all sparse S.*

3 Partly Corrupt NP-Hard Sets

We investigate how polynomial reductions can cope with sparse amounts of false data in sets that are hard for NP with respect to various reducibilities. In section 3.1 we show that altering sparse information in m-hard sets results in sets that are at least tt-hard. In particular, all m-complete sets are nonadaptively weakly deterministically self-correctable. Similarly, in section 3.2 we obtain that btt-hardness softens at most to T-hardness, if sparse information is altered. In particular, all btt-complete sets are weakly deterministically self-correctable. Moreover, we improve results by Ogiwara [3] and Fu [4], and show that no btt-hard set is p-close to P, unless P = NP. In section 3.3 we prove that adding a sparse amount of false positives to dtt-hard sets results in sets that are at least T-hard. However, it remains open whether dtt-complete sets are weakly deterministically self-correctable. At the end of section 3.3, we give evidence that this open problem is rather difficult to solve.

Finally, in subsection 3.4 we show that many-one reductions, bounded truth-table reductions, and disjunctive truth-table reductions are provably too weak to handle false positives in SAT.

3.1 Many-One Reductions

Here we alter sparse information in m-hard sets for NP. Under the assumption P $\neq$ NP, the resulting sets are still ctt-hard. Without the assumption, we can show that the resulting sets are at least tt-hard.

On the technical side we extend an idea from [7] which shows how many-one queries to NP-hard sets can be reformulated. In this way, for a given query we can generate a polynomial number of different, but equivalent queries (Lemma 1). From this we easily obtain the conditional ctt-hardness and the unconditional tt-hardness of the altered NP-hard set. As a corollary, all m-complete sets for NP are nonadaptively weakly deterministically self-correctable.

Lemma 1. *Let L be $\leq_{\mathrm{m}}^{\mathrm{p}}$-hard for* NP *and let $B \in$ NP. Then there exists a polynomial r such that for every polynomial q there is a polynomial-time algorithm $\mathcal{A}$ such that $\mathcal{A}$ on input x,*

- *either correctly decides the membership of x in B*
- *or outputs $k = q(r(|x|))$ pairwise disjoint $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$ such that for all $i \in [1, k]$,*

$$x \in B \Leftrightarrow y_i \in L.$$

Proof. Choose $R \in$ P and a polynomial p such that $x \in B$ if and only if there exists a $w \in \Sigma^{p(|x|)}$ such that $(x, w) \in R$. For $x \in B$, let w_x be the lexicographically greatest such witness. The following set is in NP.

$$\mathrm{Left}(B) = \{(x, y) \,\big|\, x \in B, |y| = p(|x|), y \leq w_x\}.$$

So there is a many-one reduction f from Left(B) to L. In particular, there exists a polynomial r such that for all $x \in \Sigma^*$ and all $y \in \Sigma^{p(|x|)}$, $|f(x, y)| \leq r(|x|)$. Choose a polynomial q. We now describe the algorithm $\mathcal{A}$.

```
1  // input x, |x| = n
2  m := p(n)
3  if (x, 1^m) ∈ R then accept
4  l := 0^m
5  if f(x,l) = f(x, 1^m) then reject
6  Q = {f(x,l)}
7  while |Q| ≤ q(r(n)) do
8     choose a ∈ Σ^m such that l ≤ a ≤ 1^m, f(x,a) ∈ Q, f(x,a+1) ∉ Q
9     l := a + 1
10    if (x,a) ∈ R then accept
11    if f(x,l) = f(x, 1^m) then reject
12    Q = Q ∪ {f(x,l)}
13 end while
14 output Q
```

Observe that the algorithm places a string $f(x,l)$ in Q only if $f(x,l) \neq f(x,1^m)$. Thus $f(x,1^m)$ is never placed in Q. So in step 8, $f(x,l) \in Q$ and $f(x,1^m) \notin Q$. Therefore, with binary search we find the desired a in polynomial time. Every iteration of the while loop adds a new string to Q or decides the membership of x in B. Thus the algorithm works in polynomial time and when it outputs some Q, then $|Q| = q(r(|x|))$ and words in Q have lengths $\leq r(n)$.

Claim 1. *If the algorithm outputs some Q, then for all $y \in Q$, $x \in B \Leftrightarrow y \in L$.*

Proof of the claim. If $x \notin B$, then for all $c \in [0^m, 1^m]$, $(x,c) \notin \mathrm{Left}(B)$. Observe that the algorithm places a string y in Q only if $y = f(x,a)$ where $a \in [0^m, 1^m]$. Since f is a many-one reduction from $\mathrm{Left}(B)$ to L, no string from Q belongs to L.

From now on we assume $x \in B$. We prove the claim by induction. Initially, $Q = \{f(x,0^m)\}$. Clearly, $x \in B \Leftrightarrow (x,0^m) \in \mathrm{Left}(B)$. Since f is a many-one reduction from $\mathrm{Left}(B)$ to L, the claim holds initially. Assume that the claim holds before an iteration of the while loop. The while loop finds a node a such that $f(x,a) \in Q$, but $f(x,a+1) \notin Q$. From $f(x,a) \in Q$ and $x \in B$ it follows (by induction hypothesis) that $f(x,a) \in L$. Thus $(x,a) \in \mathrm{Left}(B)$ which implies $a \leq w_x$. At this point the algorithm checks whether a is a witness of x. If so, then it accepts and halts. Otherwise, we have $a+1 \leq w_x$. Thus $(x,a+1) \in \mathrm{Left}(B)$ and $f(x,a+1) \in L$. So the claim also holds after the iteration of the while loop. ◇

Claim 2. *If the algorithm accepts x (resp., rejects x), then $x \in B$ (resp., $x \notin B$).*

Proof of the claim. The algorithm accepts x only if it finds a witness of x. Thus if the algorithm accepts, then $x \in B$. The algorithm rejects only if $f(x,l) = f(x,1^m)$. Note that $f(x,l) \in Q$, so by the previous claim, $x \in B \Leftrightarrow f(x,l) \in L$. Observe that $(x,1^m) \notin \mathrm{Left}(B)$. Thus $f(x,l) = f(x,1^m) \notin L$ and hence $x \notin B$. ◇

This finishes the proof of the lemma. □

Theorem 2. *The following statements are equivalent.*

1. $\mathrm{P} \neq \mathrm{NP}$
2. *If* L *is* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-hard for* NP *and* S *is sparse, then* $L \cup S$ *is* $\leq^{\mathrm{p}}_{\mathrm{ctt}}$*-hard for* NP.

Proof

2 $\Rightarrow$ **1:** If $\mathrm{P} = \mathrm{NP}$, then $L = \Sigma^* - \{0\}$ and $S = \{0\}$ are counter examples for 2.

1 $\Rightarrow$ **2:** Assume $\mathrm{P} \neq \mathrm{NP}$ and let L and S be as in statement 2. If $\overline{L}$ is sparse, then there exist sparse coNP-hard sets and hence $\mathrm{P} = \mathrm{NP}$ [8]. So it follows that $\overline{L}$ is not sparse and $L \cup S \neq \Sigma^*$. Hence there exist elements $x_0 \notin L \cup S$ and $x_1 \in L \cup S$.

Let $B \in \mathrm{NP}$; we show $B \leq^{\mathrm{p}}_{\mathrm{ctt}} L \cup S$. First, choose the polynomial r according to Lemma 1. Let q be a polynomial such that $|S \cap \Sigma^{\leq n}| < q(n)$. Lemma 1 provides an algorithm $\mathcal{A}$ that on input x either correctly decides the membership of x in B, or outputs $k = q(r(|x|))$ pairwise disjoint $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$ such that for all $i \in [1,k]$, $(x \in B \Leftrightarrow y_i \in L)$. Define the following polynomial-time-computable function.

$$g(x) \stackrel{df}{=} \begin{cases} x_0 & : \text{ if } \mathcal{A}(x) \text{ rejects} \\ x_1 & : \text{ if } \mathcal{A}(x) \text{ accepts} \\ (y_1, \ldots, y_k) & : \text{ if } \mathcal{A}(x) \text{ returns } Q = \{y_1, \ldots, y_k\} \end{cases}$$

Note that in the last case, $k = q(r(|x|))$ and the y_i have lengths $\leq r(|x|)$. So at least one of the y_i does not belong to S. From $\mathcal{A}$'s properties stated in Lemma 1 it follows that $B \leq^{\mathrm{p}}_{\mathrm{ctt}} L \cup S$ via g. □

Theorem 3. *If* L *is* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-hard for* NP *and* S *is sparse, then* $L \triangle S$ *is* $\leq^{\mathrm{p}}_{\mathrm{tt}}$*-hard for* NP.

Proof. For $B \in \mathrm{NP}$ we show $B \leq^{\mathrm{p}}_{\mathrm{tt}} L \triangle S$. First, choose the polynomial r according to Lemma 1. Let q be a polynomial such that $2 \cdot |S \cap \Sigma^{\leq n}| < q(n)$. Lemma 1 provides an algorithm $\mathcal{A}$ that on input x either correctly decides the membership of x in B, or outputs $k = q(r(|x|))$ pairwise disjoint $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$ such that for all $i \in [1,k]$, $(x \in B \Leftrightarrow y_i \in L)$. We describe a polynomial-time oracle machine M on input x: If $\mathcal{A}(x)$ accepts, then M accepts. If $\mathcal{A}(x)$ rejects, then M rejects. Otherwise, $\mathcal{A}(x)$ returns elements $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$. M queries all these elements and accepts if and only if at least $k/2$ of the answers were positive.

Clearly, if $\mathcal{A}(x)$ accepts or rejects, then $(x \in B \Leftrightarrow M(x)$ accepts). So assume that $\mathcal{A}(x)$ returns elements y_i. S contains less than $q(r(|x|))/2 = k/2$ words of length $\leq r(|x|)$. So more than $k/2$ of the y_i do not belong to S. Hence, for more than $k/2$ of the y_i it holds that

$$x \in B \quad \Leftrightarrow \quad y_i \in L \quad \Leftrightarrow \quad y_i \in L \triangle S.$$

Therefore, x belongs to B if and only if at least $k/2$ of the y_i belong to $L \triangle S$. This shows that $B \leq^{\mathrm{p}}_{\mathrm{tt}} L \triangle S$ via M. □

Corollary 2. *If L is $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard for* NP *and S is sparse, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for* NP.

Corollary 3. *All $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets for* NP *are nonadaptively weakly deterministically self-correctable.*

3.2 Bounded Truth-Table Reductions

We show that altering sparse information in btt-hard sets for NP results in sets that are still T-hard for NP. Our proof builds on techniques by Ogihara and Watanabe [9] and Ogiwara and Lozano [10]. First, in Lemma 2 we isolate the combinatorial argument for the case that a Turing machine has oracle access to the symmetric difference of a btt-hard set B and a sparse set S. Then, with this argument at hand, we perform an Ogihara-Watanabe-tree-pruning in the computation tree of a given NP-machine (Theorem 4). Finally this shows that the acceptance of the latter machine can be determined in polynomial time with access to the oracle $B \triangle S$. As a corollary we obtain that all btt-complete sets in NP are weakly deterministically self-correctable (Corollary 4). Moreover, we obtain the following improvement of results by Ogiwara [3] and Fu [4]: No btt-hard set for NP is p-close to P, unless P = NP.

For our combinatorial argument we need to define the following polynomials r_k for $k \geq 0$.

$$r_0(n) \stackrel{df}{=} 2$$
$$r_k(n) \stackrel{df}{=} 2^k(2kn+2)(r_{k-1}(n))^k \quad \text{for } k \geq 1$$

Lemma 2. *For every $k \geq 0$ there exists a polynomial-time oracle transducer M_k with the following properties: For every input $(0^n, V)$ where $V = (v_{i,j}) \in (\Sigma^{\leq n})^{k \times r_k(n)}$ and for all sets $B, S \subseteq \Sigma^{\leq n}$ where $|S| \leq n$ the computation $M_k^{B \triangle S}(0^n, V)$ outputs some $b \in (1, r_k(n)]$ such that*

$$\exists a, c \in [1, r_k(n)] \text{ such that } a < b \leq c \text{ and } \forall i \in [1,k], (v_{i,a} \in B \Leftrightarrow v_{i,c} \in B).$$

Theorem 4. *If B is $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for* NP *and S is sparse, then $B \triangle S$ is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP.

Corollary 4. *All $\leq^{\mathrm{p}}_{\mathrm{btt}}$-complete sets for* NP *are weakly deterministically self-correctable.*

Yesha [2] defined two sets A and B to be close if the census of their symmetric difference, $A \triangle B$, is a slowly increasing function. Accordingly, A and B are p-close, if the census of $A \triangle B$ is polynomially bounded. A is p-close to a complexity class $\mathcal{C}$, if there exists some $B \in \mathcal{C}$ such that A and B are p-close.

Yesha [2] poses the question of whether $\leq^{\mathrm{p}}_{\mathrm{m}}$- or $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard sets for NP can be p-close to P (assuming P $\neq$ NP). Schning [11] showed that no $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard set for NP is p-close to P, unless PH = Δ^{P}_2. Ogiwara [3] and Fu [4] proved that no $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard set for NP is p-close to P, unless P = NP.

We can strengthen the latter result as follows.

Corollary 5. *No $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard set for* NP *is p-close to* P*, unless* P = NP.

3.3 Disjunctive Truth-Table Reductions

In this section we analyze how disjunctive truth-table reductions can handle false positives. We show that the union of dtt-hard sets with arbitrary sparse sets is always T-hard.

Theorem 5. *Let L be $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for* NP*, and let S be a sparse set. Then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*.*

Contrary to sections 3.1 and 3.2, we do not know how dtt-reductions react towards false negatives. For that reason, we cannot deduce that dtt-complete sets are weakly deterministically self-correctable. We can provide evidence that the question is indeed difficult. We explain that it is related to the longstanding open question [12] of whether the existence of sparse dtt-complete sets implies $\mathrm{P} = \mathrm{NP}$.

Corollary 6. *If dtt-complete sets for* NP *are weakly deterministically self-correctable, then the existence of sparse dtt-complete sets for* NP *implies* $\mathrm{P} = \mathrm{NP}$*.*

Proof. We assume that dtt-complete sets for NP are weakly deterministically self-correctable and that there exists a sparse set L such that L is dtt-complete for NP. Since L is weakly deterministically self-correctable, it follows from Theorem 1 that for all sparse sets S, $L \leq^{\mathrm{p}}_{\mathrm{T}} L \triangle S$. It follows that $L \leq^{\mathrm{p}}_{\mathrm{T}} L \triangle L$ and hence $L \leq^{\mathrm{p}}_{\mathrm{T}} \emptyset$. This implies $\mathrm{P} = \mathrm{NP}$. □

3.4 Non-robustness against Sparse Sets of False Positives

So far we concentrated on reductions strong enough to manage partly corrupt NP-hard sets. Now we ask for reductions that are provably too weak to handle such corrupt information. Under the assumption $\mathrm{P} \neq \mathrm{NP}$ we show that many-one reductions, bounded truth-table reductions, and disjunctive truth-table reductions are weak in this sense. More precisely, altering sparse information in SAT can result in sets that are not $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard, not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard, and not $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for NP. On the other hand, Corollary 7 shows that similar results for $\leq^{\mathrm{p}}_{\mathrm{ctt}}$, $\leq^{\mathrm{p}}_{\mathrm{tt}}$, and $\leq^{\mathrm{p}}_{\mathrm{T}}$ would imply the existence of NP-complete sets that are not paddable. This explains that such results are hard to obtain.

Theorem 6. *The following statements are equivalent.*

1. $\mathrm{P} \neq \mathrm{NP}$
2. *There exists a sparse S such that* $\mathrm{SAT} \cup S$ *is not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for* NP*.*
3. *There exists a sparse S such that* $\mathrm{SAT} \cup S$ *is not $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for* NP*.*

Proof

1 ⇒ 2: Assume $\mathrm{P} \neq \mathrm{NP}$ and let $M_1, M_2, \ldots$ be an enumeration of polynomial-time oracle Turing machines such that M_i runs in time $n^i + i$ and queries at most i strings (so the machines represent all $\leq^{\mathrm{p}}_{\mathrm{btt}}$-reduction functions). We construct an increasing chain of sets $S_1 \subseteq S_2 \subseteq \cdots$ and finally let $S \stackrel{df}{=} \bigcup_{i \geq 1} S_i$. Let $S_0 \stackrel{df}{=} \{\varepsilon\}$ and define S_k for $k \geq 1$ as follows:

1. let n be greater than k and greater than the length of the longest word in S_{k-1}
2. let $T \stackrel{df}{=} (\mathrm{SAT} \cap \Sigma^{\leq n}) \cup S_{k-1} \cup \Sigma^{>n}$
3. choose a word x such that $M_k^T(x)$ accepts if and only if $x \notin \mathrm{SAT}$
4. let Q be the set of words that are queried by $M_k^T(x)$ and that are longer than n
5. let $S_k \stackrel{df}{=} S_{k-1} \cup Q$

We first observe that the x in step 3 exists: If not, then $L(M_k^T) = \mathrm{SAT}$ and T is cofinite. Hence $\mathrm{SAT} \in \mathrm{P}$ which is not true by assumption. So the described construction is possible.

If a word w of length j is added to S in step k (i.e., $w \in S_k - S_{k-1}$), then in all further steps, no words of length j are added to S (i.e., for all $i > k$, $S_i \cap \Sigma^j = S_k \cap \Sigma^j$). In the definition of S_k it holds that $|Q| \leq k \leq n$. So in step 5, at most n words are added to S and these words are of length greater than n. Therefore, for all $i \geq 0$, $|S \cap \Sigma^i| \leq i$ and hence S is sparse.

Assume $\mathrm{SAT} \cup S$ is $\leq_{\mathrm{btt}}^{\mathrm{p}}$-hard for NP. So there exists a $k \geq 1$ such that $\mathrm{SAT} \leq_{\mathrm{btt}}^{\mathrm{p}} \mathrm{SAT} \cup S$ via M_k. Consider the construction of S_k and let n, T, x, and Q be the corresponding variables. In all steps $i \geq k$, S will be only changed with respect to words of lengths greater than n. Therefore, $S \cap \Sigma^{\leq n} = S_{k-1}$ and hence

$$\forall w \in \Sigma^{\leq n}, (w \in \mathrm{SAT} \cup S \;\Leftrightarrow\; w \in T). \tag{3}$$

If q is an oracle query of $M_k^T(x)$ that is longer than n, then $q \in Q$ and hence $q \in S_k \subseteq S$. So $q \in \mathrm{SAT} \cup S$ and $q \in T$. Together with (3) this shows that the computations $M_k^T(x)$ and $M_k^{\mathrm{SAT} \cup S}(x)$ are equivalent. From step 3 it follows that $M_k^{\mathrm{SAT} \cup S}(x)$ accepts if and only if $x \notin \mathrm{SAT}$. This contradicts the assumption that M_k reduces SAT to $\mathrm{SAT} \cup S$. Hence $\mathrm{SAT} \cup S$ is not $\leq_{\mathrm{btt}}^{\mathrm{p}}$-hard for NP.

2 ⇒ 1: If P = NP, then for all sparse S, $\mathrm{SAT} \cup S$ is trivially $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete for NP.

1 ⇔ 3: Analogous to the equivalence of 1 and 2; we only sketch the differences. We use an enumeration of $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-reduction machines (i.e., machines that nonadaptively query an arbitrary number of strings and that accept if at least one query is answered positively). Moreover, we change the definition of S_k in step 5 such that

$$S_k \stackrel{df}{=} \begin{cases} S_{k-1} & : \text{ if } Q = \emptyset \\ S_{k-1} \cup \{q\} & : \text{ if } Q \neq \emptyset, \text{ where } q = \max(Q). \end{cases}$$

This makes sure that S is sparse.

Assume $\mathrm{SAT} \leq_{\mathrm{dtt}}^{\mathrm{p}} \mathrm{SAT} \cup S$ via M_k. If no query of $M_k^T(x)$ is longer than n, then $M_k^T(x)$ and $M_k^{\mathrm{SAT} \cup S}(x)$ are equivalent computations and hence $L(M_k^{\mathrm{SAT} \cup S}) \neq \mathrm{SAT}$ by step 3. Otherwise, there exists a query that is longer than n. Let q be the greatest such query and note that $q \in S_k \subseteq S$. This query gets a positive answer in the computation $M_k^T(x)$. So the computation accepts and by step 3, $x \notin \mathrm{SAT}$.

In the computation $M_k^{\mathrm{SAT}\cup S}(x)$, the query q also obtains a positive answer and hence the computation accepts. So also in this case, $L(M_k^{\mathrm{SAT}\cup S}) \neq \mathrm{SAT}$. This shows that $\mathrm{SAT} \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for NP. □

Theorem 6 tells us that while $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard, $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard, and $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard sets do not become too easy when false positives are added (as they stay NP-hard with respect to more general reducibilities, confer sections 3.1, 3.2, and section 3.3), they are not robust against sparse sets of false positives. The next result says that this is different for hard sets which are paddable.

Proposition 1. *Let L be paddable and let S be sparse.*

1. *If L is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for* NP*, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for* NP*.*
2. *If L is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*.*
3. *If L is $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-hard for* NP*, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-hard for* NP*.*

In Theorem 6 we have seen that $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete, $\leq^{\mathrm{p}}_{\mathrm{btt}}$-complete, and $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets are not robust against sparse sets of false positives. The following corollary of Proposition 1 explains the difficulty of showing the same for $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-complete, $\leq^{\mathrm{p}}_{\mathrm{tt}}$-complete, and $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete sets.

Corollary 7

1. *If there exists a $\leq^{\mathrm{p}}_{\mathrm{tt}}$-complete set L in* NP *and a sparse S such that $L \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for* NP*, then there exist $\leq^{\mathrm{p}}_{\mathrm{tt}}$-complete sets in* NP *that are not paddable.*
2. *If there exists a $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete set L in* NP *and a sparse S such that $L \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*, then there exist $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete sets in* NP *that are not paddable.*
3. *If there exists a $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-complete set L in* NP *and a sparse S such that $L \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-hard for* NP*, then there exist $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-complete sets in* NP *that are not paddable.*

References

1. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. Journal of Computer and System Sciences 47(3), 549–595 (1993)
2. Yesha, Y.: On certain polynomial-time truth-table reducibilities of complete sets to sparse sets. SIAM Journal on Computing 12(3), 411–425 (1983)
3. Ogiwara, M.: On P-closeness of polynomial-time hard sets (manuscript, 1991)
4. Fu, B.: On lower bounds of the closeness between complexity classes. Mathematical Systems Theory 26(2), 187–202 (1993)
5. Ladner, R.E., Lynch, N.A., Selman, A.L.: A comparison of polynomial time reducibilities. Theoretical Computer Science 1, 103–123 (1975)
6. Berman, L., Hartmanis, J.: On isomorphism and density of NP and other complete sets. SIAM Journal on Computing 6, 305–322 (1977)

7. Glaßer, C., Pavan, A., Selman, A.L., Zhang, L.: Redundancy in complete sets. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 444–454. Springer, Heidelberg (2006)
8. Fortune, S.: A note on sparse complete sets. SIAM Journal on Computing 8(3), 431–433 (1979)
9. Ogiwara, M., Watanabe, O.: On polynomial-time bounded truth-table reducibility of NP sets to sparse sets. SIAM Journal on Computing 20(3), 471–483 (1991)
10. Ogiwara, M., Lozano, A.: On sparse hard sets for counting classes. Theoretical Computer Science 112(2), 255–275 (1993)
11. Schöning, U.: Complete sets and closeness to complexity classes. Mathematical Systems Theory 19(1), 29–41 (1986)
12. Hemachandra, L.A., Ogiwara, M., Watanabe, O.: How hard are sparse sets? In: Structure in Complexity Theory Conference, pp. 222–238 (1992)

Termination of Priority Rewriting

Isabelle Gnaedig

INRIA & LORIA
BP 101, 54602 Villers-lès-Nancy Cedex France

Abstract. Introducing priorities in rewriting increases the expressive power of rules and helps to limit computations. Priority rewriting is used in rule-based programming as well as in functional programming. Termination of priority rewriting is then important to guarantee that programs give a result. We describe an inductive proof method for termination of priority rewriting, relying on an explicit induction on the termination property and working by generating proof trees, which model the rewriting relation by using abstraction and narrowing.

1 Introduction

In [1,2], priority rewriting systems (PRSs in short) have been introduced. A PRS is a term rewrite system (TRS in short) with a partial ordering on rules, determining a priority between some of them. Considering priorities on the rewrite rules to be used can be very useful for an implementation purpose, to reduce the non-determinism of computations or to enable divergent systems to terminate, and for a semantical purpose, to increase the expressive power of rules. Priority rewriting is enabled in rule-based languages like ASF+SDF [3] or Maude [4]. It is also used as a computation model for functional programming [5], and is underlying in the functional strategy, used for example in Lazy ML [6], Clean [7], or Haskell [8]. Let us also cite recent works on specification and correctness of security policies using rewriting with priorities [9,10].

But priority rewriting is delicate to handle. First, the priority rewriting relation is not always decidable, because a term rewrites with a given rule only if in the redex, there is no reduction leading to another redex, reducible with a rule of higher priority. A way to overcome the undecidability can be to force evaluation of the terms in reducing subterms to strong head normal form via some strategy [5], or to use the innermost strategy [11]. But in these cases, normalization can lead to non-termination.

Second, the semantics of a PRS is not always clearly defined. In [1], a semantics is proposed, relying on a notion of unique sound and complete set of closed instances of the rules of the PRS, and it is shown that bounded -the bounded property is weaker than termination- PRSs have a semantics. In [12], a fixed point based technique is proposed to compute the semantics of a PRS. It is also proved that for a bounded PRS with finitely many rules, the set of successors of any term is finite and computable. In [11], a logical semantics of PRSs based on equational logic is given. A particular class of PRSs is proved sound and

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 386–397, 2009.

complete with respect to the initial algebra, provided every priority rewriting sequence from every ground term terminates.

Then the termination problem of the priority rewriting relation naturally arises, either to guarantee that it has a semantics, or to ensure that rewriting computations always give a result. Surprisingly, it seems not to have been much investigated until now. Let us cite [13], discussing a normalizing strategy of PRSs i.e., a strategy giving only finite derivations for terms having a normal form with usual rewriting, and [14], where it is proved that termination of innermost rewriting implies termination of generalized innermost rewriting with ordered rules. But to our knowledge, the problem of finding a specific termination proof technique has only been addressed in [11], where the use of reduction orderings is extended with an instantiation condition on rules linked with the priority order.

Our purpose here is to consider the termination proof of priority rewriting from an operational point of view, with the concern of guaranteeing a result for every computation. So it seems interesting to focus on the innermost priority rewriting of [11], because it is decidable, easy to manipulate, and the innermost strategy is often used in programming contexts where priorities on rules are considered. The previously cited works on rule-based security policies also generated a need of specific termination tools: the specifications given in [9] have indeed been executed in TOM [15] with an innermost evaluation mechanism.

We use an inductive approach, whose principle has already been applied for termination of rewriting under strategies [16]. The idea is to prove, in developing proof trees simulating the rewriting trees, that every derivation starting from any term terminates, supposing that it is true for terms smaller than the starting terms. We then introduce the priority notion in the generation mechanism of the proof trees, and show how to optimize the technique in this new case.

2 Priority Rewriting

We assume that the reader is familiar with the basic definitions and notations of term rewriting given for instance in [17,18]. The ones needed in the paper can also be found in [19]. We just recall that $\mathcal{T}(\mathcal{F},\mathcal{X})$ is the set of terms built from a given finite set $\mathcal{F}$ of function symbols f, and a set $\mathcal{X}$ of variables denoted by $x, y \ldots$. $\mathcal{T}(\mathcal{F})$ is the set of ground terms (without variables). *Positions* in a term are represented as sequences of integers. The empty sequence ϵ denotes the top position. For a position p of a term t, we denote by $t|_p$ the subterm of t at position p, and by $t[s]_p$ the term obtained by replacing in t the subterm $t|_p$ by s. A *substitution* is an assignment from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F},\mathcal{X})$, written $\sigma = (x_1 = t_1, \ldots, x_n = t_n)$ whose domain, denoted by $\mathcal{D}om(\sigma)$, is $\{x_1, \ldots, x_n\}$. It uniquely extends to an endomorphism of $\mathcal{T}(\mathcal{F},\mathcal{X})$. Its application to a term $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ is written σt. An *instantiation* or ground substitution is an assignment from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F})$. Id denotes the identity substitution. Given a term rewrite system $\mathcal{R}$, a function symbol in $\mathcal{F}$ is called a *constructor* iff it does not occur in $\mathcal{R}$ at the top position of a left-hand side (*lhs* in short) of rule, and is called a *defined function symbol* otherwise. The set of defined function symbols is denoted by $\mathcal{D}$.

A *priority term rewrite system* is a pair $(\mathcal{R}, \blacktriangleright)$ of a term rewrite system $\mathcal{R}$ (always considered as finite here) and a partial ordering $\blacktriangleright$ on the rules of $\mathcal{R}$. A rule r_1 has a higher priority than a rule r_2 iff $r_1 \blacktriangleright r_2$, which is also written $\downarrow_{r_2}^{r_1}$.

Definition 1 ([11]). *Let $\mathcal{R}$ be a PRS on $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A term s is IP-reducible and (IP-) rewrites to t at position p with the rule $l \to r$, and the substitution σ which is written $s \to^{IP}_{p,l\to r,\sigma} t$ iff:*

- *s rewrites into t : $t = s[\sigma r]_p$ with $s|_p = \sigma l$,*
- *no proper subterm of the redex $s|_p$ is $IP-$reducible,*
- *$s|_p$ is not $IP-$reducible by any rule in $\mathcal{R}$ of higher priority than $l \to r$.*

Example 1. With the PRS $\{f(g(x)) \to b, g(a) \to c \blacktriangleright g(a) \to d\}$, on $f(g(a))$, the first rule should apply, but this would not be an innermost rewrite step. So the second rule applies, but the third one does not, because $g(a) \to c \blacktriangleright g(a) \to d$.

A PRS $\mathcal{R}$ *IP-terminates* if and only if every IP-rewriting chain (IP-derivation) of the rewriting relation induced by $\mathcal{R}$ is finite. If t' is in an IP-derivation issued from t and t' is IP-irreducible, then t' is called a(n) *(IP-)normal form* of t and is denoted by $t{\downarrow}$. Note that given t, $t{\downarrow}$ may be not unique.

3 Inductively Proving Termination of IP-Rewriting

We prove termination of IP-rewriting by induction on the ground terms. Working on ground terms is appropriate, since most of the time, the algebraic semantics of rule-based languages is initial. Moreover, in [11], to guarantee stability by substitution of the innermost rewriting relation, the rules without highest priority only can reduce ground terms. Finally, there are TRSs which are non-innermost terminating on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and innermost terminating on $\mathcal{T}(\mathcal{F})$. A termination proof method working on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ could not handle them.

For proving that a PRS on $\mathcal{T}(\mathcal{F})$ IP-terminates, we reason with a local notion of termination on terms: a term t of $\mathcal{T}(\mathcal{F})$ is said to be *IP-terminating* for a PRS $\mathcal{R}$ if every IP-rewriting chain starting from t is finite.

For proving that a term t of $\mathcal{T}(\mathcal{F})$ is IP-terminating, we then proceed by induction on $\mathcal{T}(\mathcal{F})$ with a noetherian ordering $\succ$, assuming the property for every t' such that $t \succ t'$. To guarantee non emptiness of $\mathcal{T}(\mathcal{F})$, we assume that $\mathcal{F}$ contains at least one constructor constant.

Rewriting derivations are simulated, using a lifting mechanism, by proof trees developed from initial patterns $t_{ref} = g(x_1, \ldots, x_m)$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, for every $g \in \mathcal{D}$, by alternatively using narrowing and an abstraction mechanism. For a term t of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ occurring in a proof tree,

– first, some subterms $\theta t|_j$ of θt are supposed to be IP-terminating for every instantiation θ by the induction hypothesis, if $\theta t_{ref} \succ \theta t|_j$ for the induction ordering $\succ$. So the $t|_j$ are replaced in t by *abstraction variables* X_j representing respectively any of their normal forms. Reasoning by induction allows us to only suppose the existence of the normal forms *without explicitly computing them*. Obviously, the whole term t can be abstracted. If the ground

instances of the resulting term are IP-terminating (either if the induction hypothesis can be applied to them, or if they can be proved IP-terminating by other means presented later), then the ground instances of the initial term are IP-terminating. Otherwise,

– the resulting term $u = t[X_j]_{\{i_1,\ldots,i_p\}}$ ($i_1, \ldots, i_p$ are the abstraction positions in t) is narrowed in all possible ways into terms v, with an appropriate narrowing relation corresponding to IP-rewriting u according to the possible instances of the X_j.

The process is iterated on each v, until we get a term t' such that either $\theta t_{ref} \succ \theta t'$, or $\theta t'$ can be proved IP-terminating.

This technique was inspired from the one we proposed for proving the innermost termination of classical rewrite systems in [16]. We now give the concepts needed to formalize and automate it.

4 Abstraction, Narrowing, Constraints

Ordering Constraints. The induction ordering is not defined a priori but is constrained along the proof by inequalities between terms that must be comparable, called *ordering constraints*, each time the induction hypothesis is used for abstraction. More formally, an *ordering constraint* is a pair of terms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ denoted by $(t > t')$. It is said to be *satisfiable* if there is an ordering $\succ$, such that for every instantiation θ whose domain contains $\mathcal{V}ar(t) \cup \mathcal{V}ar(t')$, we have $\theta t \succ \theta t'$. We say that $\succ$ satisfies $(t > t')$. A conjunction C of ordering constraints is satisfiable if there is an ordering satisfying all conjuncts. The empty conjunction, always satisfied, is denoted by $\top$.

Satisfiability of a constraint conjunction of this form is undecidable. But a sufficient condition for C to be satisfiable is to find a simplification ordering $\succ$ such that $t \succ t'$ for every constraint $t > t'$ of C. The constraints generated by our approach are often satisfiable by a Recursive Path Ordering or a Lexicographic Path Ordering. Otherwise, automatic constraint solvers can provide adequate polynomial orderings. See [16] for experiments.

Abstraction. Let $\mathcal{N}$ be a set of variables disjoint from $\mathcal{X}$. Symbols of $\mathcal{N}$ are called *abstraction variables*. Substitutions and instantiations are extended to $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ so that for every substitution σ (resp. instantiation θ) such that $\mathcal{D}om(\sigma)$ (resp. $\mathcal{D}om(\theta)$) contains a variable $X \in \mathcal{N}$, σX (resp. θX) is in IP-normal form. The formal definition of a term abstraction can be found in [16].

IP-termination on $\mathcal{T}(\mathcal{F})$ is in fact proved by reasoning on terms with abstraction variables i.e., on terms of $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. Ordering constraints are extended on $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. When a subterm $t|_j$ is abstracted by X_j, we state an *abstraction constraint* $t|_j\downarrow = X_j$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ and $X \in \mathcal{N}$, to express that its instances can only be normal forms of the corresponding instances of $t|_j$.

Narrowing. After abstracting the current term t into $t' = t[X_j]_{j \in \{i_1,\ldots,i_p\}}$, we test whether the possible ground instances of t' are reducible, according to the possible values of the instances of the X_j, by narrowing t' with the PRS.

To schematize innermost rewriting on ground terms, in [16], we introduced a specific narrowing definition involving constrained substitutions. The problem here is to see how to express priorities in the narrowing mechanism and how to integrate them in the previous constraint based definition. In [16], the usual notion of narrowing was refined as follows. With the usual innermost narrowing relation, if a position p in a term t is a narrowing position, no suffix position of p can be a narrowing position as well. However, if we consider ground instances of t, we can have rewriting positions p for some instances, and p' for other instances, such that p' is a suffix position of p. So, when using the narrowing relation to schematize innermost rewriting of ground instances of t, the narrowing positions p to consider depend on a set of ground instances of t, which is defined by excluding the ground instances of t that would be narrowable at some suffix position of p. For instance, with the TRS $R = \{g(a) \to a, f(g(x)) \to b\}$, the innermost narrowing positions of the term $f(g(X))$ are 1 with the narrowing substitution $\sigma = (X = a)$, and ϵ with any σ such that $\sigma X \neq a$. This leads us to introduce constrained substitutions.

Let σ be a substitution on $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. In the following, we identify $\sigma = (x_1 = t_1, \ldots, x_n = t_n)$ with the equality formula $\bigwedge_i (x_i = t_i)$, with $x_i \in \mathcal{X} \cup \mathcal{N}$, $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. Similarly, we call *negation* $\overline{\sigma}$ of the substitution σ the formula $\bigvee_i (x_i \neq t_i)$. The negation of Id means that no substitution can be applied.

Definition 2 ([16]). *A substitution σ is said* to satisfy *a constraint $\bigwedge_j \bigvee_{i_j} (x_{i_j} \neq t_{i_j})$, iff for every ground instantiation θ, $\bigwedge_j \bigvee_{i_j} (\theta\sigma x_{i_j} \neq \theta\sigma t_{i_j})$. A* constrained substitution *σ is a formula $\sigma_0 \wedge \bigwedge_j \bigvee_{i_j} (x_{i_j} \neq t_{i_j})$, where σ_0 is a substitution, and $\bigwedge_j \bigvee_{i_j} (x_{i_j} \neq t_{i_j})$ the constraint to be satisfied by σ_0.*

Definition 3 (Innermost narrowing [16]). *Let $\mathcal{R}$ be a TRS. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$* innermost narrows *into a term $t' \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ at the non-variable position p of t, using the rule $l \to r \in \mathcal{R}$ with the constrained substitution $\sigma = \sigma_0 \wedge \bigwedge_{j \in [1..k]} \overline{\sigma_j}$, which is written $t \rightsquigarrow^{Inn}_{p,l\to r,\sigma} t'$, iff $t' = \sigma_0(t[r]_p)$, where σ_0 is the most general unifier (*mgu*) of $t|_p$ and l and $\sigma_j, j \in [1..k]$ are all* mgus *of $\sigma_0 t|_{p'}$ and a* lhs *l' of a rule of $\mathcal{R}$, for all suffix position p' of p in t.*

It is always assumed that there is no variable in common between the rule and the term i.e., that $Var(l) \cap Var(t) = \emptyset$. In the following, we are only interested in the restriction of the narrowing substitution applied to the current term t. We then omit its definition on the variables of the *lhs* of rules.

Now, we have to see how to simulate the IP-rewriting steps of a given term following the possible instances of its variables, by narrowing it with the rules, considering their priority. Unlike for simulating rewriting without priorities, where the narrowing process only depends on the term to be rewritten and of the rule considered, simulating IP-rewriting of ground instances of a term with a given rule requires to take into account the narrowing steps with the rules having a higher priority. Like for the innermost mechanism of Definition 3, this requires to use negations of substitutions. Let us consider the PRS $\{f(g(x), y) \to a \blacktriangleright f(x, h(y)) \to b \blacktriangleright f(x, y) \to c\}$. The term $f(x, y)$ innermost narrows into a with the first rule and the *mgu* $\sigma_1 = (x = g(x'))$, into b

with the second rule, the *mgu* $\sigma_2 = (y = h(y'))$ and the constraint $x \neq g(x')$ (which is the negation of the *mgu* of $\sigma_2 f(x, y)$ with the *lhs* of the first rule), and finally into c with the third rule, the *mgu* σ_3 equal to Id and the constraint $x \neq g(x') \wedge y \neq h(y')$ (which is the negation of the *mgu* of $\sigma_3 f(x, y)$ with the first rule and of the *mgu* of $\sigma_3 f(x, y)$ with the second rule).

So, applying the rules one after the other on t, with the current *mgu* σ, we have to accumulate the negation of the *mgu*s of σt and the previous rules, without their constraint part. We have now to see how to manage together the constraints due to the innermost mechanism and those due to the priority mechanism.

If the narrowing substitutions are in their full form $\sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j}$ with a constraint part coming from the innermost mechanism of Definition 3, this constraint part is also ignored by the priority mechanism. Indeed, it is defined from σ_0, and has no meaning for the negation of σ_0. With the PRS $\{f(g(h(x))) \to a \blacktriangleright h(a) \to b \blacktriangleright f(g(x)) \to c\}$, the term $f(x)$ innermost narrows into *with* a the first rule and $\sigma_1 = (x = g(h(x')) \wedge x' \neq a)$, the second rule does not apply, and the third rule applies with $\sigma_3 = (x = g(x') \wedge x' \neq h(x''))$ (where $(x' = h(x'') \wedge x'' \neq a)$ is the narrowing substitution of $\sigma_3 f(x)$ with the first rule.

Also, if the constraint part of a substitution is due to the priority mechanism, the negation of this substitution by the innermost mechanism also only considers the *mgu* of the substitution. With the PRS $\{f(g(h(x, y)), z) \to a \blacktriangleright f(x, y) \to b$, $h(a, x) \to a \blacktriangleright h(x, b) \to b\}$, the term $f(x, y)$ innermost narrows into a with the first rule and $\sigma_1 = (x = g(h(x', y')) \wedge x' \neq a \wedge y' \neq b)$, because $h(x', y')$ narrows with the third rule and $\sigma = (x' = a)$, and with the fourth rule and $\sigma = (y' = b \wedge x' \neq a)$. Note that the term $f(x, y)$ also innermost narrows into b with the second rule and $\sigma_2 = (Id \wedge x \neq g(h(x', y')))$.

Definition 4 (Innermost priority narrowing). *Let $\mathcal{R}$ be a PRS. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ IP-*narrows *into $t' \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ at the non-variable position p of t, using the rule $l \to r \in \mathcal{R}$ with the constrained substitution $\sigma = \sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j} \bigwedge_{i\in[1..n]} \overline{\sigma_0^i}$, which is written $t \rightsquigarrow^{IP}_{p,l\to r,\sigma} t'$, iff $t' = \sigma_0(t[r]_p)$, where σ_0 is the* mgu *of $t|_p$ and l, $\sigma_j, j \in [1..k]$ are all* mgu*s of $\sigma_0 t|_{p'}$ and a* lhs *l' of a rule of $\mathcal{R}$, for all suffix position p' of p in t, and $\sigma_0^i, i \in [1..n]$ are the* mgu*s of $\sigma_0 t|_p$ with the* lhs *of the rules having a greater priority than $l \to r$.*

The following lifting lemma ensures that the previously defined narrowing relation simulates IP-rewriting on ground terms.

Lemma 1 (Priority Innermost Lifting Lemma). *Let $\mathcal{R}$ be a PRS. Let $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, α a ground substitution such that αs is IP-reducible at a non variable position p of s, and $\mathcal{Y} \subseteq \mathcal{X}$ a set of variables such that $\mathcal{V}ar(s) \cup \mathcal{D}om(\alpha) \subseteq \mathcal{Y}$. If $\alpha s \to^{IP}_{p,l\to r} t'$, then there exist a term $s' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and substitutions $\beta, \sigma = \sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j} \bigwedge_{i\in[1..n]} \overline{\sigma_0^i}$. such that:*

1. $s \rightsquigarrow^{IP}_{p,l\to r,\sigma} s'$,
2. $\beta s' = t'$,
3. $\beta\sigma_0 = \alpha[\mathcal{Y} \cup Var(l)]$
4. β satisfies $\bigwedge_{j\in[1..k]} \overline{\sigma_j} \bigwedge_{i\in[1..n]} \overline{\sigma_0^i}$

where σ_0 *is the* mgu *of* $s|_p$ *and* l *and* $\sigma_j, j \in [1..k]$ *are all* mgus *of* $\sigma_0 s|_{p'}$ *and a* lhs l' *of a rule of* $\mathcal{R}$*, for all suffix position* p' *of* p *in* s*, and* $\sigma_0^i, i \in [1..n]$ *are the* mgus *of* $\sigma_0 s|_p$ *with the* lhs *of the rules having a greater priority than* $l \to r$*.*

Accumulating Constraints. Abstraction constraints have to be combined with the narrowing substitutions to characterize the ground terms schematized by the current term t in the proof tree. Indeed, a narrowing step on the current term u with narrowing substitution σ represents a rewriting step for any ground instance of σu. So σ, considered as the narrowing constraint attached to the narrowing step, is added to the abstraction constraint, or in practice, propagated into it by applying its substitution part to the variables of the constraint.

Note that if σ does not satisfy the abstraction constraint, the narrowing step is meaningless: it does not correspond to any rewriting step of the considered ground instances.

This leads to the introduction of abstraction constraint formulas.

Definition 5. *An* abstraction constraint formula *(ACF in short) is a formula* $\bigwedge_i (t_i{\downarrow} = t'_i) \wedge \bigwedge_j (x_j = u_j)$*, where* $x_j \in \mathcal{X} \cup \mathcal{N}$*,* $t_i, t'_i, u_j, \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$*. It is* satisfiable *iff there is at least one instantiation* θ *such that* $\bigwedge_i (\theta t_i{\downarrow} = \theta t'_i) \wedge \bigwedge_j (\theta x_j = \theta u_j)$*;* θ *is then said to satisfy the ACF* A *and is called solution of* A*.*

An ACF A is attached to each term u in the proof trees; the ground substitutions solutions of A define the instances of the current term u, for which we are observing IP-termination. When A has no solution, the current node of the proof tree represents no ground term. Such nodes are then irrelevant for the proof. Detecting and suppressing them during a narrowing step allows us to control the narrowing mechanism, well known to easily diverge. So, we have the choice between generating only the relevant nodes of the proof tree, by testing the satisfiability of A at each step, or stopping the proof on a branch on an irrelevant node, by testing the unsatisfiability of A.

The satisfiability of A is in general undecidable, but it is often easy in practice to exhibit an instantiation satisfying it: most of the time, solutions built on constructor terms can be synthesized in an automatic way. Other automatable sufficient conditions, relying in particular on the characterization of normal forms [20], are also under study. The unsatisfiability of A is also undecidable in general, but here also, simple automatable sufficient conditions can be used. In Sect. 5, we present the procedure exactly simulating the rewriting trees i.e., dealing with the satisfiability of A. In Sect. 6, we give the alternative approach dealing with the unsatisfiability, present the sufficient conditions to test it, and show why it is particularly advantageous for IP-rewriting.

5 The IP-Termination Procedure

We are now ready to describe the inference rules defining our proof mechanism. They transform a set T of 3-tuples (U, A, C) where $U = \{t\}$ or $\emptyset$, t is the

current term whose ground instances have to be proved IP-terminating, A is an abstraction constraint formula, C is a conjunction of ordering constraints.

Before to give the inference rules, let us note that the inductive reasoning can be completed as follows. When the induction hypothesis cannot be applied to a term u, it may be possible to prove IP-termination of every ground instance of u in another way. Let $IPT(u)$ be a predicate that is true iff every ground instance of u is IP-terminating. In the first and third inference rules, we then associate the alternative predicate $IPT(u)$ to the condition $t > u$.

To establish $IPT(u)$, decidable sufficient conditions exist, applicable in practice, because the predicate is only considered for particular terms introduced along the proof, and not for any term. Simple cases often arise like non narrowable terms of $\mathcal{T}(\mathcal{F},\mathcal{N})$. The notion of usable rules [21,16], is also suitable for proving termination of u and can also be adapted to IP-rewriting.

The inference rules are given in Table 1. For a detailed explanation as well as considerations about the predicate IPT and the usable rules, see [19].

We generate the proof trees of $\mathcal{R}$ by applying, for each symbol $g \in \mathcal{D}$, the inference rules on the initial 3-tuple $(\{t_{ref} = g(x_1,\ldots,x_m)\}, \top, \top)$ (if g is a constant, then $t_{ref} = g$), with a specific strategy $S{-}RULES$:

$$repeat^*(try{-}skip(\textbf{Abstract}), try{-}stop(\textbf{Narrow}), try{-}skip(\textbf{Stop})).$$

"$repeat^*(T_1,\ldots,T_n)$" repeats the control strategies $T_1,\ldots,T_n$ until none of them is applicable anymore, $try{-}skip(T)$ expresses that T is tried and skipped when it cannot be applied, $try{-}stop(T)$ stops $S{-}RULES$ if T cannot be applied.

The process may not terminate if there is an infinite number of applications of **Abstract** and **Narrow** on the same branch of a proof tree. It may stop on

Table 1. Inference rules for IP-termination

Abstract:
$$\frac{\{t\},\ A,\ C}{\{u\},\ A \wedge \bigwedge_{j\in\{i_1,\ldots,i_p\}} t|_j\!\downarrow = X_j,\ C \wedge \bigwedge_{j\in\{i_1,\ldots,i_p\}} H_C(t|_j)}$$

where t is abstracted into u at positions $i_1,\ldots,i_p \neq \epsilon$
if $C \wedge H_C(t|_{i_1}) \ldots \wedge H_C(t|_{i_p})$ is satisfiable

Narrow:
$$\frac{\{t\},\ A,\ C}{\{v_i\},\ A\sigma,\ C} \quad \text{if } t \rightsquigarrow^{IP}_{\sigma} v_i \text{ and } A\sigma \text{ is satisfiable}$$

Stop:
$$\frac{\{t\},\ A,\ C}{\emptyset,\ A \wedge H_A(t),\ C \wedge H_C(t)} \quad \text{if } (C \wedge H_C(t)) \text{ is satisfiable.}$$

$$H_A(t) = \begin{cases} \top & \text{if } t \text{ is in } \mathcal{T}(\mathcal{F},\mathcal{N}) \\ & \text{and is not narrowable} \\ t\!\downarrow = X & \text{otherwise.} \end{cases} \qquad H_C(t) = \begin{cases} \top & \text{if } IPT(t) \\ t_{ref} > t & \text{otherwise.} \end{cases}$$

Abstract (resp. **Narrow**) when the ordering (resp. abstraction) constraints cannot be proved satisfiable. Nothing can be said in these cases about IP-termination. The good case is when all branches of the proof trees end with an application of **Stop**: then IP-termination is established.

A finite proof tree is said to be *successful* if its leaves are states of the form $(\emptyset, A, C)$. We write $SUCCESS(g, \succ)$ if the application of $S{-}RULES$ on $(\{g(x_1, \ldots, x_m)\}, \top, \top)$ gives a successful proof tree, whose sets C of ordering constraints are satisfied by the same ordering $\succ$.

Theorem 1. *Let $\mathcal{R}$ be a priority term rewrite system on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ having at least one constructor constant. Every term of $\mathcal{T}(\mathcal{F})$ is IP-terminating iff there is a noetherian ordering $\succ$ such that for each $g \in \mathcal{D}$, we have $SUCCESS(g, \succ)$.*

Examples using the rules of Table 1 can be found in [19], as well as the proofs of Lemma 1 and Theorem 1.

6 Abstraction Constraints for Priority Rewriting

We present in this section an alternative approach to our procedure, dealing with the unsatisfiability of A instead of the satisfiability, and show in comparison why it is suitable for IP-termination. As explained in Sect. 4, instead of testing whether each node generated in the proof tree is relevant i.e., whether A is satisfiable, we test whether we have generated irrelevant nodes, and stop the proof process on the irrelevant nodes.

For this, we just have to suppress the satisfiability test of $A \wedge \sigma$ in the condition of **Narrow**, and to add the condition "A is unsatisfiable" as an alternative condition of **Stop**. **Narrow** is then applied with $try{-}skip$ instead of $try{-}stop$.

In [16], we give automatable sufficient conditions for the unsatifiability of an abstraction constraint $t{\downarrow} = t'$, often applicable in practice. As these conditions only work on the equalities of A, dealing with the unsatisfiability of A instead of the satisfiability is of particular interest when A involves many negations of substitutions. Testing the satisfiability instead requires to verify that the solutions of the equational part of A verify its disequality part.

Testing the unsatisfiability is precisely advantageous for a succession of priority rules involving more than two or three rules, since narrowing with the nth rule requires to accumulate the negation of $n-1$ narrowing substitutions.

Moreover, since the unsatisfiability test is an alternative condition of **Stop**, dealing with the unsatisfiability of A instead of the satisfiability is obviously interesting when **Stop** applies with the first condition ($(C \wedge H_C(t))$ is satisfiable). Analyzing A can then be completely avoided. It is immediate when rules have constant right-hand sides: **Narrow** then generates constant terms, for which the predicate IPT trivially holds.

As said in the introduction, rewriting-based specifications with priorities on rules have recently been used to specify security policies, with a concern of

verification of consistency, termination and completeness. They often have the two characteristics enlightened above. The example we give below has been proposed in [9], for a conference management system described in [22]. Its termination, due to priority arguments, could not be formally proved until now.

Example 2. If we do not consider priorities, the following rewrite system is divergent, because of the eighth rule. Let us prove that it is IP-terminating.

$$
\left|\begin{array}{l}
aut(q(author(x), SP, pap(x,z)), SUBMIS, u) \quad \to PERMIT \\
aut(q(author(x), SP, pap(x,z)), v, u) \quad \to DENY \\
aut(q(author(x), RSC, pap(x,z)), v, u) \quad \to DENY \\
aut(q(rev(x), w, p), v, conf(x,p)) \quad \to DENY \\
aut(q(rev(x), SR, pap(y,z)), REV, ass(x, pap(y,z))) \quad \to PERMIT \\
aut(q(rev(x), SR, pap(y,z)), v, ass(x, pap(y,z))) \quad \to DENY \\
aut(q(rev(x), RSC, pap(y,z)), MEE, ass(x, pap(y,z))) \quad \to PERMIT \\
aut(q(rev(x), w, pap(x,z)), v, u) \to aut(q(rev(x), w, pap(x,z)), v, conf(x, pap(x,z))) \\
aut(x,y,z) \to NAPPLIC\ .
\end{array}\right.
$$

We apply the strategy $S{-}RULES$ on the initial pattern $t_{ref} = aut(x,y,z)$. The proof tree is given in Fig. 1. We first have an **Abstract** step. Then, a **Narrow** step gives 9 branches, following the 9 above rules. With the constrained narrowing substitutions $\sigma_1, (\sigma_2 \wedge \overline{\sigma_2^1}), \ldots (\sigma_7 \wedge \overline{\sigma_7^1} \wedge \ldots \wedge \overline{\sigma_7^6})$ (where σ_i^j is the *mgu* of $\sigma_i aut(X,Y,Z)$ with the jth rule), the first seven ones give respectively the states $PERMIT$, $DENY$, $DENY$, $DENY$, $PERMIT$, $DENY$, $PERMIT$, on which **Stop** then applies. Indeed, we have $IPT(PERMIT)$ and $IPT(\ DENY)$ since $PERMIT$ and $DENY$ are constructor constants. The ninth branch gives the state $NAPPLIC$, on which **Stop** applies too.

$$
\begin{array}{c}
t_{ref} = aut(x,y,z) \\
A = \top, C = \top \\
\Big\downarrow \textbf{Abstract} \\
aut(X,Y,Z) \\
A = (x{\downarrow} = X, y{\downarrow} = Y, z{\downarrow} = Z), C = (aut(x,y,z) > x, y, z) \\
\Big\downarrow \underset{Rule\ 8}{\textbf{Narrow}}\ \sigma_8 \wedge \overline{\sigma_8^1} \wedge \ldots \overline{\sigma_8^7} \\
aut(q(rev(X'), w, pap(X', Z')), Y,\ conf(X', pap(X', Z'))) \\
A = (x{\downarrow} = X, y{\downarrow} = Y, z{\downarrow} = Z \\
\wedge X = q(rev(X'), w, pap(X', Z'))\ \wedge \overline{\sigma_1} \ldots \overline{\sigma_7}) \\
\Big\downarrow \underset{Rule\ 4}{\textbf{Narrow}}\ \sigma_{10} = Id \\
DENY \\
\Big\downarrow \textbf{Stop} \\
\emptyset
\end{array}
$$

Fig. 1. Proof tree for symbol *aut*

The interesting branch is the eighth one, giving the state $aut(q(rev(X'), w, pap(X', Z')), Y, conf(X', pap(X', Z')))$ with the substitution $\sigma_8 = (X = q(rev(X'), w, pap(X', Z')))$ constrained by $\overline{\sigma_8^1} \wedge \ldots \overline{\sigma_8^7}$. To lighten the figure, we only specify this branch in the proof tree.

From this last state, we still apply **Narrow**, with three narrowing possibilities: one, with the fourth rule and the substitution $\sigma_9 = Id$, gives the state $DENY$, on which **Stop** then applies, because we have $IPT(DENY)$. The two other ones are not valid: using the eighth and the ninth rules, we also have the narrowing substitution Id , which becomes empty once constrained by $\overline{\sigma_9}$.

Applying the inference rules dealing with the satisfiability of A would have required to perform the satisfiability test for the nine branches of the first **Narrow** step, which is avoided here.

As one can see, the rule **Stop** applies on all branches of the proof tree thanks to the predicate IPT. So, on this example, we do not even need to consider A. To satisfy the ordering constraints, any simplification ordering holds. So this example can be treated in a completely automatic way.

7 Conclusion

In this paper, we have proposed an inductive method for proving termination of the decidable innermost priority rewriting relation of C.K. Mohan [11]. This work is an extension to priority rewriting of an inductive approach given in [16] for proving innermost termination of rewriting.

In our termination proof technique, the priority mechanism localizes in the specific narrowing relation used to model the rewriting relation on ground terms. Moreover, it can be expressed through negations of substitutions, then introducing constraints similar to those already required to model ground innermost rewriting. We then have generalized the innermost narrowing relation introduced in [16], to model the IP-rewriting relation on ground terms and have given a lifting lemma ensuring the correctness of this modelization.

Constraints are crucial in our approach: ordering constraints guarantee the applicability of the induction principle, abstraction constraints define the ground terms considered at each step of the proof, and help to contain the narrowing mechanism. When the treatment of the constraints is automatable – sufficient conditions for ordering constraints as well as for abstraction constraints can be given for this – the proof procedure is completely automatable. Considering unsatisfiability of abstraction constraints instead of satisfiability is, in general, particularly suitable for priority rewriting and more precisely for rule-based security policies.

As termination of the original priority rewriting relation of [2] guarantees a semantics for this relation, one can think that IP-termination guarantees a semantics for the IP-rewriting relation. This has to be investigated. We also plan to generalize our technique to the termination proof of other priority rewriting relations, in particular for specifying security policies.

References

1. Baeten, J.C.M., Bergstra, J.A., Klop, J.W.: Term rewriting systems with priorities. In: Lescanne, P. (ed.) RTA 1987. LNCS, vol. 256, pp. 83–94. Springer, Heidelberg (1987)
2. Baeten, J.C.M., Bergstra, J.A., Klop, J.W., Weijland, W.P.: Term-rewriting systems with rule priorities. Theoretical Computer Science 67(2-3), 283–301 (1989)
3. van den Brand, M.G., Klint, P., Verhoef, C.: Term Rewriting for Sale. In: WRLA 1998. ENTCS, vol. 15, pp. 139–161. Elsevier, Amsterdam (1998)
4. Marti-Oliet, N., Meseguer, J., Verdejo, A.: Towards a strategy language for Maude. In: WRLA 2004. ENTCS, vol. 117, pp. 417–441. Elsevier, Amsterdam (2004)
5. Plasmeijer, R., van Eekelen, M.: Functional Programming and Parallel Graph Rewriting. Addison-Wesley, Reading (1993)
6. Augustsson, L.: A compiler for lazy ML. In: LFP 1984, pp. 218–227. ACM, New York (1984)
7. Home of Clean, http://clean.cs.ru.nl/index.html
8. Wiki homepage of Haskell, http://www.haskell.org/haskellwiki/Haskell
9. de Oliveira, A.S.: Réécriture et Modularité pour les Politiques de Sécurité. PhD thesis, Univesité Henri Poincaré, Nancy, France (2008)
10. de Oliveira, A.S., Wang, E.K., Kirchner, C., Kirchner, H.: Weaving rewrite-based access control policies. In: FMSE 2007, pp. 71–80. ACM, New York (2007)
11. Mohan, C.K.: Priority rewriting: Semantics, confluence, and conditionals. In: Dershowitz, N. (ed.) RTA 1989. LNCS, vol. 355, pp. 278–291. Springer, Heidelberg (1989)
12. van de Pol, J.: Operational semantics of rewriting with priorities. Theoretical Computer Science 200(1-2), 289–312 (1998)
13. Sakai, M., Toyama, Y.: Semantics and strong sequentiality of priority term rewriting systems. Theoretical Computer Science 208(1–2), 87–110 (1998)
14. van de Pol, J., Zantema, H.: Generalized innermost rewriting. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 2–16. Springer, Heidelberg (2005)
15. Moreau, P., Ringeissen, C., Vittek, M.: A Pattern Matching Compiler for Multiple Target Languages. In: Hedin, G. (ed.) CC 2003. LNCS, vol. 2622, pp. 61–76. Springer, Heidelberg (2003)
16. Gnaedig, I., Kirchner, H.: Termination of rewriting under strategies. ACM Transactions on Computational Logic (to appear, 2008), tocl.acm.org/accepted/315gnaedig.ps
17. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press, Cambridge (1998)
18. Terese: Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)
19. Gnaedig, I.: Termination of Priority Rewriting - Extended Version. HAL-INRIA Open Archive Number inria-00349031 (2008)
20. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007) (release October 12th, 2007), http://www.grappa.univ-lille3.fr/tata
21. Arts, T., Giesl, J.: Proving innermost normalisation automatically. In: Comon, H. (ed.) RTA 1997. LNCS, vol. 1232, pp. 157–171. Springer, Heidelberg (1997)
22. Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Specifying and reasoning about dynamic access-control policies. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 632–646. Springer, Heidelberg (2006)

State Complexity of Combined Operations for Prefix-Free Regular Languages

Yo-Sub Han[1], Kai Salomaa[2], and Sheng Yu[3]

[1] Intelligence and Interaction Research Center, KIST
P.O.BOX 131, Cheongryang, Seoul, Korea
emmous@kist.re.kr
[2] School of Computing, Queen's University
Kingston, Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca
[3] Department of Computer Science, University of Western Ontario
London, Ontario N6A 5B7, Canada
syu@csd.uwo.ca

Abstract. We investigate the state complexity of combined operations for prefix-free regular languages. Prefix-free minimal deterministic finite-state automata have a unique structural property that plays an important role to obtain the precise state complexity of basic operations. Based on the same property, we establish the precise state complexity of four combined operations: star-of-union, star-of-intersection, star-of-reversal and star-of-catenation.

1 Introduction

Regular languages are widely used in many applications such as text searching, speech processing or software engineering [1,2,3]. Given a regular language L, researchers often use the number of states in the minimal DFA for L to represent the complexity of L. Based on this notation, we, then, define the state complexity of an operation for regular languages to be the number of states that are necessary and sufficient in the worst-case for the minimal DFA that accepts the language obtained from the operation [4]. The state complexity of an operation is calculated based on the the structural properties of given regular languages and the function of a given operation. Recently, due to large amount of memory and fast CPUs, many applications using regular languages require huge size of finite-state automata (FAs). This makes the estimated upper bound of the state complexity useful in practice since it is directly related to the efficient resource management in applications. Moreover, it is a challenging quest to verify whether or not an estimated upper bound can be reached.

Yu [5] gave a comprehensive survey of the state complexity of regular languages. Salomaa et al. [6] studied classes of languages for which the reversal operation reaches the exponential upper bound. As special cases of the state complexity, researchers examined the state complexity of finite languages [7,8],

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 398–409, 2009.

the state complexity of unary language operations [9] and the nondeterministic descriptional complexity of regular languages [10]. For regular language codes, Han et al. [11] studied the state complexity of prefix-free regular languages. They tackled the problem based on the structural property of prefix-free DFAs: A prefix-free DFA must be non-exiting assuming all states are useful [11]. Similarly, based on suffix-freeness, Han and Salomaa [12] looked at the state complexity of suffix-free regular languages. There are several other results with respect to the state complexity of different operations [13,14,15,16,17,18].

While people mainly looked at the state complexity of single operations (union, intersection, catenation and so on), Yu and his co-authors [19,20,21] recently started investigating the state complexity of combined operations (star-of-union, star-of-intersection and so on). They showed that the state complexity of a combined operation is usually not equal to the composition of the state complexities of the participating individual operations. On the other hand, they also observed that in a few cases, the state complexity of a combined operation is very close to the composition of the state complexities. This leads us to study the state complexity of combined operations and examine the cases that give a similar state complexity to the composition of the state complexities of single operations.

We choose prefix-free regular languages for the state complexity of combined operations. Note that state complexity of prefix-free regular languages is very different from the state complexity of regular languages because prefix-freeness gives a unique structural property in a prefix-free minimal DFA [11,22]. Moreover, prefix-free languages are used in many coding theory applications (Huffman coding is an example), and for this reason results on state complexity of combined operations for prefix-free regular languages may be useful. Furthermore, determining the state complexity of combined operations on fundamental subfamilies of the regular languages can provide valuable insights on connections between restrictions placed on language definitions and descriptional complexity.

In Section 2, we define some basic notions. Then, we present the state complexities of four combined operations in Section 3. We compare the state complexity of basic operations and the state complexity of combined operations for prefix-free regular languages, and conclude the paper in Section 4.

2 Preliminaries

Let Σ denote a finite alphabet of characters and Σ^* denote the set of all strings over Σ. The size $|\Sigma|$ of Σ is the number of characters in Σ. A language over Σ is any subset of Σ^*. The symbol $\emptyset$ denotes the empty language and the symbol λ denotes the null string. For strings x, y and z, we say that x is a *prefix* of y if $y = xz$. We define a (regular) language L to be prefix-free if for any two distinct strings x and y in L, x is not a prefix of y. Given a string x in a set X of strings, let x^R be the reversal of x, in which case $X^R = \{x^R \mid x \in X\}$.

An FA A is specified by a tuple $(Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $s \in Q$ is the

start state and $F \subseteq Q$ is a set of final states. If F consists of a single state f, then we use f instead of $\{f\}$ for simplicity. Given a DFA A, we assume that A is complete; namely, each state has $|\Sigma|$ out-transitions and, therefore, A may have a sink state. We assume that A has a unique sink state since all sink states are equivalent and can be merged into a single state. Let $|Q|$ be the number of states in Q. The size $|A|$ of A is $|Q|$. For a transition $\delta(p, a) = q$ in A, we say that p has an *out-transition* and q has an *in-transition*. Furthermore, p is a *source state* of q and q is a *target state* of p. We say that A is *non-returning* if the start state of A does not have any in-transitions and A is *non-exiting* if all out-transitions of any final state in A go to the sink state.

A string x over Σ is accepted by A if there is a labeled path from s to a final state such that this path reads x. We call this path an *accepting path*. Then, the language $L(A)$ of A is the set of all strings spelled out by accepting paths in A. We say that a state of A is *useful* if it appears in an accepting path in A; otherwise, it is *useless*. Unless otherwise mentioned, in the following we assume that all states are useful.

A regular expression E is prefix-free if $L(E)$ is prefix-free and an FA A is prefix-free if $L(A)$ is prefix-free. Moreover, if $L(A)$ is prefix-free, then A must be non-exiting. We recall that an arbitrary minimal DFA recognizing a prefix-free language has exactly one final state and all of its out-transitions go to the sink state [11].

For complete background knowledge in automata theory, the reader may refer to the textbook [23].

3 State Complexity of Combined Operations

We consider four combined operations of prefix-free regular languages: star-of-union, star-of-reversal, star-of-catenation and star-of-intersection. For each operation, we compare the state complexity of a combined operation and the composition of the state complexities of two individual operations. In the following section, let $\mathcal{SC}(L)$ denote the state complexity of L.

3.1 Star of Union

First we give an upper bound construction for the state complexity of star-of-union of two prefix-free languages. Let $A_i = (Q_i, \Sigma, \delta_i, q_{0,i}, f_i)$, $|Q_i| = m_i$, $i = 1, 2$ be arbitrary minimal DFAs recognizing prefix-free languages. Here $f_i \in Q_i$ is the unique final state and we can assume that $f_i \neq q_{i,0}$. (If the final state is the start state, A_i must recognize $\{\lambda\}$.) We denote by $d_i \in Q_i$ the sink state of A_i, $Q'_i = Q_i \setminus \{f_i, d_i\}$ is the set of states of A_i excluding the final state and the sink state. Without loss of generality we assume that $Q_1 \cap Q_2 = \emptyset$.

We construct a DFA

$$A = (Q, \Sigma, \delta, \{q_0, q_{0,1}, q_{0,2}\}, F) \tag{1}$$

for the language $(L(A_1) \cup L(A_2))^*$. We choose Q to be the collection of subsets of $P \subseteq \{q_0\} \cup Q'_1 \cup Q'_2$ such that

$$\text{if } q_0 \in P, \text{ then } q_{0,1}, q_{0,2} \in P. \tag{2}$$

The set of final states F consists of all elements of Q that contain q_0 and the transition function δ is defined as follows. Let $P = X \cup P_1 \cup P_2$, where $P_i \subseteq Q'_i$, $i = 1, 2$, X is $\{q_0\}$ or $\emptyset$, and $c \in \Sigma$. Then we define:

$$\delta(P, c) = \begin{cases} \delta_1(P_1, c) \cup \delta_2(P_2, c), & \text{if } f_1 \notin \delta_1(P_1, c) \text{ and } f_2 \notin \delta_2(P_2, c), \\ \delta_1(P_1, c) \cup \delta_2(P_2, c) \cup \{q_{0,1}, q_{0,2}, q_0\}, & \text{otherwise.} \end{cases}$$

The transitions defined above clearly preserve the property (2), that is, for any $P \in Q$, $\delta(P, c) \in Q$.

The construction of A can be viewed as constructing an NFA for the star-of-union of the original languages, and performing a subset construction on the NFA. In the subset construction, we can merge the final states of the original DFAs. The construction is illustrated in Fig. 1.

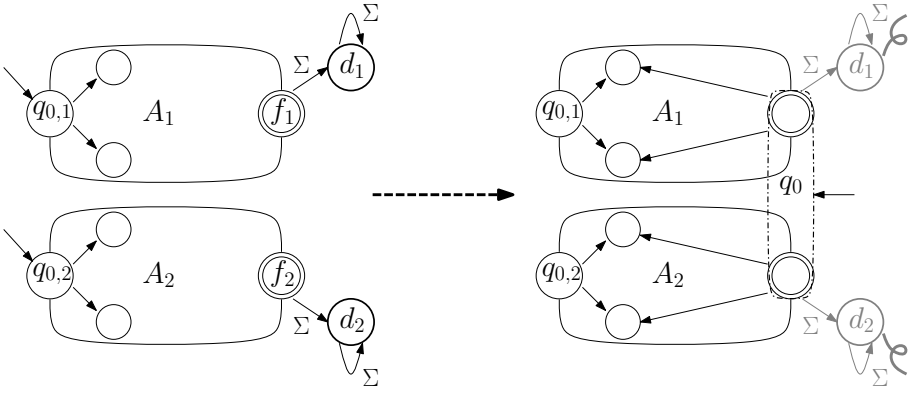


Fig. 1. Construction of an NFA for the star-of-union. Note that the merged state q_0, which is a final state as well, is the start state.

The DFA A uses the symbol q_0 to represent the merging of the two final states f_1 and f_2 of the original DFAs. The start state of A is $\{q_0, q_{0,1}, q_{0,2}\}$ which means that the computation begins by simulating both the computation of A_1 and the computation of A_2. Whenever one of the simulated computations enters the final state, the computation of A adds both $q_{0,1}$ and $q_{0,2}$ to the current set of states and begins new computations simulating A_1 and A_2. Namely, q_0 in the current set indicates that the previously simulated computation step is accepted either in A_1 or in A_2. Note that the presence of $q_{0,1}$ or $q_{0,2}$ in the current state of A is not sufficient to guarantee this property. The choice of the final states guarantees that A recognizes exactly the language $(L(A_1) \cup L(A_2))^*$.

Assuming, $q_{0,i}$, f_i, and d_i are all distinct, for $i = 1, 2$, the set Q of states defined by (2) contains $2^{m_1+m_2-4}$ subsets of $Q'_1 \cup Q'_2$ that do not contain the

merged final state q_0. Additionally, Q contains $2^{m_1+m_2-6}$ subsets of the form $\{q_0, q_{0,1}, q_{0,2}\} \cup P$ where $P \subseteq (Q_1 \setminus \{q_{0,1}, f_1, s_1\}) \cup (Q_2 \setminus \{q_{0,2}, f_2, s_2\})$. Note that if $q_{0,i}$, f_i and d_i, $i \in \{1, 2\}$, are not distinct, then A_i recognizes one of the trivial languages $\{\lambda\}$ or $\emptyset$.

Now we obtain the following upper bound for the state complexity of star-of-union of prefix-free regular languages from the construction.

Lemma 1. *Let L_i be a prefix-free regular language with $\mathcal{SC}(L_i) = m_i$, $m_i \geq 3$, $i = 1, 2$. Then*

$$\mathcal{SC}((L_1 \cup L_2)^*) \leq 5 \cdot 2^{m_1+m_2-6}.$$

In the following, we give a worst-case construction that reaches the upper bound of Lemma 1. Let $\Sigma = \{a, b, c, d, e\}$ and $m, n \geq 3$. We define

$$A_1 = (R, \Sigma, \delta_1, r_0, \{r_{m-2}\}), \tag{3}$$

where $R = \{r_0, \ldots, r_{m-1}\}$, and the transitions of δ_1 are defined as:

- $\delta_1(r_i, a) = r_{i+1}$, $i = 0, \ldots, m-4$, $\delta_1(r_{m-3}, a) = r_0$.
- $\delta_1(r_i, b) = \delta_1(r_i, d) = r_i$, $i = 0, \ldots, m-3$.
- $\delta_1(r_0, c) = r_{m-2}$, $\delta_1(r_i, c) = r_i$, $i = 1, \ldots, m-3$.

The state r_{m-1} is the sink state of A_1 and above all undefined transitions go to r_{m-1}. In particular, note that all transitions of A_1 on input symbol e go to the sink state. The language $L(A_1)$ is prefix-free since all out-transitions from the final state r_{m-2} go to the sink state.

We define the second DFA as

$$A_2 = (S, \Sigma, \delta_2, s_0, \{s_{n-2}\}), \tag{4}$$

where $S = \{s_0, \ldots, s_{n-1}\}$, and δ_2 is defined by setting:

- $\delta_2(s_i, b) = s_{i+1}$, $i = 0, \ldots, n-4$, $\delta_2(s_{n-3}, b) = s_0$.
- $\delta_2(s_i, a) = \delta_2(s_i, e) = s_i$, $i = 0, \ldots, n-3$.
- $\delta_2(s_0, c) = s_{n-2}$, $\delta_2(s_i, c) = s_i$, $i = 1, \ldots, n-3$.

Again, s_{n-1} is the sink state of A_2 and all above undefined transitions go to the sink state. In particular, any state of A_2 transitions with input symbol d to the sink state.

The DFAs A_1 and A_2 are depicted in Fig. 2.

We show that the DFAs in Fig. 2 reach the upper bound of Lemma 1 when $m, n \geq 3$. Note that any complete DFA recognizing a prefix-free language that is not $\{\lambda\}$ or $\emptyset$ has to have at least three states.

Lemma 2. *Let $m, n \geq 3$ and let A_1 and A_2 be DFAs as defined in (3) and (4), respectively.*

We claim that $\mathcal{SC}((L(A_1) \cup L(A_2))^*)$ is $5 \cdot 2^{m+n-6}$.

By Lemmas 1 and 2 we get a precise bound for the state complexity of star-of-union of prefix-free regular languages.

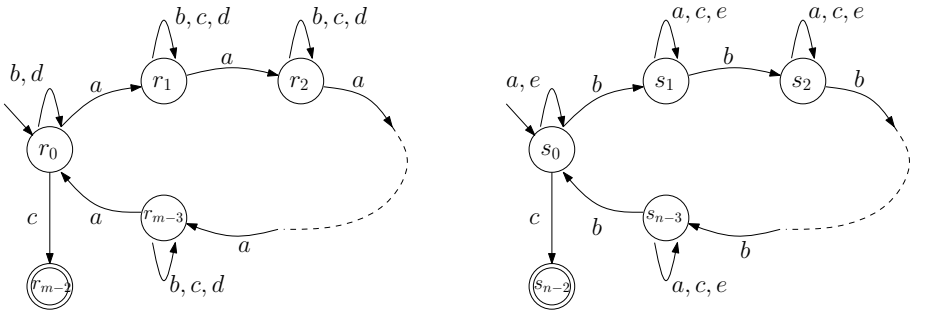


Fig. 2. The DFAs A_1 and A_2. The figure does not show the sink states r_{m-1} and s_{n-1} and their in-transitions.

Theorem 1. *The worst-case state complexity of the star-of-union of an m_1-state and an m_2-state, $m_1, m_2 \geq 3$, prefix-free regular languages is precisely $5 \cdot 2^{m_1+m_2-6}$, where $|\Sigma| \geq 5$.*

Theorem 1 implies that the upper bound can be reached for all values $m_1, m_2 \geq 3$. If, for example, $m_2 = 2$, the state complexity is m_1. The result follows from the state complexity of star for prefix-free languages [11] because with $m_2 = 2$, the worst-case example corresponds to the case where $L(A_2) = \{\lambda\}$ and $(L(A_1) \cup L(A_2))^* = L(A_1)^*$.

Note that the state complexity of the union of two prefix-free regular languages is $m_1 m_2 - 2$ [11] and the state complexity of the star of an m-state regular language is $3 \cdot 2^{m-2}$ [4]. Thus, the composition of two complexities is $3 \cdot 2^{m_1 m_2 - 4}$. Therefore, the state complexity of the star-of-union is much lower (one has a linear exponent and the other has a quadratic exponent) than the composition of two complexities.

The lower bound construction of Lemma 2 uses an alphabet of size five. It remains an open question whether the worst-case bound can be reached by prefix-free regular languages over alphabets of size 2, 3 or 4.

3.2 Star of Reversal

Let A be a minimal DFA for a regular language and $|A| = m$. Since the state complexity of $L(A)$ is m, the reversal $L(A)^R$ of $L(A)$ can be accepted by an NFA A^R with m states, where A^R can have multiple start states. We, then, use the subset construction on A^R and the resulting DFA can have at most 2^m states. Thus, the upper bound for the reversal of regular languages is 2^m. Leiss [24] demonstrated that some classes of languages can reach the upper bound. Later, Salomaa et al. [6] showed the conditions for such regular languages and obtained the following result.

Proposition 1 (Salomaa et al. [6]). *Let Σ be an alphabet with at least 2 characters. There exists a minimal DFA A that has a maximal blow-up, 2^m, in the transition to its reversal $L(A)^R$, where the transition function of A is functionally complete, $L(A) \neq \emptyset, \Sigma^*$ and $m = |A|$.*

Given a minimal prefix-free DFA $A = (Q, \Sigma, \delta, s, f)$, we can obtain an FA for $L((A)^R)^*$ as follows: We first flip all transition directions in A such that f is the start state and s is the final state. Then, the resulting FA $A^R = (Q, \Sigma, \delta^R, f, s)$ is the reversal of A; $L(A)^R = L(A^R)$. The sink state d of A is now unreachable from f in A^R and, thus, we remove it. Next, we add all out-transitions of f to s and make f to be final in A^R. Note that we keep original out-transition of s in A^R as well. Therefore, we have an FA $A' = (Q \setminus \{d\}, \Sigma, \delta', f, \{s, f\})$, where

$$\delta'(q,a) = \begin{cases} \delta^R(q,a) \text{ if } q \neq s, \\ \delta^R(q,a) \cup \delta^R(f,a) \text{ otherwise.} \end{cases}$$

It is clear from the construction that $L(A') = L((A)^R)^*$. Note that A' is not necessarily deterministic since we have flipped transition directions. Fig. 3 illustrates this construction.

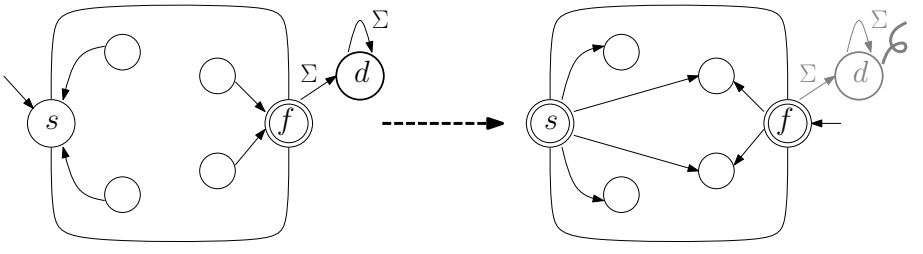


Fig. 3. Construction of an NFA for the star-of-reversal

Lemma 3. *Let L be a prefix-free regular language with $\mathcal{SC}(L) = m$. Then,*

$$\mathcal{SC}((L^R)^*) \leq 2^{m-2} + 1.$$

Next, we demonstrate that $2^{m-2} + 1$ states are necessary for the star-of-reversal of an m-state prefix-free regular language L. Given a (regular) language L over Σ, $L\#$ is prefix-free if the character $\#$ is not in Σ. Our approach is an extension of the method used by Han et al. [11] for computing the reversal of prefix-free minimal DFAs.

Let $A = (Q, \Sigma, \delta, s, F)$ be a minimal DFA in Proposition 1 over Σ, which is not prefix-free in general. We construct a prefix-free minimal DFA $A_\# = (Q', \Sigma \cup \{\#\}, \delta', s, f')$ that requires m states as follows:

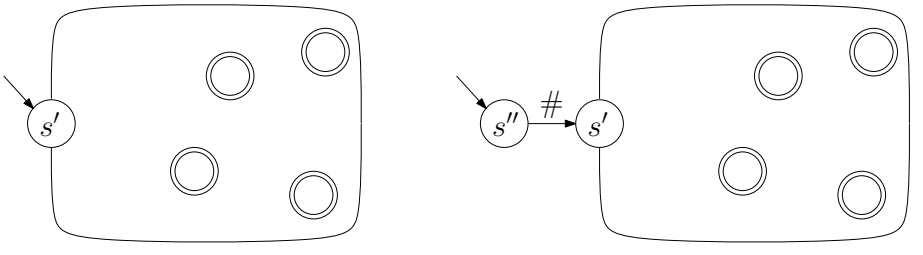
$Q' = Q \cup \{d, f'\}$,
for $q \in Q$ and $a \in \Sigma \cup \{\#\}$,

$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{if } q \in Q \text{ and } a \neq \#, \\ f' & \text{if } q \in F \text{ and } a = \#, \\ d & \text{otherwise.} \end{cases}$$

Namely, we introduce a new final state f' and connect all states in F of A to f' with label $\#$. We also introduce a sink state d. Since A is functionally

complete, that is, the transition function of A consists of all mappings from Q to Q, A cannot have a sink state, and consequently, d is not equivalent with any of the states of A. Note that by the construction, $A_{\#}$ is deterministic and minimal. Furthermore, $L(A_{\#})$ is prefix-free. Thus, if A has $m-2$ states, then $A_{\#}$ has m states.

Proposition 2 (Han et al. [11]). *Given a prefix-free minimal DFA $A_{\#}$ as constructed above, $2^{m-2}+1$ states are necessary for the minimal DFA of $L(A_{\#})^R$, where $m = |A_{\#}|$ and $\# \notin \Sigma$.*



minimal DFA for $L(A^R)$ minimal DFA for $L(A_{\#})^R$

Fig. 4. Construction of the minimal DFA for $L(A_{\#})^R$ from the minimal DFA for $L(A^R)$. Note that the left DFA is defined over Σ and the right DFA is defined over $\Sigma \cup \{\#\}$.

The brief sketch of Proposition 2 is that we flip the transition directions of A and apply the subset construction on A^R. Let s' be the start state of the resulting DFA. Then, we introduce a new start state s'' and connect s'' to s' with label #. Then, the new FA is the minimal DFA for $L(A_{\#})^R$ and the number of states is $2^{m-2}+1$ if $|A| = m-2$. See Fig. 4.

We are ready to compute the star-of-reversal for $A_{\#}$. From the minimal DFA $M = (Q, \Sigma, \delta, s'', F)$ for $L(A_{\#})^R$ as depicted in Fig. 4, we add #-transitions from all final states in F to s', which is the only target state of s'' except for the sink state d. We also make s'' to be a final state. Let $M_{\#}$ be the resulting FA. It is easy to verify that $L(M_{\#}) = (L(A_{\#})^R)^*$. We only need to show that $M_{\#}$ is a minimal DFA. Since $\delta(f, \#) = d$ for $f \in F$ in M, $M_{\#}$ is deterministic.

We show that all states of $M_{\#}$ are pairwise inequivalent. First consider two nonfinal states of $M_{\#}$, r_1 and r_2. Now since r_1 and r_2 are inequivalent as states of M, there exists $w \in \Sigma^*$ such that $\delta(r_1, w) \in F$ and $\delta(r_2, w) \notin F$. Note that since r_1 and r_2 are distinct from s'', we can assume that w does not contain occurrences of #, and the same string w distinguishes r_1 and r_2 as states of $M_{\#}$.

Next, consider two final states of $M_{\#}$. If they are both also final states in M, then we can use the same argument as above. It remains to show that s'' is not equivalent with any other final state r of $M_{\#}$. We note that since the original DFA A is functionally complete, r must have an out-transition on a symbol of Σ to some state of Q. On the other hand, the only out-transition from s'' (either in M or $M_{\#}$) whose target state is not the sink state d is on the input symbol #. This means that s'' and r are inequivalent. Therefore, $M_{\#}$ is minimal.

Note that, from M, we have constructed the minimal DFA for the star-of-reversal of $A_{\#}$ without adding new states. This implies that we get a precise bound for the state complexity of star-or-reversal of prefix-free regular languages.

Theorem 2. *The worst-case state complexity of the star-of-reversal of an m-state prefix-free regular language is precisely $2^{m-2}+1$, where $|\Sigma| \geq 3$.*

Note that the state complexity of the reversal of an m-state prefix-free regular language is $2^{m-2}+1$ [11] and the state complexity of the star of an m-state suffix-free regular language is $2^{m-2}+1$ [12]. Thus, the composition of two complexities is $2^{2^{m-2}+1-2}+1$. Therefore, the state complexity of the star-of-reversal is much lower than the composition of two complexities.

3.3 Star of Catenation and Intersection

We examine two operations, star-of-catenation and star-of intersection, based on the following observations:

1. Prefix-freeness is preserved by catenation and intersection.
2. Given an m-state prefix-free DFA A, $\mathcal{SC}(L(A)^*) = m$ [11].

Theorem 3. *The worst-case state complexity of the star-of-catenation of an m_1-state and an m_2-state, $m_1, m_2 \geq 3$, prefix-free regular languages is precisely $m_1 + m_2 - 2$.*

Theorem 3 shows that the state complexity of the star-of-catenation is equal to the state complexity calculated by the composition of state complexities of star and catenation. This is due to the fact that prefix-freeness is closed under catenation. Since prefix-freeness is also closed under intersection, we expect to see a similar case for the state complexity of the star-of-intersection.

Theorem 4. *The worst-case state complexity of the star-of-intersection of an m_1-state and an m_2-state, $m_1, m_2 \geq 3$, prefix-free regular languages is precisely $m_1m_2 - 2(m_1 + m_2) + 6$, where $|\Sigma| \geq 4$.*

Proof. Han et al. [11] gave a construction for the intersection of two prefix-free DFAs based on the Cartesian product of states. The main idea of the construction is to remove unreachable states and merge equivalent states that are identified based on the structural properties of prefix-free DFAs. Based on the construction, they showed that $m_1m_2-2(m_1+m_2)+6$ states are sufficient for the intersection of an m_1-state prefix-free minimal DFA and an m_2-state prefix-free minimal DFA. Since prefix-freeness is closed under intersection, the resulting minimal DFA is also prefix-free. Therefore, $m_1m_2 - 2(m_1 + m_2) + 6$ states are sufficient for the star-of-intersection since the state complexity of the Kleene star is m for an m-state prefix-free minimal DFA [11].

We only need to prove the necessary part. Assume that $\Sigma = \{a, b, c, d\}$. Given a string w over Σ, let $|w|_a$ denote the number of a's in w. Let A_1 be the minimal DFA for

$$L(A_1) = \{wc \mid |w|_a \equiv 0 \pmod{m_1 - 2}, \text{ for } w \in \{a, b\}^*\}$$

and A_2 be the minimal DFA for

$$L(A_2) = \{wc \mid |w|_b \equiv 0 \pmod{m_2 - 2}, \text{ for } w \in \{a, b\}^*\}.$$

It is easy to verify that $L(A_1)$ and $L(A_2)$ are prefix-free and $|A_1| = m$ and $|A_2| = n$. Let $L = (L(A_1) \cap L(A_2))^*$. We claim that the minimal DFA for L needs $mn - 2(m+n) + 6$ states. To prove the claim, it is sufficient to show that there exists a set R of $mn - 2(m+n) + 6$ strings over Σ that are pairwise inequivalent modulo the right invariant congruence of L, $\equiv_L$.

Let $R = R_1 \cup R_2$, where

$$R_1 = \{\lambda, d\},$$
$$R_2 = \{a^i b^j \mid 1 \leq i \leq m_1 - 2 \text{ and } 1 \leq j \leq m_2 - 2\}.$$

Any string $a^i b^j$ from R_2 cannot be equivalent with λ since $a^i b^j \cdot \lambda \notin L$ but $\lambda \cdot \lambda \in L$. Similarly, $a^i b^j$ cannot be equivalent with d since $a^i b^j \cdot a^{m_1-2-i} b^{m_2-2-j} c \in L$ but $d \cdot a^{m_1-2-i} b^{m_2-2-j} c \notin L$. Note that λ and d are not equivalent as well.

Next, consider two distinct strings $a^i b^j$ and $a^k b^l$ from R_2. Since $a^i b^j \neq a^k b^l$, $a^i b^j \cdot a^{m_1-2-i} b^{m_2-2-j} c \in L$ but $a^k b^l \cdot a^{m_1-2-i} b^{m_2-2-j} c \notin L$. Therefore, any two distinct strings from R_2 are pairwise inequivalent.

Therefore, all $mn - 2(m+n) + 6$ strings in R are pairwise inequivalent. This concludes the proof. □

4 Conclusions

Recently, researchers started looking at state complexity of combined operations [19,25,26]. Usually, we can obtain a much lower state complexity for combined operations compared with the composition of state complexities of individual operations. However, in a few cases, the state complexity of combined operations and the composition of state complexities are similar. We have examined prefix-free regular languages and computed the state complexity of combined operations.

operation	complexity	operation	complexity
$L_1 \cup L_2$	$m_1 m_2 - 2$	$(L_1 \cup L_2)^*$	$5 \cdot 2^{m_1+m_2-6}$
$L_1 \cap L_2$	$m_1 m_2 - 2(m_1 + m_2) + 6$	$(L_1 \cap L_2)^*$	$m_1 m_2 - 2(m_1 + m_2) + 6$
$L_1 \cdot L_2$	$m_1 + m_2 - 2$	$(L_1 \cdot L_2)^*$	$m_1 + m_2 - 2$
L_1^R	$2^{m_1-2} + 1$	$(L_1^R)^*$	$2^{m_1-2} + 1$
L_1^*	m_1	$(L_1^*)^* = L_1^*$	m_1

The table summaries the state complexity of basic operations and the state complexity for combined operations of prefix-free regular languages. Note that prefix-freeness is closed under both intersection and catenation and both operations have the same state complexity. On the other hand, for union, the state complexity of the star-of-union is much lower. An interesting case is the reversal case. We know that prefix-freeness is not closed under reversal since the reversal of a prefix-free language is suffix-free. However, the complexities of single and combined operations are the same. We think this is because suffix-free minimal

DFAs preserve a certain structural property. Therefore, natural future work is to examine the state complexity of combined operations for suffix-free regular languages.

References

1. Friedl, J.E.F.: Mastering Regular Expressions. O'Reilly & Associates, Inc., Sebastopol (2002)
2. Harel, D., Politi, M.: Modeling Reactive Systems with Statecharts: The Statemate Approach. McGraw-Hill, New York (1998)
3. Karttunen, L.: Applications of finite-state transducers in natural language processing. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 34–46. Springer, Heidelberg (2001)
4. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoretical Computer Science 125(2), 315–328 (1994)
5. Yu, S.: State complexity of regular languages. Journal of Automata, Languages and Combinatorics 6(2), 221–234 (2001)
6. Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages. Theoretical Computer Science 320(2-3), 315–329 (2004)
7. Câmpeanu, C., Culik II, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 60–70. Springer, Heidelberg (2001)
8. Han, Y.S., Salomaa, K.: State complexity of union and intersection of finite languages. International Journal of Foundations of Computer Science 19(3), 581–595 (2008)
9. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function. International Journal of Foundations of Computer Science 13(1), 145–159 (2002)
10. Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. International Journal of Foundations of Computer Science 14(6), 1087–1102 (2003)
11. Han, Y.S., Salomaa, K., Wood, D.: State complexity of prefix-free regular languages. In: Proceedings of DCFS 2006, pp. 165–176 (2006); Full version is submitted for publication
12. Han, Y.S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 501–512. Springer, Heidelberg (2007)
13. Câmpeanu, C., Salomaa, K., Yu, S.: Tight lower bound for the state complexity of shuffle of regular languages. Journal of Automata, Languages and Combinatorics 7(3), 303–310 (2002)
14. Domaratzki, M.: State complexity of proportional removals. Journal of Automata, Languages and Combinatorics 7(4), 455–468 (2002)
15. Domaratzki, M., Salomaa, K.: State complexity of shuffle on trajectories. Journal of Automata, Languages and Combinatorics 9(2-3), 217–232 (2004)
16. Hricko, M., Jirásková, G., Szabari, A.: Union and intersection of regular languages and descriptional complexity. In: Proceedings of DCFS 2005, pp. 170–181 (2005)
17. Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation of regular languages. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 178–189. Springer, Heidelberg (2005)

18. Nicaud, C.: Average state complexity of operations on unary automata. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 231–240. Springer, Heidelberg (1999)
19. Gao, Y., Salomaa, K., Yu, S.: The state complexity of two combined operations: Star of catenation and star of reversal. Fundamenta Informaticae 83(1-2), 75–89 (2008)
20. Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations. Theoretical Computer Science 383(2-3), 140–152 (2007)
21. Salomaa, K., Yu, S.: On the state complexity of combined operations and their estimation. International Journal of Foundations of Computer Science 18, 683–698 (2007)
22. Berstel, J., Perrin, D.: Theory of Codes. Academic Press, Inc., London (1985)
23. Wood, D.: Theory of Computation. John Wiley & Sons, Inc., New York (1987)
24. Leiss, E.L.: Succinct representation of regular languages by boolean automata. Theoretical Computer Science 13, 323–330 (1981)
25. Ellul, K., Krawetz, B., Shallit, J., Wang, M.W.: Regular expressions: New results and open problems. Journal of Automata, Languages and Combinatorics 9, 233–256 (2004)
26. Rampersad, N.: The state complexity of L^2 and L^k. Information Processing Letters 98(6), 231–234 (2006)

Towards a Taxonomy for ECFG and RRPG Parsing

Kees Hemerik

Dept. of Mathematics and Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
c.hemerik@tue.nl

Abstract. Extended Context-Free Grammars (ECFGs) and Regular Right-Part Grammars (RRPGs) have many applications, but they are sparsely covered in the vast literature on parsing and grammars. This paper presents first steps towards a taxonomy of parsers for ECFGs and RRPGs, in order to make this subject more accessible.

1 Introduction

Extended Context-Free Grammars (ECFGs) and Regular Right-Part Grammars (RRPGs) are two common extensions of Context-Free Grammars (CFGs) with many applications. In computer science they are quite popular as formalisms for describing programming languages, often in the forms of EBNF (Extended Backus-Naur Form) [1] and *syntax diagrams*. There even exists an ISO standard for EBNF [2]. In computational linguistics, they often appear in the form of *transition networks* [3,4,5,6]. In spite of their importance for practical language definitions and tools, they are but sparsely covered in the vast literature on grammars and parsing. To the best of our knowledge, there does not exist a uniform, coherent treatment of the subject of ECFGs, RRPGs, and their parsers.

Taxonomies, in a precise technical sense, are an effective means of presenting such a subject. A taxonomy is a systematic classification of problems and solutions in a particular (algorithmic) problem domain.Taxonomies also are a good starting point for the construction of highly coherent algorithm toolkits. In the past, taxonomies and/or toolkits of this kind have been constructed for sorting [7,8], garbage collection [9], string pattern matching, finite automata construction and minimization [10,11], and, recently, acceptors and pattern matchers for regular trees [12,13].

In this paper we present first steps towards a taxonomy of parsers for ECFGs and RRPGs. Roughly, the field can be subdivided into four subfields, viz. recursive descent, *LL*-like, *LR*-like, and tabular parsing. We outline a taxonomy of recursive descent parsing for ECFGs, present a literature overview for *LR*-parsing of ECFGs and RRPGs, and briefly comment on the other two subfields. In doing so, we hope to make the subject more accessible to interested readers.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 410–421, 2009.

2 Definitions

We only outline definitions of some of the main notions.

An *extended context-free grammar* (ECFG) is a 4-tuple (N, P, T, S), where N and T are finite sets of *nonterminals* and *terminals*, $S \,\epsilon\, N$ is the *start symbol*, and P (the *productions*) is a map from N to regular expressions over alphabet $N \cup T$. Similarly, a *Regular Right Part Grammar* (RRPG) is a 4-tuple (N, P, T, S), where P is a map from N to finite automata over alphabet $N \cup T$. Both grammar kinds have productions with right parts which are regular languages over $N \cup T$. The language generated by such a grammar can be defined in different ways:

- Using the standard derivation notions for CFGs, but with the possibly infinite sets of production rules of the ECFG or RRPG;
- Defining a rewrite system for regular expressions, with expressions in T^* as normal forms [14];
- Defining ECFG trees (i.e. trees in which each node is labeled with a subexpression of the ECFG and has a number of sons matching the label), and taking the frontiers of these trees [15];
- By interpreting the ECFG as a set of regular language equations and taking the least solution w.r.t. the pointwise subset ordering.

It is obvious that each ECFG or RRPG can be transformed into a CFG by transforming each right part to a context-free sub-grammar. Heilbrunner [16] defines two such transformations - called $LL(1)$ *transformation* and *unambiguous right-linear transformation* - and uses these to define the $ELL(k)$ and $ELR(k)$ grammars as follows:

- An ECFG is an $ELL(k)$ *grammar* if there is an $LL(1)$ transformation yielding an $LL(k)$ grammar;
- An ECFG is an $ELR(k)$ grammar if there is an unambiguous right-linear transformation yielding an $LR(k)$ grammar.

3 Recursive Descent

Recursive descent is a technique for implementing a parser in the form of a recursive program. Taking a few notational shortcuts, the most basic recursive descent scheme can be described as follows.

Let $G = (N, T, P, S)$ be an ECFG, with $P = \{A_1 = e_1, \cdots, A_n = e_n\}$. P is mapped to the following set of procedure declarations:

$$\begin{aligned} &\textbf{proc}\ A_1 = \mathcal{E}[\![e_1]\!] \\ &\qquad\vdots \\ &\textbf{proc}\ A_n = \mathcal{E}[\![e_n]\!] \end{aligned}$$

where the function $\mathcal{E}$, which maps regular expressions over $N \cup T$ to statements, is defined recursively as:

re	$\mathcal{E}[\![re]\!]$
ε	$skip$
a	$term(a)$
A	A
$e_1 \bullet \cdots \bullet e_n$	$\mathcal{E}[\![e_1]\!]; \cdots ; \mathcal{E}[\![e_n]\!]$
$e_1 \mid \cdots \mid e_n$	**if** $sym \in LA(e_1) \rightarrow \mathcal{E}[\![e_1]\!] \,[\!]\, \cdots \,[\!]\, sym \in LA(e_n) \rightarrow \mathcal{E}[\![e_n]\!]$ **fi**
e^*	**do** $sym \in FIRST(e) \rightarrow \mathcal{E}[\![e]\!]$ **od**

In this scheme, variable *sym* is the lookahead symbol, procedure $term(a)$ checks whether *sym* equals terminal a and, if so, advances *sym* to the next input symbol, and functions LA and $FIRST$ yield $ELL(1)$ first sets and lookahead sets respectively.

Recursive descent is a well-known and popular technique. Its origins are hard to trace. One of the oldest written accounts is given by Lucas in [17]. Conway [3] seems to have been the first to apply it to transition networks, which are essentially RRPGs. Recursive descent occurs in many textbooks on compilers, usually in the form of examples, but there exist only few publications describing general schemes. The most notable are:

- Wirth describes in several places (e.g.[18], ch.5) how syntax diagrams can be mapped to Pascal programs;
- Lewi and his co-workers [19,15] give detailed accounts of translation schemes (some of them including error recovery) from ECFGs to Ada-like code;
- Wilhelm and Maurer ([14], ch.8) give translation schemes for ECFGs, based on dotted rules.

3.1 Sub-taxonomy for Recursive Descent

Although the basic scheme for recursive descent is well-understood, in practice there exist many variations, depending a.o. on which of the following aspects are taken into account:

Grammar processing. We can distinguish between *interpreting* parsers – which are fixed recursive procedures that take a grammar element as a parameter and try to recognize a terminal production of that grammar element – and *generated* parsers, which are sets of grammar specific recursive procedures, generated by a parser generator. The choice between the two kinds is one of flexibility vs. speed.

Handling of alternatives. We distinguish various strategies:

Deterministic. In this case the grammar should satisfy the $ELL(k)$ condition, which allows choices to be resolved by k-symbol lookahead;

Limited backtrack. In this case all alternatives are pursued, but the parser stops at the first success;

Full backtrack. In this case all alternatives are pursued completely, resulting in sets of solutions.

Additional tasks. Parsing is rarely done for its own sake. Usually it is combined with other tasks, such as tree construction or transduction from input to output language.

Error handling. Some options are: abortion, graceful termination, error recovery, and error correction.

Optimizations. E.g. memoization to make backtracking practically feasible.

Based on these details, it is possible to develop a sub-taxonomy for recursive descent parsing of ECFGs. We are currently working on this sub-taxonomy. Some results have been reported in [20,21]. Figure 1 shows the taxonomy graph. All parsers in this taxonomy have been described in an abstract notation of the kind commonly found in texts on algorithms or programming methods. An example of this style is given in Appendix A.

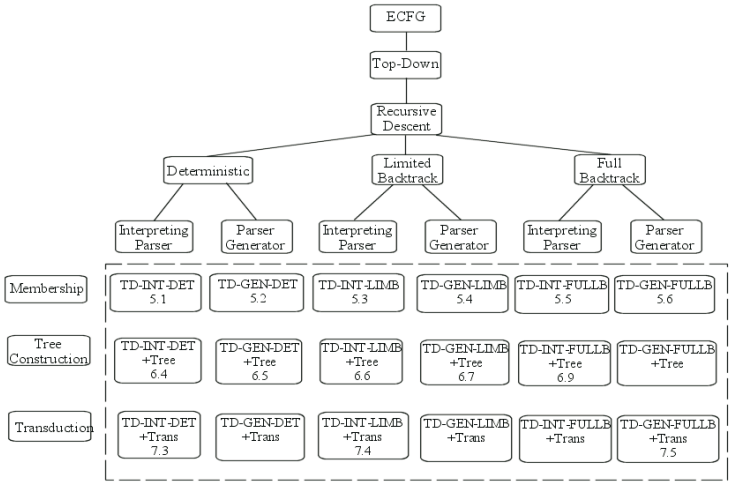


Fig. 1. A taxonomy diagram of the family of recursive descent parsers. The diagram has been taken from [20]. The numbers in the diagram refer to corresponding sections in that document.

4 *LL*-Like Parsers

Little has been published about *LL*-like push-down automata for ECFGs. A classical *LL*-parser for an ordinary CFG (N, T, P, S) is a push-down automaton with a stack alphabet corresponding (roughly) to $N \cup T$. A terminal on top of the stack is matched away against the lookahead symbol; a nonterminal on top of the stack is replaced by one of its right-hand sides. The actions are determined by a parse-table indexed by the top-of-stack symbol and the lookahead symbol.

This model can simply be generalized to ECFGs by using stack symbols which are dotted production rules, and modifying the transition relation accordingly. Wilhelm and Maurer ([14], ch.8) give the construction in full detail.

A somewhat different approach is taken by Brüggemann-Klein and Wood [22,23]. In their proposal the stack is not present explicitly, but represented by the list of uncompleted subtrees of the partial parse tree that is being built up.

5 *LR*-Like Parsers

In this section we consider parsing methods for RRPGs which are adaptations or generalizations of *LR*-like parsers for CFGs. Conventional *LR*-like parsers are of the shift-reduce type; they work on a stack and an input stream by means of two operations: a *shift* operation, which removes a symbol from the input and pushes a state on the stack, and a *reduce* operation by a specified production, which pops a number (dependent on the production) of states from the stack and pushes a new state on the stack. The top state and lookahead symbols determine which action to take. The actions are encoded in parse tables indexed by states and lookahead symbols.

Because RRPGs may lead to derivations with production right parts of unbounded lengths, determining the number of states to pop (the length of the *handle*) becomes a problem. In the literature the following main approaches to this problem can be distinguished:

1. Transform the RRPG to an equivalent CFG and apply standard techniques for constructing an *LR* parser [24,16,25];
2. Use the RRPG directly to construct an *LR*-like parser, extended with a mechanism to read back into the stack to determine the left end of the handle [26,27,28,29,25];
3. Use the RRPG to construct a shift-reduce parser which only pushes a state if it may correspond to the beginning of a production and only pops one state upon reduction [30,31,32,33,34,35];
4. Embedding of some of the techniques mentioned above in different frameworks for *LR*-like parsing [36,37].

In the following four subsections we will elaborate these approaches.

5.1 Transformation of RRPG to CFG

As already indicated in Section 2, any RRPG can be transformed into a CFG by replacing each right part by a context-free sub-grammar. This suggests one way of solving the parsing problem for RRPGs: first transform to an equivalent CFG, and then use any parsing method for CFGs. There are some disadvantages to such an approach, however: choosing an unsuitable transformation may result in a CFG with undesirable parsing properties. Also, it may be difficult to relate the structure of the input under the new CFG to the structure according to the original ECFG. The following papers discuss some transformations:

- Heilbrunner [16] is concerned with the definition of the notions of $ELL(k)$ and $ELR(k)$ grammar. To this end he uses certain transformations from ECFG to CFG, as we have already outlined in Section 2.
- Madsen and Christensen [25] transform an ECFG to a CFG by replacing each right part by an equivalent right-linear subgrammar. They use this transformation to define their $ELR(k)$ notion. Heilbrunner [16] shows that an ECFG is $ELR(k)$ according to this notion iff it is unambiguous and $ELR(k)$ according to his own notion.
- Celentano [24] gives a transformation from an ECFG to an equivalent CFG and uses this transformation as the basis for describing the construction of an *LR*-like parser for an ECFG, which simulates a conventional *LR* parser for an equivalent CFG.

5.2 *LR*-Parsers with Readback

The main problem with *LR*-like parsers for RRPGs is to locate the left end of the handle on the stack. It is not sufficient to construct a DFA (deterministic finite automaton) for each production $A \to e$ to recognize the reverse of the language of e. Instead, the *LR*-parser can be extended with *readback automata*, which can be constructed to recognize the reverse of the sequences of states entered by the *LR*-automaton as it reads strings in the language of e. These states distinguish between different contexts in which a symbol appears, so they can be used to find the context corresponding to the first symbol of a handle.

- LaLonde was the first to elaborate the idea of recognizing handles in the way just outlined [28,29]. LaLonde mentions that his parse tables can be divided into essentially two classes (*readahead* and *readback*), but this is not so evident from the way he describes the parsing algorithm.
- Chapman [26,27] gives a simpler construction for the readback automata, derived from a particular algorithm for computing $LALR(1)$ lookahead sets. The construction may yield readback automata that are non-deterministic. Chapman restricts his method to grammars for which this is not the case. Chapman clearly distinguishes readahead and readback automata and describes the parsing algorithm as a piece of program in which the forward and backward phases can easily be recognized.
- Madsen and Christensen [25] base their method on ECFGs rather than RRPGs. They present an $LR(0)$ construction based on sets of items which are rules of the ECFG with markers. In order to determine the left end of the handle they use a recursive algorithm which matches the actual handle with the possible forms of the production right part subexpressions.

In order to give an impression of the way a parser with readback automata works, we present below a somewhat polished version of Chapman's algorithm in [27]. The components have the following meanings: $\overline{Q}$ normal parser states, $\overline{q_0}$ initial state, $\delta : \overline{Q} \times T \to \overline{Q}$ forward transitions, R readback states, $\overline{F} \subseteq R$ accepting readback states, $\rho : R \times \overline{Q} \to R$ readback transitions, $I : \overline{Q} \times P \to R$ initial readback states.

con $G = (N, T,$
$P, S) : RRPG; \{$ G is the given grammar$\}$
con $LRB = (\overline{Q}, N \cup T, P, \overline{q_0}, \overline{\delta}, R, \rho, I, \overline{F}); \{LRB$ is $LR(1)$ automaton with readback for $G\}$

var
$in : T^*; \{$input sequence$\}$
$sym : T; \{$lookahead symbol$\}$
$\overline{q} : \overline{Q}; \{$current state$\}$
σ : **stack of** $\overline{Q}; \{$parse stack$\}$

$in := INPUT \oplus \langle \dashv \rangle;$
$\overline{q} := \overline{q_0};$
$\sigma := \langle \overline{q} \rangle;$
$nextsym;$
do $\delta(\overline{q}, sym) :: shift(s) \rightarrow \overline{q} := s;$
$\sigma.push(\overline{q});$
$nextsym$
$[\!]$ $\delta(\overline{q}, sym) :: reduce(A \rightarrow \alpha) \rightarrow [\![$ **var** $r : R;$
$r := I(\overline{q}, A \rightarrow \alpha); \{$initial readback state$\}$
do $r \notin \overline{F} \rightarrow r := \rho(r, \sigma.top); \{$readback loop$\}$
$\sigma.pop$
od ;
let $s \in \overline{Q}$ **s.t.** $\delta(\sigma.top, A) :: shift(s);$
$\overline{q} := s;$
$\sigma.push(\overline{q})$
$]\!]$
od ;
if $\delta(\overline{q}, sym) :: accept \rightarrow$ skip
$[\!]$ $\delta(\overline{q}, sym) :: error \rightarrow errormessage(\sigma, sym)$
fi

5.3 Purdom and Brown's Technique

Purdom and Brown [32] start out with an RRPG where each right part is a DFA. To this grammar they apply a construction which is very similar to the standard $LALR(k)$-construction for CFGs. The main difference with the approaches in the previous subsection is in the handling of the stack. A state is stacked only when the symbol being processed indicates the beginning of a new right side. When the end of the production is found, exactly one entry is popped from the stack.

This approach does not work for every $ELR(k)$ grammar because for some (state, symbol) pairs it may not be possible to tell whether the parser is starting a new right side without seeing more of the input. In this situation the construction produces a parser with a *stacking conflict*.

Purdom and Brown also present an algorithm for transforming any $ELR(k)$ grammar whose grammar has stacking conflicts into one whose parser has none. They claim that this transformation permits to parse any $ELR(k)$ grammar with their method.

Several variations on the basic scheme by Purdom and Brown have been published, which differ mainly in the way stacking conflicts are handled:

- Nakata and Sassa [31] use lookback states at reduction if stacking conflicts occur;
- Shin and Choe [34] improve on the storage requirements of Nakata and Sassa [31] by using only kernel items in the construction and by allowing NFAs rather than DFAs in the right parts of productions;
- Fortes Gálvez [38] points out some errors in [31,34] by means of a counter-example.
- Sassa and Nakata [33] use counters at reduction if stacking conflicts occur.
- Zhang and Nakata [35] use path numbers at reduction if stacking conflicts occur;
- Morimoto and Sassa [30] additionally put production symbols on the stack during shift actions in order to reduce the work needed during reduce actions.

5.4 Embedding in Different LR-Like Frameworks

- Colby and Bermudez [36] generalize the counters of Sassa and Nakata [33] and embed the resulting parser in a general theory of *LR*-like parsers developed by Bermudez.
- Kannapinn's dissertation [37] consists of two parts. In the first part he studies how redundancy may be eliminated from *LR* parsers. In the second part he uses his framework to design an *LR*-parser for ECFGs. He first gives a very critical review of many existing publications on the same subject and then presents his own constructions, which are essentially as follows: right parts of ECFGs are mapped to equivalent FAs; these in turn are mapped to equivalent right-linear subgrammars. This way, the construction of *ELR*-parsers for ECFGs and for RRPGs can be unified and both can be reduced to the construction of *LR*-parsers for pure CFGs. It is a pity that this thorough and critical work, published in German, has not received a wider readership.

6 Tabular Parsers

Classical tabular parsers like Earley's method [39] can handle arbitrary CFGs, even ambiguous ones. Usually, they are presented in the form of a subject string $w \epsilon T^*$ and a matrix M, where M_{ij} contains a set of elements related to all derivations of substring w_{ij}. The matrix is filled by a kind of dynamic programming algorithm, and eventually M_{0n} holds information about all derivations of w.

Not much literature can be found on extending tabular methods to ECFGs:

- Leermakers [40] presents a formalism for Earley-like parsers, which accomodates the simulation of non-deterministic pushdown automata. In particular, the theory is applied to non-deterministic *LR*-parsers for Recursive Transition Networks, which are equivalent to RRPGs.
- Schneider [41] shows how ECFGs can be embedded in a more general formalism called *state transition grammars* (STG), and presents an Earley-like parser schema for STGs.
- Spanbroek [42] gives full formal derivations of ECFG versions of the Cocke-Younger-Kasami algorithm and the bottom-up Earley algorithm.

7 Concluding Remarks

The picture of parsing ECFGs and RRPGs emerging from the survey above is:

- Recursive descent parsing is so simple to use that not many feel tempted to publish about its theory;
- What has been published about *LR*-like parsing theory is so complex that not many feel tempted to use it;
- *LL*-like parsing seems to have little theoretical or practical relevance;
- Tabular parsing seems feasible, but is largely unexplored.

Of course, such a simplistic resumé does not do justice to the great efforts that have gone into research and implementation of the methods described. But it is a striking phenomenon that the ideas behind recursive descent parsing of ECFGs can be grasped and applied immediately, whereas most of the literature on *LR*-like parsing of RRPGs is very difficult to access. Given the developments in computing power and software engineering, and the practical importance of ECFGs and RRPGs, a uniform and coherent treatment of the subject seems in order. We hope that this paper is a useful step towards that goal.

Acknowledgments. Mark van den Brand, Loek Cleophas and an anonymous referee gave many useful comments on an earlier version of this paper. Due to space limitations they could not all be taken into account.

References

1. Wirth, N.: What can we do about the unnecessary diversity of notation for syntactic definitions? CACM 20, 822–823 (1977)
2. ISO/IEC: International Standard EBNF Syntax Notation. 14977 edn. (1996), http://www.iso.ch/cate/d26153.html
3. Conway, M.E.: Design of a separable transition-diagram compiler. CACM 6, 396–408 (1963)
4. Lomet, D.B.: A formalization of transition diagram systems. JACM 20, 235–257 (1973)

5. Perlin, M.: LR recursive transition networks for Earley and Tomita parsing. In: Procs. 29th Annual Meeting on Association for Computational Linguistics (ACL), Berkeley, California, pp. 98–105 (1991)
6. Woods, W.A.: Transition network grammars for natural language analysis. CACM 13, 591–606 (1970)
7. Broy, M.: Program construction by transformations: a family tree of sorting programs. In: Biermann, A.W., Guiho, G. (eds.) Computer Program Synthesis Methodologies, pp. 1–49. Reidel (1983)
8. Darlington, J.: A synthesis of several sorting algorithms. Acta Inf. 11, 1–30 (1978)
9. Jonkers, H.: Abstraction, specification and implementation techniques, with an application to garbage collection. PhD thesis, Technische Univ. Eindhoven (1983)
10. Watson, B.W.: Taxonomies and Toolkits of Regular Language Algorithms. PhD thesis, Technische Universiteit Eindhoven (1995)
11. Watson, B.W.: Implementing and using finite automata toolkits. In: Kornai, A. (ed.) Procs. 12th European Conference on Artificial Intelligence, Budapest, Hungary, pp. 97–100 (1996)
12. Cleophas, L.: Tree Algorithms - Two Taxonomies and a Toolkit. PhD thesis, Technische Universiteit Eindhoven (2008)
13. Cleophas, L.: Forest FIRE and FIRE Wood - tools for tree automata and tree algorithms. In: FSMNLP (2008)
14. Wilhelm, R., Maurer, D.: Compiler Design. Addison-Wesley, Reading (1995)
15. Lewi, J., de Vlaminck, K., Steegmans, E., Horebeek, I.V.: Software Development by LL(1) Syntax Description. Wiley, Chichester (1992)
16. Heilbrunner, S.: On the definition of ELR(k) and ELL(k) grammars. Acta Informatica 11, 169–176 (1979)
17. Lucas, P.: The structure of formula-translators. ALGOL Bulletin 16 (1961)
18. Wirth, N.: Algorithms + Data Structures = Programs. Prentice-Hall, Englewood Cliffs (1975)
19. Lewi, J., Vlaminck, K.D., Huens, J., Huybrechts, M.: The ELL(l) parser generator and the error recovery mechanism. Acta Informatica 10, 209–228 (1978)
20. Keim, E.: Theory and implementation of a family of parser components. Master's thesis, Dept. of Math. and Comp.Sci., Technische Universiteit Eindhoven (2007)
21. Ssanyu, J., Hemerik, K.: Algorithms for recursive descent parsing of ECFGs. (in preparation, 2008)
22. Brüggemann-Klein, A., Wood, D.: On predictive parsing and extended context-free grammars. In: Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2002. LNCS, vol. 2608, pp. 239–247. Springer, Heidelberg (2003)
23. Brüggemann-Klein, A., Wood, D.: The parsing of extended context-free grammars. HKUST Theoretical Computer Science Center Research Report HKUST-TCSC-2002-08, Hong Kong University (2002)
24. Celentano, A.: An LR parsing technique for extended context-free grammars. Computer Languages 6, 95–107 (1981)
25. Madsen, O.L., Kristensen, B.B.: LR-parsing of extended context free grammars. Acta Informatica 7, 61–73 (1976)
26. Chapman, N.P.: LALR(1,1) parser generation for regular right part grammars. Acta Informatica 21, 29–45 (1984)
27. Chapman, N.P.: LR Parsing - Theory and Practice. Cambridge Univ. Press, Cambridge (1987)
28. LaLonde, W.R.: Regular right part grammars and their parsers. CACM 20, 731–741 (1977)

29. LaLonde, W.R.: Lalonde: Constructing LR parsers for regular right part grammars. Acta Informatica 11, 177–193 (1979)
30. Morimito, S.I., Sassa, M.: Yet another generation of LALR parsers for regular right part grammars. Acta Informatica 37, 671–697 (2001)
31. Nakata, I., Sassa, M.: Generation of efficient LALR parsers for regular right part grammars. Acta Informatica 23, 149–162 (1986)
32. Purdom, P.W., Brown, C.A.: Parsing extended LR(k) grammars. Acta Informatica 15, 115–127 (1981)
33. Sassa, M., Nakata, I.: A simple realization of LR-parsers for regular right part grammars. Information Processing Letters 24, 113–120 (1987)
34. Shin, H.C., Choe, K.M.: An improved LALR(k) parser generation for regular right part grammars. Inf. Proc. Lett. 47, 123–129 (1993)
35. Zhang, Y., Nakata, I.: Generation of path directed LALR(k) parsers for regular right part grammars. J. Inf. Process. 14, 325–334 (1991)
36. Colby, J.E., Bermudez, M.E.: Lookahead LR parsing with regular right part grammars. Intern. J. Computer Math. 64, 1–15 (1997)
37. Kannapinn, S.: Eine Rekonstruktion der LR-Theorie zur Elimination von Redundanz mit Anwendung auf den Bau von ELR-Parsern. PhD thesis, Technische Universität Berlin (2001) (in German)
38. Fortes Gálvez, J.: A note on a proposed LALR parser for extended context-free grammars. Inf. Proc. Lett. 50, 303–305 (1994)
39. Earley, J.C.: An efficient context-free parsing algorithm. CACM 13, 94–102 (1970)
40. Leermakers, R.: How to cover a grammar. In: Procs. 27th Ann. Meeting of Association for Computational Linguistics, Vancouver, Canada, pp. 135–142 (1989)
41. Schneider, K.M.: Parsing schemata for grammars with variable number and order of constituents. In: Procs. 18th Conf. on Comp. Linguistics, vol. 2, pp. 733–739 (2000)
42. Spanbroek, M.: Towards a unification of regular and context-free recognition. Master's thesis, Dept. of Math. and Comp. Sci., Technische Universiteit Eindhoven (2005)

A Interpreting Limited Backtrack Recursive Descent Parser Scheme

As an example of the way the parsing algorithms are described in the taxonomy, we show the abstract code for one of the algorithms. The parser is a limited backtrack recursive descent parser which interprets the grammar and which makes use of a scanner *sc* and a tree builder *tb*. The parser is coded as a recursive procedure $T(\downarrow e, \uparrow r, \uparrow t)$, where input parameter e is a regular expression over terminals and nonterminals; boolean output parameter r indicates whether the parse was successful. If so, output parameter t holds the constructed tree. The characteristics of the algorithm and the corresponding code are given below:

grammar kind	*ECFG*
grammar conditions	no left recursion, no *Nullable*(e) for expressions of the form e^*
grammar processing	interpreted
architecture	recursive descent procedure with scanner *sc* and tree builder *tb*
strategy	limited backtrack, no lookahead, success boolean, no memo
scanner operations	*sym, nextsym, mark, reset*
tree builder actions	*mkeps*, *mkterm*, *mkprod*, *mkdot*, *mkstick*, *mkstar*

```
proc T(↓ e : SE; ↑ r : bool; ↑ t : Tree) =
  if e :: φ → r, t := false, nil
  [] e :: ε → r, t := true, tb.mkeps(ẽ)
  [] e :: term(a) → if sc.sym = a → sc.nextsym;
                                    r, t := true, tb.mkterm(ẽ, a)
                     [] sc.sym ≠ a → r, t := false, nil
                     fi
  [] e :: nonterm(A) → [[ var b : bool; t' : Tree
                        ▷ T(↓ P(A); ↑ b; ↑ t');
                          if b → t := tb.mkprod(ẽ, A, t')
                          [] ¬b → t := nil
                          fi ;
                          r := b
                        ]]
  [] e :: (e0 · ... · el−1) → [[ var j : nat; b : bool; t' : Tree; s : seq of Tree
                               ▷ j, b, s := 0, true, ⟨⟩;
                                 do (j ≠ l) ∧ b → T(↓ ej; ↑ b; ↑ t');
                                                  s := s ⊕ ⟨t'⟩
                                                  j := j + 1
                                 od ;
                                 if b → t := tb.mkdot(ẽ, s)
                                 [] ¬b → t := nil
                                 fi ;
                                 r := b;
                               ]]
  [] e :: (e0 | ... | el−1) → [[ var m, j : nat; b : bool; t' : Tree
                               ▷ m := sc.mark;
                                 j, b, t := 0, false, nil;
                                 do (j ≠ l) ∧ ¬b → T(↓ ej; ↑ b; ↑ t');
                                                   if b → t := tb.mkstick(ẽ, j, t')
                                                   [] ¬b → sc.reset(m)
                                                   fi ;
                                                   j := j + 1
                                 od ;
                                 r := b;
                               ]]
  [] e :: f* → [[ var m : nat; b : bool; t' : Tree; s : seq of Tree
                ▷ b, s := true, ⟨⟩;
                  do b → m := sc.mark;
                          T(↓ f; ↑ b; ↑ t');
                          if b → s := s ⊕ ⟨t'⟩
                          [] ¬b → sc.reset(m)
                          fi ;
                  od ;
                  r, t := true, tb.mkstar(ẽ, s)
                ]]
  fi
```

Counting Parameterized Border Arrays for a Binary Alphabet

Tomohiro I[1], Shunsuke Inenaga[2], Hideo Bannai[1], and Masayuki Takeda[1]

[1] Department of Informatics, Kyushu University
[2] Graduate School of Information Science and Electrical Engineering, Kyushu University
744 Motooka, Nishiku, Fukuoka, 819–0395 Japan
{tomohiro.i,bannai,takeda}@i.kyushu-u.ac.jp,
inenaga@c.csce.kyushu-u.ac.jp

Abstract. The parameterized pattern matching problem is a kind of pattern matching problem, where a pattern is considered to occur in a text when there exists a renaming bijection on the alphabet with which the pattern can be transformed into a substring of the text. A *parameterized border array* (*p-border array*) is an analogue of a border array of a standard string, which is also known as the failure function of the Morris-Pratt pattern matching algorithm. In this paper we present a linear time algorithm to verify if a given integer array is a valid p-border array for a binary alphabet. We also show a linear time algorithm to compute all binary parameterized strings sharing a given p-border array. In addition, we give an algorithm which computes all p-border arrays of length at most n, where n is a a given threshold. This algorithm runs in time linear in the number of output p-border arrays.

1 Introduction

1.1 Parameterized Matching and Parameterized Border Array

The *parameterized matching* (*p-matching*) problem [1] is a kind of string matching problem, where a pattern is considered to occur in a text when there exists a renaming bijection on the alphabet with which the pattern can be transformed into a substring of the text. A *parameterized string* (*p-string*) is formally an element of $(\Pi \cup \Sigma)^*$, where Π is the set of *parameter symbols* and Σ the set of *constant symbols*. The renaming bijections used in p-matching are the identity on Σ, that is, every constant symbol $\mathtt{X} \in \Sigma$ is mapped to $\mathtt{X}$, while symbols in Π can be interchanged. Parameterized matching has applications in software maintenance [2,1], plagiarism detection [3], and RNA structural matching [4], thus it has been extensively studied in the last decade [5,6,7,8,9,10,11,12].

Of various efficient methods solving the p-matching problem, this paper focuses on the algorithm of Idury and Schäffer [13] that solves the p-matching problem for multiple patterns. Their algorithm modifies the Aho-Corasick automata [14], replacing the *goto* and *fail* functions with the *pgoto* and *pfail* functions, respectively. When the input is a single pattern p-string of length m, the

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 422–433, 2009.

pfail function can be implemented by an array of length m, and we call the array the *parameterized border array* (*p-border array*) of the pattern p-string, which is the parameterized version of the border array [15]. The p-border array of a given pattern can be computed in linear time [13].

1.2 Reverse and Enumerating Problems on Strings

The reverse problem for standard border arrays [15] was first introduced by Franěk et al. [16]. They proposed a linear time algorithm to verify if a given integer array is the border array of some string. Their algorithm works for both bounded and unbounded alphabets. Duval et al. [17] proposed a simpler algorithm to solve the same problem in linear time for bounded alphabets.

Moore et al. [18] presented an algorithm to enumerate all border arrays of length at most n, where n is a given positive integer. They proposed a notion of *b-equivalence* of strings such that two strings are b-equivalent if they have the same border array. The lexicographically smallest one of each b-equivalence class is called *b-canonical* string of the class. Their algorithm is also able to output all b-canonical strings of length up to a given integer n. Franěk et al. [16] pointed out that the time complexity analysis of [18] is incorrect, and showed a new algorithm which solves the same problem in $O(b_n)$ time using $O(b_n)$ space, where b_n denotes the number of border arrays of length at most n.

The reverse problem for some other string data structures, such as suffix arrays [19], directed acyclic word graphs [20], directed acyclic subsequence graphs [21] have been solved in linear time [22,23]. The problem of enumerating all suffix arrays was considered in [24]. An algorithm to enumerate all p-distinct strings was proposed in [18], where two strings are said to be p-distinct if they do *not* parameterized-match.

1.3 Our Contribution

This paper considers the reversal of the problem of computing the p-border array of a given pattern p-string. That is, given an integer array α, determine if there exists a p-string whose p-border array is α. In this paper, we present a linear time algorithm which solves the above problem for a binary parameter alphabet ($|\Pi| = 2$). We then consider a more challenging problem: given a positive integer n, enumerate all p-border arrays of length at most n. We propose an algorithm that solves the enumerating problem in $O(B_n)$ time for a binary parameter alphabet, where B_n is the number of all p-border arrays of length n for a binary parameter alphabet. We also give a simple algorithm to output all strings which share the same p-border array.

A p-border is a dual concept of a *parameterized period* of a p-string. Apostolico and Giancarlo [11] showed that a complete analogy to the weak periodicity lemma [25] stands for p-strings over a binary alphabet. Our result reveals yet another similarity of p-strings over a binary alphabet and standard strings in terms of periodicity.

2 Preliminaries

2.1 Parameterized String Matching

Let Σ and Π be two disjoint finite sets of *constant symbols* and *parameter symbols*, respectively. An element of $(\Sigma \cup \Pi)^*$ is called a *p-string*. The length of any p-string s is the total number of constant and parameter symbols in s and is denoted by $|s|$. For any p-string s of length n, the i-th symbol is denoted by $s[i]$ for each $1 \leq i \leq n$, and the *substring* starting at position i and ending at position j is denoted by $s[i : j]$ for $1 \leq i \leq j \leq n$. In particular, $s[1 : j]$ and $s[i : n]$ denote the *prefix* of length j and the *suffix* of length $n - i + 1$ of s, respectively.

Any two p-strings s and t of the same length m are said to *parameterized match* if s can be transformed into t by applying a renaming function f from the symbols of s to the symbols of t, such that f is the identity on the constant alphabet. For example, let $\Pi = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, $\Sigma = \{\mathtt{X}, \mathtt{Y}\}$, $s = \mathtt{abcXabY}$ and $t = \mathtt{bcaXbcY}$. We then have $s \simeq t$ with the renaming function f such that $f(\mathtt{a}) = \mathtt{b}$, $f(\mathtt{b}) = \mathtt{c}$, $f(\mathtt{c}) = \mathtt{a}$, $f(\mathtt{X}) = \mathtt{X}$, and $f(\mathtt{Y}) = \mathtt{Y}$. We write $s \simeq t$ when s and t p-match.

Amir et al. [5] showed that we have only to consider p-strings over Π when considering p-matching.

Lemma 1 ([5]). *The p-matching problem on alphabet $\Sigma \cup \Pi$ is reducible in linear time to the p-matching problem on alphabet Π.*

2.2 Parameterized Border Arrays

As in the case of standard string matching, we can define the parameterized border (p-border) and the parameterized border array (p-border array).

Definition 1. *A* parameterized border *(*p-border*) of a p-string s of length n is any integer j such that $0 \leq j < n$ and $s[1 : j] \simeq s[n - j + 1 : n]$.*

For example, the set of p-borders of p-string $\mathtt{aabbaa}$ is $\{4, 2, 1, 0\}$, since $\mathtt{aabb} \simeq \mathtt{bbaa}$, $\mathtt{aa} \simeq \mathtt{aa}$, $\mathtt{a} \simeq \mathtt{a}$, and $\varepsilon \simeq \varepsilon$.

Definition 2. *The* parameterized border array *(*p-border array*) β_s of any p-string s of length n is an array of length n such that $\beta_s[i] = j$, where j is the longest p-border of $s[1 : i]$.*

For example, the p-border array of p-string $\mathtt{aabbaa}$ is $[0, 1, 1, 2, 3, 4]$.

When it is clear from the context, we abbreviate β_s as β.

The p-border array β_s of p-string s was first explicitly introduced by Idury and Schäffer [13] as the *pfail* function, where the *pfail* function is used in their Aho-Corasick [14] type algorithm that solves the p-matching problem for multiple patterns. Given a pattern p-string p of length m, the p-border array β_p can be computed in $O(m \log |\Pi|)$ time, and the p-matching problem can be solved in $O(n \log |\Pi|)$ time for any text p-string of length n.

2.3 Problems

This paper deals with the following problems.

Problem 1 (Verifying valid p-border array). Given an integer array α of length n, determine if there exists a p-string s such that $\beta_s = \alpha$.

Problem 2 (Computing all p-strings sharing the same p-border array). Given an integer array α which is a valid p-border array, compute every p-string s such that $\beta_s = \alpha$.

Problem 3 (Computing all p-border arrays). Given a positive integer n, compute all p-border arrays of length at most n.

In the following section, we give efficient solutions to the above problems for a binary alphabet, that is, $|\Pi| = 2$.

3 Algorithms

This section presents our algorithms which solve Problem 1, Problem 2 and Problem 3 for the case $|\Pi| = 2$.

We begin with the basic proposition on p-border arrays.

Proposition 1. *For any p-border array $\beta[1..i]$ of length $i \geq 2$, $\beta[1..i-1]$ is a p-border array of length $i-1$.*

Proof. Let s be any p-string such that $\beta_s = \beta$. It is clear from Definition 2 that $\beta_s[1..i-1]$ is the p-border array of the p-string $s[1:i-1]$. □

Due to the above proposition, given an integer array $\alpha[1..n]$, we can check if it is a p-border array of some string of length n by testing each element of α in increasing order (from 1 to n). If we find any $1 \leq i \leq n$ such that $\alpha[1..i]$ is not a p-border array of length i, then $\alpha[1..n]$ can never be a p-border of length n. In what follows, we show how to check each element of a given integer array in increasing order.

For any p-border array β of length n and any integer $1 \leq i \leq n$, let

$$\beta^k[i] = \begin{cases} \beta[i] & \text{if } k = 1, \\ \beta[\beta^{k-1}[i]] & \text{if } k > 1 \text{ and } \beta^{k-1}[i] \geq 1. \end{cases}$$

It follows from Definition 2 that the sequence $i, \beta[i], \beta^2[i], \ldots$ is monotone decreasing to zero, hence finite.

Lemma 2. *For any p-string s of length i, $\{\beta_s^1[i], \beta_s^2[i], \ldots, 0\}$ is the set of the p-borders of s.*

Proof. First we show by induction that for every k, $1 \leq k \leq k'$, $\beta_s^k[i]$ is a p-border of s, where k' is the integer such that $\beta_s^{k'}[i] = 0$. By Definition 2, $\beta_s^1[i]$ is the longest p-border of s. Suppose that for some k, $1 \leq k < k'$, $\beta_s^k[i]$ is a

p-border of s. Here $\beta_s^{k+1}[i]$ is the longest p-border of $\beta_s^k[i]$. Let f and g be the bijections such that

$$f(s[1])f(s[2])\cdots f(s[\beta_s^k[i]]) = s[i-\beta_s^k[i]+1:i],$$
$$g(s[1])g(s[2])\cdots g(s[\beta_s^{k+1}[i]]) = s[\beta_s^k[i]-\beta_s^{k+1}[i]+1:\beta_s^k[i]].$$

Since

$$\begin{aligned} & f(g(s[1]))f(g(s[2]))\cdots f(g(s[\beta_s^{k+1}[i]])) \\ &= f(s[\beta_s^k[i]-\beta_s^{k+1}[i]+1])f(s[\beta_s^k[i]-\beta_s^{k+1}[i]+2])\cdots f(s[\beta_s^k[i]]) \\ &= s[i-\beta_s^{k+1}[i]+1:i], \end{aligned}$$

we obtain $s[1:\beta_s^{k+1}[i]] \simeq s[i-\beta_s^{k+1}[i]+1:i]$. Hence $\beta_s^{k+1}[i]$ is a p-border of s.

We now show any other j is not a p-border of s. Assume for contrary that j, $\beta_s^{k+1}[i] < j < \beta_s^k[i]$, is a p-border of s. Let q be the bijection such that

$$q(s[i-j+1])q(s[i-j+2])\cdots q(s[i]) = s[1:j].$$

Since

$$\begin{aligned} & q(f(s[\beta_s^k[i]-j+1]))q(f(s[\beta_s^k[i]-j+2]))\cdots q(f(s[\beta_s^k[i]])) \\ &= q(s[i-j+1])q(s[i-j+2])\cdots q(s[i]) \\ &= s[1:j], \end{aligned}$$

we obtain $s[1:j] \simeq s[\beta_s^k[i]-j+1:\beta_s^k[i]]$. Hence j is a p-border of $s[1:\beta_s^k[i]]$. However this contradicts with the definition that $\beta_s^{k+1}[i]$ is the longest p-border of $s[1:\beta_s^k[i]]$. □

Lemma 3. *For any p-string s of length $i \geq 1$ and $a \in \Pi$, every p-border of sa is an element of the set $\{\beta_s^1[i]+1, \beta_s^2[i]+1, \ldots, 1\}$.*

Proof. Assume for contrary that sa has a p-border $j+1 \notin \{\beta_s^1[i]+1, \beta_s^2[i]+1, \ldots, 1\}$. Since $s[1:j+1] \simeq s[i-j+1:i]a$, we have $s[1:j] \simeq s[i-j+1:i]$ and j is a p-border of s. It follows from Lemma 2 that $j \in \{\beta_s^1[i], \beta_s^2[i], \ldots, 0\}$. This contradicts with the assumption. □

Based on Lemma 2 and Lemma 3, we can efficiently compute the p-border array β_s of a given p-string s. Also, our algorithm to solve Problem 1 is based on these lemmas. Note that Proposition 1, Lemma 2 and Lemma 3 hold for p-strings over Π of arbitrary size.

In the sequel we show how to select $m \in \{\beta_s^1[i]+1, \beta_s^2[i]+1, \ldots, 1\}$ such that $\beta_s[1..i]m$ is a valid p-border array of length $i+1$. The following proposition, lemmas and theorems hold for a binary parameter alphabet, $|\Pi| = 2$.

For p-border arrays of length at most 2, we have the next proposition.

Proposition 2. *For any p-string s of length 1, $\beta_s[1] = 0$. For any p-string s' of length 2, $\beta_{s'}[2] = 1$.*

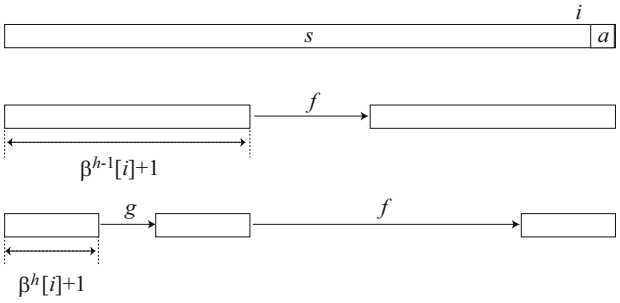


Fig. 1. Illustration for Lemma 5

Proof. Let $\Pi = \{a, b\}$. It is clear that the longest p-border of a and b is 0. The p-strings of length 2 over Π are aa, ab, ba, and bb. Obviously the longest p-border of each of them is 1. □

For p-border arrays of length more than 2, we have the following lemmas.

Lemma 4. *For any p-string $s \in \Pi^*$, if $j \geq 2$ is a p-border of sa with $a \in \Pi$, then j is not a p-border of sb, where $b \in \Pi - \{a\}$.*

Proof. Assume for contrary that j is a p-border of sb. Then, let f and g be the bijections on Π such that

$$f(s[1])f(s[2])\cdots f(s[j]) = s[i-j+2:i]a,$$
$$g(s[1])g(s[2])\cdots g(s[j]) = s[i-j+2:i]b.$$

We get from $f(s[1])f(s[2])\cdots f(s[j-1]) = s[i-j+2:i] = g(s[1])f(s[2])\cdots g(s[j-1])$ that f and g are the same bijections. However, $f(s[j]) = a \neq b = g(s[j])$ implies that f and g are different bijections, a contradiction. Hence j is not a p-border of sb. □

Lemma 5. *For any p-string s of length i, if $\beta_s[\beta_s^{h-1}[i]+1] = \beta_s^h[i]+1$ and $\beta_s^{h-1}[i]+1$ is a p-border of sa with $a \in \Pi$, then $\beta_s^h[i]+1$ is a p-border of sa. (See also Fig. 1.)*

Proof. Let f and g be the bijections on Π such that

$$f(s[1])f(s[2])\cdots f(s[\beta_s^{h-1}[i]+1]) = s[i-\beta_s^{h-1}[i]+1:i]a,$$
$$g(s[1])g(s[2])\cdots g(s[\beta_s^h[i]+1]) = s[\beta_s^{h-1}[i]-\beta_s^h[i]+1:\beta_s^{h-1}[i]+1].$$

Since

$$\begin{aligned} & f(g(s[1]))f(g(s[2]))\cdots f(g(s[\beta_s^h[i]+1])) \\ &= f(s[\beta_s^{h-1}[i]-\beta_s^h[i]+1])f(s[\beta_s^{h-1}[i]-\beta_s^h[i]+2])\cdots f(s[\beta_s^{h-1}[i]+1]) \\ &= s[i-\beta_s^h[i]+1:i]a, \end{aligned}$$

we obtain $s[1:\beta_s^h[i]+1] \simeq s[i-\beta_s^h[i]+1:i]a$. Hence $\beta_s^h[i]+1$ is a p-border of sa. □

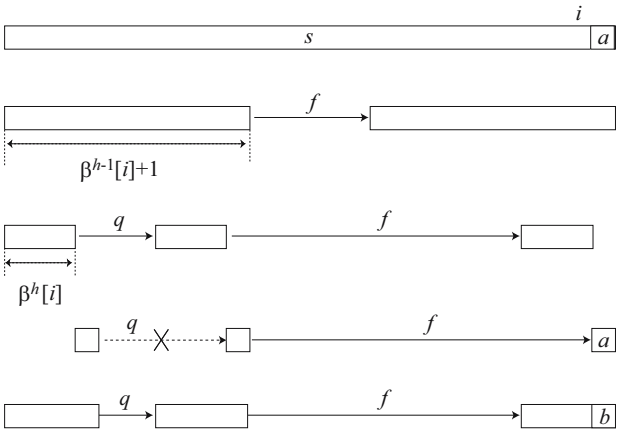


Fig. 2. Illustration for Lemma 6

Lemma 6. *For any p-string s of length i, if $\beta_s[\beta_s^{h-1}[i]+1] \neq \beta_s^h[i]+1$ and $\beta_s^{h-1}[i]+1$ is a p-border of sa with $a \in \Pi$, then $\beta_s^h[i]+1$ is a p-border of sb such that $b \in \Pi - \{a\}$. (See also Fig. 2.)*

Proof. Let f and g be the bijections on Π such that

$$f(s[1])f(s[2])\cdots f(s[\beta_s^{h-1}[i]+1]) = s[i-\beta_s^{h-1}[i]+1:i]a,$$
$$q(s[1])q(s[2])\cdots q(s[\beta_s^h[i]]) = s[\beta_s^{h-1}[i]-\beta_s^h[i]+1:\beta_s^{h-1}[i]].$$

Because $\beta_s[\beta_s^{h-1}[i]+1] \neq \beta_s^h[i]+1$, we know that $q(s[\beta_s^h[i]+1]) \neq s[\beta_s^{h-1}[i]+1]$. Since $f(s[\beta_s^{h-1}[i]+1]) = a$ and $\Pi = \{a,b\}$, $f(q(s[\beta_s^h[i]+1])) = b$. Hence $\beta_s^h[i]+1$ is a p-border of sb. □

The following is a key lemma to solving our problems.

Lemma 7. *For any p-border array β of length $i \geq 2$, $\beta[1..i]m_1$ and $\beta[1..i]m_2$ are the p-border arrays of length $i+1$, where $m_1 = \beta[i]+1$ and*

$$m_2 = \begin{cases} \beta^l[i]+1 & \text{if } \beta[\beta^{l-1}[i]+1] \neq \beta^l[i]+1 \text{ for some } 1<l<k' \text{ and} \\ & \beta[\beta^{h-1}[i]+1] = \beta^h[i]+1 \text{ for any } 1<h<l, \\ 1 & \text{otherwise,} \end{cases}$$

where k' is the integer such that $\beta^{k'}[i]=0$.

Proof. Consider any p-string s of length i such that $\beta_s = \beta$. By definition, there exists a bijection f on Π such that $f(s[1])f(s[2])\cdots f(s[\beta[i]]) = s[i-\beta[i]+1:i]$. Let $a = f(s[\beta[i]+1])$. Then $f(s[1])f(s[2])\cdots f(s[\beta[i]])f(s[\beta[i]+1]) = s[i-\beta[i]+1:i]a$. Note that $\beta[1..i](\beta[i]+1)$ is the p-border array of sa because sa can have no p-borders longer than $\beta[i]+1$.

It follows from Lemma 5 that $\beta^h[i]+1$ is a p-border of sa. Then, by Lemma 6, $\beta^l[i]+1$ is a p-border of sb. Since $\beta^h[i] \geq 1$, by Lemma 4, $\beta^h[i]+1$ is not a p-border of sb. Hence $\beta^l[i]+1$ is the longest p-border of sb. □

Algorithm 1. Algorithm to solve Problem 1

```
Input: α[1..n] : a given integer array
Output: return whether α is a valid p-border array or not
1  if α[1..2] ≠ [0, 1] then return invalid;
2  for i = 3 to n do
3  |  if α[i] = α[i − 1] + 1 then continue;
4  |  d′ ← α[i − 1];
5  |  d ← α[d′];
6  |  while d > 0 & d + 1 = α[d′ + 1] do
7  |  |  d′ ← d;
8  |  |  d ← α[d′];
9  |  if α[i] = d + 1 then continue;
10 |  return invalid;
11 return valid;
```

We are ready to state the following theorem.

Theorem 1. *Problem 1 can be solved in linear time for a binary parameter alphabet.*

Proof. Algorithm 1 describes the operations to solve Problem 1. Given an integer array of length n, the algorithm first checks if $\alpha[1..2] = [0, 1]$ due to Proposition 2. If $\alpha[1..2] = [0, 1]$, then for each $i = 3, \ldots, n$ (in increasing order) the algorithm checks whether $\alpha[i]$ satisfies one of the conditions of Lemma 7.

The time analysis is similar to that of Theorem 2.3 of [16]. In each iteration of the **for** loop, the value of d' increases by at most 1. However, each execution of the **while** loop decreases the value of d' by at least 1. Hence the total time cost of the **for** loop is $O(n)$. □

Theorem 2. *Problem 2 can be solved in linear time for a binary parameter alphabet.*

Proof. It follows from Proposition 2 that the p-border array of all p-string of length 2 (aa, ab, ba, and bb) is $[0, 1]$. By Proposition 1, for any p-border array $\beta[1..n]$ with $n \geq 2$, we have $\beta[1..2] = [0, 1]$. Hence each p-border array $\beta[1..n]$ with $n \geq 2$ corresponds to exactly four p-strings each of which begins with aa, ab, ba, and bb, respectively. Algorithm 2 is an algorithm to solve Problem 2. Technically x_{aa} can be computed by $s_{aa}[\beta[i]]$ xor $s_{aa}[\beta[i]+1]$ xor $s_{aa}[i]$ on binary alphabet $\Pi = \{0, 1\}$. Hence this counting algorithm works in linear time. □

We now consider Problem 3. By Proposition 1 and Lemma 7, computing all p-border arrays of length at most n can be accomplished using a rooted tree structure T_n of height $n - 1$. Each node of T_n of height $i - 1$ corresponds to an integer j such that j is the longest p-border of some p-string of length i over a binary alphabet, hence the path from the root to that node represents the p-border array of the p-string. Fig. 3 represents T_4.

Algorithm 2. Algorithm to compute all p-strings sharing the same p-border array

Input: $\beta[1..n]$: a p-border array
Output: all p-strings sharing the same p-border array $\beta[1..n]$

$s_{aa} \leftarrow aa$; $s_{ab} \leftarrow ab$; $s_{bb} \leftarrow bb$; $s_{ba} \leftarrow ba$;
for $i = 3$ **to** n **do**
  Let f be the bijection on Π s.t. $f(s_{aa}[\beta[i]]) = s_{aa}[i]$;
  Let g be the bijection on Π s.t. $g(s_{ab}[\beta[i]]) = s_{ab}[i]$;
  $x_{aa} \leftarrow f(s_{aa}[\beta[i]+1])$; $x_{ab} \leftarrow g(s_{ab}[\beta[i]+1])$;
  $\overline{x_{aa}} \leftarrow y \in \Pi - \{x_{aa}\}$; $\overline{x_{ab}} \leftarrow z \in \Pi - \{x_{ab}\}$;
  if $\beta[i] = \beta[i-1]+1$ **then**
    $s_{aa}[i] \leftarrow x_{aa}$; $s_{ab}[i] \leftarrow x_{ab}$;
    $s_{bb}[i] \leftarrow \overline{x_{aa}}$; $s_{ba}[i] \leftarrow \overline{x_{ab}}$;
  else
    $s_{aa}[i] \leftarrow \overline{x_{aa}}$; $s_{ab}[i] \leftarrow \overline{x_{ab}}$;
    $s_{bb}[i] \leftarrow x_{aa}$; $s_{ba}[i] \leftarrow x_{ab}$;
  end
Output: $s_{aa}[1:n]$, $s_{ab}[1:n]$, $s_{bb}[1:n]$, $s_{ba}[1:n]$

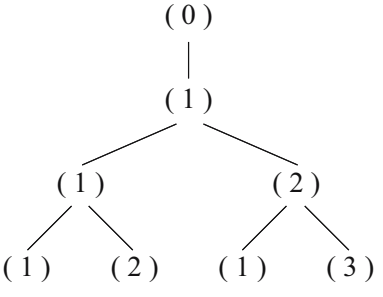


Fig. 3. The tree T_4 which represents all p-border arrays of length at most 4 for a binary alphabet

Theorem 3. *Problem 3 can be solved in $O(B_n)$ time for a binary parameter alphabet, where B_n denotes the number of p-border arrays of length n.*

Proof. Proposition 2 and Lemma 7 imply that every internal node of T_n of height at least 1 has exactly two children. Hence the total number of nodes of T_n is $O(B_n)$. We compute T_n in a depth-first manner. Algorithm 3 shows a function that computes the children of a given node of T_n. It is not difficult to see that each child of a given node can be computed in amortized constant time. Hence Problem 3 can be solved in $O(B_n)$ time for a binary parameter alphabet. □

We remark that if each p-border array in T_n can be discarded after it is generated, then we can compute all p-border arrays of length at most n using $O(n)$ space. Since every internal node of T_n of height at least 1 has exactly two children

Algorithm 3. Function to compute the children of a node of T_n

Input: i : length of the current p-border array, $2 \leq i \leq n$
Result: compute the children of the current node
// $\beta[1..n]$ `is allocated globally and` $\beta[1..i]$ `represents the current p-border array.`
1 **function** `getChildren`(i)
2 **if** $i = n$ **then return**;
3 $\beta[i+1] \leftarrow \beta[i] + 1$;
4 **report** $\beta[i+1]$;
5 `getChildren`$(i+1)$;
6 $d' \leftarrow \beta[i]$;
7 $d \leftarrow \beta[d']$;
8 **while** $d > 0$ & $d + 1 = \beta[d'+1]$ **do**
9 $\quad d' \leftarrow d$;
10 $\quad d \leftarrow \beta[d']$;
11 $\beta[i+1] \leftarrow d + 1$;
12 **report** $\beta[i+1]$;
13 `getChildren`$(i+1)$;
14 **return**;

and the root has one child, $B_n = 2^{n-2}$ for $n \geq 2$. Thus the space requirement can be reduced to $O(\log B_n)$.

4 Conclusions and Open Problems

A parameterized border array (p-border array) is a useful data structure for parameterized pattern matching. In this paper, we presented a linear time algorithm which tests if a given integer array is a valid p-border array for a binary alphabet. We also gave a linear time algorithm to compute all binary p-strings that share a given p-border array. Finally, we proposed an algorithm which computes all p-border arrays of length at most n, where n is a given threshold. This algorithm works in $O(B_n)$ time, where B_n denotes the number of p-border arrays of length n for a binary alphabet.

Problems 1, 2, and 3 are open for a larger alphabet. To see one of the reasons of why, we show that Lemma 4 does not hold for a larger alphabet. Consider a p-string $s = \mathtt{abac}$ over $\Pi = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Observe that $\beta_s = [0, 1, 2, 2]$. Although $\beta_s[4] = 2$ is a p-border of `abac`, it is also a p-border of another p-string `abab` since `ab` $\simeq$ `ab`. Hence Lemma 4 does not hold if $|\Pi| \geq 3$.

Our future work also includes the following:

- Verify if a given integer array is a *parameterized suffix array* [12].
- Compute all parameterized suffix arrays of length at most n.

In [12], a linear time algorithm which directly constructs the parameterized suffix array for a given binary string was proposed. This algorithm might be used as a basis for solving the above problems regarding parameterized suffix arrays.

References

1. Baker, B.S.: Parameterized pattern matching: Algorithms and applications. Journal of Computer and System Sciences 52(1), 28–42 (1996)
2. Baker, B.S.: A program for identifying duplicated code. Computing Science and Statistics 24, 49–57 (1992)
3. Fredriksson, K., Mozgovoy, M.: Efficient parameterized string matching. Information Processing Letters 100(3), 91–96 (2006)
4. Shibuya, T.: Generalization of a suffix tree for RNA structural pattern matching. Algorithmica 39(1), 1–19 (2004)
5. Amir, A., Farach, M., Muthukrishnan, S.: Alphabet dependence in parameterized matching. Information Processing Letters 49(3), 111–115 (1994)
6. Baker, B.S.: Parameterized pattern matching by Boyer-Moore-type algorithms. In: Proc. 6th annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1995), pp. 541–550 (1995)
7. Kosaraju, S.R.: Faster algorithms for the construction of parameterized suffix trees. In: Proc. 36th Annual Symposium on Foundations of Computer Science (FOCS 1995), pp. 631–637 (1995)
8. Hazay, C., Lewenstein, M., Tsur, D.: Two dimensional parameterized matching. In: Apostolico, A., Crochemore, M., Park, K. (eds.) CPM 2005. LNCS, vol. 3537, pp. 266–279. Springer, Heidelberg (2005)
9. Hazay, C., Lewenstein, M., Sokol, D.: Approximate parameterized matching. ACM Transactions on Algorithms 3(3), Article No. 29 (2007)
10. Apostolico, A., Erdös, P.L., Lewenstein, M.: Parameterized matching with mismatches. Journal of Discrete Algorithms 5(1), 135–140 (2007)
11. Apostolico, A., Giancarlo, R.: Periodicity and repetitions in parameterized strings. Discrete Applied Mathematics 156(9), 1389–1398 (2008)
12. Deguchi, S., Higashijima, F., Bannai, H., Inenaga, S., Takeda, M.: Parameterized suffix arrays for binary strings. In: Proc. The Prague Stringology Conference 2008 (PSC 2008), pp. 84–94 (2008)
13. Idury, R.M., Schäffer, A.A.: Multiple matching of parameterized patterns. Theoretical Computer Science 154(2), 203–224 (1996)
14. Aho, A.V., Corasick, M.J.: Efficient string matching: An aid to bibliographic search. Communications of the ACM 18(6), 333–340 (1975)
15. Morris, J.H., Pratt, V.R.: A linear pattern-matching algorithm. Technical Report Report 40, University of California, Berkeley (1970)
16. Franek, F., Gao, S., Lu, W., Ryan, P.J., Smyth, W.F., Sun, Y., Yang, L.: Verifying a border array in linear time. J. Combinatorial Math. and Combinatorial Computing 42, 223–236 (2002)
17. Duval, J.P., Lecroq, T., Lefevre, A.: Border array on bounded alphabet. Journal of Automata, Languages and Combinatorics 10(1), 51–60 (2005)
18. Moore, D., Smyth, W., Miller, D.: Counting distinct strings. Algorithmica 23(1), 1–13 (1999)
19. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Computing 22(5), 935–948 (1993)
20. Blumer, A., Blumer, J., Haussler, D., Ehrenfeucht, A., Chen, M.T., Seiferas, J.: The smallest automaton recognizing the subwords of a text. Theoretical Computer Science 40, 31–55 (1985)
21. Baeza-Yates, R.A.: Searching subsequences (note). Theoretical Computer Science 78(2), 363–376 (1991)

22. Duval, J.P., Lefebvre, A.: Words over an ordered alphabet and suffix permutations. Theoretical Informatics and Applications 36, 249–259 (2002)
23. Bannai, H., Inenaga, S., Shinohara, A., Takeda, M.: Inferring strings from graphs and arrays. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 208–217. Springer, Heidelberg (2003)
24. Schürmann, K.B., Stoye, J.: Counting suffix arrays and strings. Theoretical Computer Science 395(2-1), 220–234 (2008)
25. Lyndon, R.C., Schützenberger, M.P.: The equation $a^M = b^N c^P$ in a free group. Michigan Math. J. 9(4), 289–298 (1962)

Bounded Hairpin Completion

Masami Ito[1], Peter Leupold[1,⋆], and Victor Mitrana[2,⋆⋆]

[1] Department of Mathematics, Faculty of Science
Kyoto Sangyo University, Department of Mathematics
Kyoto 603-8555, Japan
{ito,leupold}@cc.kyoto-su.ac.jp
[2] University of Bucharest, Faculty of Mathematics and Computer Science
Str. Academiei 14, 010014, Bucharest, Romania
and
Department of Information Systems and Computation
Technical University of Valencia,
Camino de Vera s/n. 46022 Valencia, Spain
mitrana@fmi.unibuc.ro

Abstract. We consider a restricted variant of the hairpin completion called bounded hairpin completion. The hairpin completion is a formal operation inspired from biochemistry. Applied to a word encoding a single stranded molecule x such that either a suffix or a prefix of x is complementary to a subword of x, hairpin completion produces a new word z, which is a prolongation of x to the right or to the left by annealing.

The restriction considered here concerns the length of all prefixes and suffixes that are added to the current word by hairpin completion. They cannot be longer than a given constant. Closure properties of some classes of formal languages under the non-iterated and iterated bounded hairpin completion are investigated. We also define the inverse operation, namely bounded hairpin reduction, and consider the set of all primitive bounded hairpin roots of a regular language.

1 Introduction

This paper is a continuation of a series of works started with [4] (based on some ideas from [3]), where a new formal operation on words inspired by the DNA manipulation called hairpin completion was introduced. That initial work has been followed up by a several related papers ([11,12,13,14]), where both the hairpin completion as well as its inverse operation, namely the hairpin reduction, were further investigated.

Several problems remained unsolved in these papers. This is the mathematical motivation for the work presented here. By considering a weaker variant of the hairpin completion operation, called here the bounded hairpin completion, we

⋆ This work was done, while the author was funded as a post-doctoral fellow by the Japanese Society for the Promotion of Science under number P07810.

⋆⋆ Work supported by the Grant-in-Aid No. 19-07810 by Japan Society for the Promotion of Science.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 434–445, 2009.

hope to be able to solve some of the problems in this new setting that remained unsolved in the aforementioned papers. Another motivation is a practical one, closely related to the biochemical reality that inspired the definition of this operation. It seems more practical to consider that the prefix/suffix added by the hairpin completion cannot be arbitrarily long. In a laboratory every step of a computation will have to make do with a finite amount of resources and finite time; thus the length of the added string would be bounded by both the amount of additional nucleic acids in the test tube and the time given for one step of computation.

We briefly highlight some of the biological background that inspired the definition of the *Watson-Crick superposition* in [3]. The starting point is the structure of the DNA molecule. It consists of a double strand, each DNA single strand being composed by nucleotides which differ from each other in their bases: A (adenine), G (guanine), C (cytosine), and T (thymine). The two strands which form the DNA molecule are kept together by relatively weak hydrogen bonds between the bases: A always bonds with T and C with G. This phenomenon is usually referred to as *Watson-Crick complementarity*. The formation of these hydrogen bonds between complementary single DNA strands is called *annealing*.

A third essential feature from biochemistry is the PCR (Polymerase Chain Reaction). From two complementary, annealed strands, where one is shorter than the other, it produces a complete double stranded DNA molecule as follows: enzymes called polymerases add the missing bases (if they are available in the environment) to the shorter strand called *primer* and thus turn it into a complete complement of the longer one called *template*.

We now informally explain the superposition operation and how it can be related to the aforementioned biochemical concepts. Let us consider the following hypothetical biological situation: two single stranded DNA molecules x and y are given such that a suffix of x is Watson-Crick complementary to a prefix of y or a prefix of x is Watson-Crick complementary to a suffix of y, or x is Watson-Crick complementary to a subword of y. Then x and y get annealed in a DNA molecule with a double stranded part by complementary base pairing and then a complete double stranded molecule is formed by DNA polymerases. The mathematical expression of this hypothetical situation defines the superposition operation. Assume that we have an alphabet and a complementary relation on its letters. For two words x and y over this alphabet, if a suffix of x is complementary to a prefix of y or a prefix of x is complementary to a suffix of y, or x is complementary to a subword of y, then x and y bond together by complementary letter pairing and then a complete double stranded word is formed by the prolongation of x and y. Now the word obtained by the prolongation of x is considered to be the result of the superposition applied to x and y. Clearly, this is just a mathematical operation that resembles a biological reality considered here in an idealized way.

On the other hand, it is known that a single stranded DNA molecule might produce a hairpin structure, a phenomenon based on the first two biological principles mentioned above. Here one part of the strand bonds to another part of the same strand. In many DNA-based algorithms, these DNA molecules often

cannot be used in the subsequent steps of the computation. Therefore it has been the subject of a series of studies to find encodings that will avoid the formation of hairpins, see e.g. [5,6,7] or [10] and subsequent work for investigations in the context of Formal Languages. On the other hand, those molecules which may form a hairpin structure have been used as the basic feature of a new computational model reported in [18], where an instance of the 3-SAT problem has been solved by a DNA-algorithm whose second phase is mainly based on the elimination of hairpin structured molecules.

We now consider again a hypothetical biochemical situation: we are given one single stranded DNA molecule z such that either a prefix or a suffix of z is Watson-Crick complementary to a subword of z. Then the prefix or suffix of z and the corresponding subword of z get annealed by complementary base pairing and then z is lengthened by DNA polymerases up to a complete hairpin structure. The mathematical expression of this hypothetical situation defines the hairpin completion operation. By this formal operation one can generate a set of words, starting from a single word. This operation is considered in [4] as an abstract operation on formal languages. Some algorithmic problems regarding the hairpin completion are investigated in [11]. In [12] the inverse operation to the hairpin completion, namely the hairpin reduction, is introduced and one compares some properties of the two operations. This comparison is continued in [13], where a mildly context-sensitive class of languages is obtained as the homomorphic image of the hairpin completion of linear context-free languages. This is, to our best knowledge, the first class of mildly context-sensitive languages obtained in a way that does not involve grammars or acceptors.

In the aforementioned papers, no restriction is imposed on the length of the prefix or suffix added by the hairpin completion. This fact seems rather unrealistic though this operation is a purely mathematical one and the biological reality is just a source of inspiration. On the other hand, several natural problems regarding the hairpin completion remained unsolved in the papers mentioned above. A usual step towards solving them might be to consider a bit less general setting and try to solve the problems in this new settings. Therefore, we consider here a restricted variant of the hairpin completion, called *bounded hairpin completion*. This variant assumes that the length of the prefix and suffix added by the hairpin completion is bounded by a constant.

2 Basic Definitions

We assume the reader to be familiar with the fundamental concepts of formal language theory and automata theory, particularly the notions of grammar and finite automaton [16] and basics from the theory of abstract families of languages [19].

An alphabet is always a finite set of letters. For a finite set A we denote by $card(A)$ the cardinality of A. The set of all words over an alphabet V is denoted by V^*. The empty word is written λ; moreover, $V^+ = V^* \setminus \{\lambda\}$. Two languages

are considered to be equal if they contain the same words with the possible exception of the empty word.

A concept from the theory of abstract families of languages that we will refer to is that of a *trio*. This is is a non-empty family of languages closed under non-erasing morphisms, inverse morphisms and intersection with regular languages. A trio is *full* if it is closed under arbitrary morphisms.

Given a word w over an alphabet V, we denote by $|w|$ its length, while $|w|_a$ denotes the number of occurrences of the letter a in w. If $w = xyz$ for some $x, y, z \in V^*$, then x, y, z are called prefix, subword, suffix, respectively, of w. For a word w, $w[i..j]$ denotes the subword of w starting at position i and ending at position j, $1 \leq i \leq j \leq |w|$. If $i = j$, then $w[i..j]$ is the i-th letter of w which is simply denoted by $w[i]$.

Let Ω be a "superalphabet", that is an infinite set such that any alphabet considered in this paper is a subset of Ω. In other words, Ω is the *universe* of the alphabets in this paper, i.e., all words and languages are over alphabets that are subsets of Ω. An *involution* over a set S is a bijective mapping $\sigma : S \longrightarrow S$ such that $\sigma = \sigma^{-1}$. Any involution σ on Ω such that $\sigma(a) \neq a$ for all $a \in \Omega$ is said to be a *Watson-Crick involution*. Despite the fact that this is nothing more than a fixed point-free involution, we prefer this terminology since the hairpin completion defined later is inspired by the DNA lengthening by polymerases, where the Watson-Crick complementarity plays an important role. Let $\overline{\cdot}$ be a Watson-Crick involution fixed for the rest of the paper. The Watson-Crick involution is extended to a morphism from Ω to Ω^* in the usual way. We say that the letters a and $\overline{a}$ are complementary to each other. For an alphabet V, we set $\overline{V} = \{\overline{a} \mid a \in V\}$. Note that V and $\overline{V}$ can intersect and they can be, but need not be, equal. Recall that the DNA alphabet consists of four letters, $V_{DNA} = \{A, C, G, T\}$, which are abbreviations for the four nucleotides and we may set $\overline{A} = T$, $\overline{C} = G$.

We denote by $(\cdot)^R$ the mapping defined by ${}^R : V^* \longrightarrow V^*$, $(a_1 a_2 \dots a_n)^R = a_n \dots a_2 a_1$. Note that ^{R} is an involution and an *anti-morphism* ($(xy)^R = y^R x^R$ for all $x, y \in V^*$). Note also that the two mappings $\overline{\cdot}$ and $\cdot^R$ commute, namely, for any word x, $(\overline{x})^R = \overline{x^R}$ holds.

The reader is referred to [4] or any of the subsequent papers [11,12,13,14] for the definition of the (unbounded) *k-hairpin completion*; it is essentially the same as for the bounded variant defined below, only without the bound $|\gamma| \leq p$. Thus the prefix or suffix added by hairpin completion can be arbitrarily long. By the mathematical and biological reasons mentioned in the introductory part, in this work we are interested in a restricted variant of this operation that allows only prefixes and suffixes of a length bounded by a constant to be added. Formally, if V is an alphabet, then for any $w \in V^+$ we define the p-bounded k-hairpin completion of w, denoted by $pHC_k(w)$, for some $k, p \geq 1$, as follows:

$$
\begin{aligned}
pHC \curvearrowleft_k (w) &= \{\overline{\gamma^R} w | w = \alpha\beta\overline{\alpha^R}\gamma, |\alpha| = k,\ \alpha, \beta \in V^+, \gamma \in V^*, |\gamma| \leq p\} \\
pHC \curvearrowright_k (w) &= \{w\overline{\gamma^R} | w = \gamma\alpha\beta\overline{\alpha^R}, |\alpha| = k,\ \alpha, \beta \in V^+, \gamma \in V^*, |\gamma| \leq p\} \\
pHC_k(w) &= HC \curvearrowleft_k (w) \cup HC \curvearrowright_k (w).
\end{aligned}
$$

This operation is schematically illustrated in Figure 1.

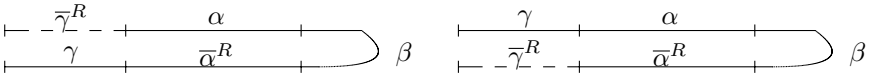


Fig. 1. Bounded hairpin completion

The *p-bounded hairpin completion* of w is defined by

$$pHC(w) = \bigcup_{k \geq 1} pHC_k(w).$$

As above, the p-bounded hairpin completion operation is naturally extended to languages by

$$pHC_k(L) = \bigcup_{w \in L} pHC_k(w) \qquad pHC(L) = \bigcup_{w \in L} pHC(w).$$

The iterated version of the p-bounded hairpin completion is defined in a similar way to the unbounded case, namely:

$$\begin{array}{ll} pHC_k^0(w) = \{w\}, & pHC^0(w) = \{w\}, \\ pHC_k^{n+1}(w) = pHC_k(pHC_k^n(w)), & pHC^{n+1}(w) = pHC(pHC^n(w)), \\ pHC_k^*(w) = \bigcup_{n \geq 0} pHC_k^n(w), & pHC^*(w) = \bigcup_{n \geq 0} pHC^n(w), \\ pHC_k^*(L) = \bigcup_{w \in L} pHC_k^*(w), & pHC^*(L) = \bigcup_{w \in L} pHC^*(w). \end{array}$$

3 The Non-iterated Case

The case of bounded hairpin completion is rather different in comparison to the unbounded variant considered in [11,12,13,14]. As it was expected, the closure problem of any trio under bounded hairpin completion is simple: every (full) trio is closed under this operation.

Proposition 1. *Every (full) trio is closed under p-bounded k-hairpin completion for any* $k, p \geq 1$.

Proof. It is sufficient to consider a generalized sequential machine (*gsm*) that adds a suffix (prefix) of length at most p to its input provided that its prefix (suffix) satisfies the conditions from the definitions. As every trio is closed under *gsm* mappings, see [19], we are done. □

We recall that neither the class of regular languages nor that of context-free languages is closed under unbounded hairpin completion. By the previous theorem, both classes are closed under bounded hairpin completion.

On the other hand, in [11] it was proved that if the membership problem for a given language L is decidable in $\mathcal{O}(f(n))$, then the membership problem for the hairpin completion of L is decidable in $\mathcal{O}(nf(n))$ for any $k \geq 1$. Further,

the factor n is not needed for the class of regular languages, but the problem of whether or not this factor is needed for other classes remained open in [11]. An easy adaption of the algorithm provided there shows that this factor is never needed in the case of bounded hairpin completion and thus membership is always decidable in $\mathcal{O}(f(n))$; presenting the adapted algorithm would exceed the scope of this work, though.

4 The Iterated Case

As in non-iterated case, the iterated bounded hairpin completion offers also a rather different picture of closure properties in comparison to the unbounded variant considered in the same papers cited above. We start with a general result.

Theorem 1. *Let $p, k \geq 1$ and $\mathcal{F}$ be a (full) trio closed under substitution. Then $\mathcal{F}$ is closed under iterated p-bounded k-hairpin completion if and only if $pHC_k^*(w) \in \mathcal{F}$ for any word w.*

Proof. The "only if" part is obvious as any trio contains all singleton languages.

For the "if" part, let $L \in \mathcal{F}$ be a language over the alphabet V. We write $L = L_1 \cup L_2$, where

$$L_1 = \{x \in L \mid |x| < 2(k+p)+1\},$$
$$L_2 = \{x \in L \mid |x| \geq 2(k+p)+1\}.$$

Clearly, $pHC_k^*(L) = pHC_k^*(L_1) \cup pHC_k^*(L_2)$. As any trio contains all finite languages, it follows that any trio closed under substitution is closed under union. Therefore, as L_1 is a finite language, we conclude that $pHC_k^*(L_1) \in \mathcal{F}$. Consequently, it remains to show that $pHC_k^*(L_2) \in \mathcal{F}$.

Let $\alpha, \beta \in V^+$ be two arbitrary words with $|\alpha| = |\beta| = k+p$. We define $L_2(\alpha, \beta) = L_2 \cap \{\alpha\}V^+\{\beta\}$. We have that

$$L_2 = \bigcup_{|\alpha|=|\beta|=k+p} L_2(\alpha,\beta) \quad \text{and} \quad pHC_k^*(L_2) = \bigcup_{|\alpha|=|\beta|=k+p} pHC_k^*(L_2(\alpha,\beta)).$$

On the other hand, it is plain that $pHC_k^*(L_2(\alpha,\beta)) = s(pHC_k^*(\alpha X \beta))$, where X is a new symbol not in V and s is a substitution $s : (V \cup \{X\})^* \longrightarrow 2^{V^*}$ defined by $s(a) = \{a\}$ for all $a \in V$ and $s(X) = \{w \in V^+ \mid \alpha w \beta \in L_2(\alpha,\beta)\}$. The language $\{w \in V^+ \mid \alpha w \beta \in L_2(\alpha,\beta)\}$ is in $\mathcal{F}$ (even $\mathcal{F}$ is not full) as it is the image of a language from $\mathcal{F}$, namely $L_2(\alpha,\beta)$, through a *gsm* mapping that deletes both the prefix and suffix of length $k+p$ of the input word. By the closure properties of $\mathcal{F}$, it follows that $pHC_k^*(L_2(\alpha,\beta))$ is in $\mathcal{F}$ for any α, β as above, which completes the proof. □

We recall that none of the families of regular, linear context-free, and context-free languages is closed under iterated unbounded hairpin completion. Here the bounded hairpin completion is much more tractable.

Corollary 1. *The family of context-free languages is closed under iterated p-bounded k-hairpin completion for any $k, p \geq 1$.*

Proof. By the previous result, it suffices to prove that $pHC_k(w)$ is context-free for any word w. Given $w \in V^+$, we construct the arbitrary grammar $G = (\{S, X\}, V \cup \{\#\}, S, P)$, where the set of productions P contains the following rules:

$$P = \{S \to yXz \mid w = zy\} \cup \{\overline{z}^R yXz \to \overline{z}^R yX\overline{y}^R z \mid 1 \leq |y| \leq p, |z| = k\}$$
$$\cup \{\overline{z}^R Xyz \to \overline{z^R y^R} Xyz \mid 1 \leq |y| \leq p, |z| = k\} \cup \{X \to \#\}.$$

By a result of Baker (see [1]), the language generated by G is context-free. Further we have that $pHC_k^*(w) = h(cp(L(G)) \cap V^+\{\#\})$. Here cp maps every word in the set of all its circular permutations and every language in the set of all circular permutations of its words, while h is a morphism that erases # and leaves unchanged all letters of V. As the class of context-free languages is closed under circular permutation [17], we infer that $pHC_k^*(w)$ is context-free. □

The above argument does not work for the class of linear context-free languages as this class is known not to be closed under circular permutation. However, also this family is closed under iterated bounded hairpin completion.

Theorem 2. *The family of linear context-free languages is closed under iterated p-bounded k-hairpin completion for any $k, p \geq 1$.*

Proof. Let L be a language generated by the linear grammar $G = (N, T, S, P)$. We construct the linear grammar $G' = (N', T, S', P')$, where

$$N' = N \cup \{S'\} \cup \{[\alpha, \beta] \mid \alpha, \beta \in T^*, 0 \leq |\alpha|, |\beta| \leq k + p\}$$
$$\cup \{[\alpha, A, \beta] \mid \alpha, \beta \in T^*, 0 \leq |\alpha|, |\beta| \leq k + p, A \in N\},$$

and the set of productions P' is defined by (in the definition of every set $\alpha, \beta \in T^*, 0 \leq |\alpha|, |\beta| \leq k + p, A \in N$ holds):

$$P' = P \cup \{S' \to S\} \cup \{S' \to [\alpha, \beta] \mid \alpha, \beta \in T^*, 0 \leq |\alpha|, |\beta| \leq k + p\}$$
$$\cup \{[\alpha, \beta] \to [\alpha', \beta']\overline{y}^R \mid \alpha = \alpha' = yvw, \beta = u\overline{v^R y^R}, |v| = k, |y| \leq p,$$
$$\beta' = xu\overline{v}^R, x \in T^*, |\beta'| \leq k + p\}$$
$$\cup \{[\alpha, \beta] \to y[\alpha', \beta']\overline{y}^R \mid \alpha = yvw, \beta = \beta' = u\overline{v^R y^R}, |v| = k, |y| \leq p,$$
$$\alpha' = vwx, x \in T^*, |\alpha'| \leq k + p\}$$
$$\cup \{[\alpha, \beta] \to [\alpha, S, \beta] \mid \alpha, \beta \in T^*, 0 \leq |\alpha|, |\beta| \leq k + p\}$$
$$\cup \{[\alpha, A, \beta] \to x[\alpha', B, \beta']y \mid A \to xBy \in P, \alpha = x\alpha', \beta = \beta' y, \alpha', \beta' \in T^*\}$$
$$\cup \{[\alpha, A, \beta] \to \alpha x[\lambda, B, \beta']y \mid A \to \alpha xBy \in P, \beta = \beta' y, \beta' \in T^*\}$$
$$\cup \{[\alpha, A, \beta] \to x[\alpha', B, \lambda]y\beta \mid A \to xBy\beta \in P, \alpha = x\alpha', \alpha' \in T^*\}$$
$$\cup \{[\lambda, A, \lambda] \to A \mid A \in N\}.$$

It is rather easy to note that we have the derivation

$$S' \Longrightarrow^* x[\alpha, \beta]y \Longrightarrow x[\alpha, S, \beta]y \Longrightarrow^* x\alpha w\beta y$$

in G' if and only if $S \Longrightarrow^* \alpha w\beta$ in G and $x\alpha w\beta y \in pHC_k^*(\alpha w\beta)$. This concludes the proof. □

The problem of whether or not the iterated unbounded hairpin completion of a word is context-free is open. By the previous result, it follows that the iterated bounded hairpin completion of a word is always linear context-free. We do not know whether this language is always regular. More generally, the status of the closure under iterated bounded hairpin completion of the class of regular languages remains unsettled.

We finish this section with another general result.

Theorem 3. *Every trio closed under circular permutation and iterated finite substitution is closed under iterated bounded hairpin completion.*

Proof. We take two positive integers $k, p \geq 1$. Let $\mathcal{F}$ be a family of languages with the above properties and $L \subseteq V^*$ be a language in $\mathcal{F}$. Let L_1 be the circular permutation of $L\{\#\}$, where $\#$ is a new symbol not in V. Clearly, L_1 still lies in $\mathcal{F}$. We consider the alphabet $W = \{[x\#y] \mid x, y \in V^*, 0 \leq |x|, |y| \leq p + k\}$ and define the morphism $h : (W \cup V)^* \longrightarrow (V \cup \{\#\})^*$ by $h([x\#y]) = x\#y$, for any $[x\#y] \in W$, and $h(a) = a$, for any $a \in V$. We now consider the language $L_2 \in \mathcal{F}$ given by $L_2 = h^{-1}(L_1)$. By the closure properties of $\mathcal{F}$, the language $L_3 = s^*(L_2)$ is in $\mathcal{F}$, where s is the finite substitution $s : (W \cup V)^* \longrightarrow 2^{(W \cup V)^*}$ defined by $s(a) = \{a\}, a \in V$, and $s([x\#y]) = \{x\#y\} \cup R$ with

$$\begin{aligned}
R = \{&[x\#\overline{u}^R y] \mid x = vzu, y = \overline{z}^R w, u, v, z, w \in V^*, |z| = k,\\
&\quad |\overline{u}^R y| \leq p + k, |u| \leq p\} \cup\\
&\{[x\#\overline{u}^R y']y'' \mid x = vzu, y = \overline{z}^R w = y'y'', u, v, z, w, y', y'' \in V^*, |z| = k,\\
&\quad |\overline{u}^R y'| = p + k, |u| \leq p\}\} \cup\\
&\{[x\overline{u}^R\#y] \mid x = w\overline{z}^R, y = uzv, u, v, z, w \in V^*, |z| = k,\\
&\quad |x\overline{u}^R| \leq p + k, |u| \leq p\}\} \cup\\
&\{x''[x'\overline{u}^R\#y] \mid x = w\overline{z}^R = x''x', y = uzv, u, v, z, w, x', x'' \in V^*, |z| = k,\\
&\quad |x'\overline{u}^R| = p + k, |u| \leq p\}\}.
\end{aligned}$$

Finally, let L_4 be the circular permutation of $h(L_3)$. Then we obtain that $pHC_k^*(L) = g(L_4 \cap V^*\{\#\})$, where g is a morphism that removes $\#$ and leaves unchanged all symbols from V. □

5 An Inverse Operation: The Bounded Hairpin Reduction

We now define the inverse operation of the bounded hairpin completion, namely the bounded hairpin reduction in a similar way to [13], where the unbounded hairpin reduction is introduced. Let V be an alphabet, for any $w \in V^+$ we define

the *p-bounded k-hairpin reduction* of w, denoted by $pHR_k(w)$, for some $k, p \geq 1$, as follows:

$$pHR\circlearrowleft_k(w) = \{\alpha\beta\overline{\alpha^R\gamma^R} \mid w = \gamma\alpha\beta\overline{\alpha^R\gamma^R}, |\alpha| = k,\ \alpha,\beta,\gamma \in V^+, 1 \leq |\gamma| \leq p\},$$
$$pHR\circlearrowright_k(w) = \{\gamma\alpha\beta\overline{\alpha^R} \mid w = \gamma\alpha\beta\overline{\alpha^R\gamma^R}, |\alpha| = k,\ \alpha,\beta,\gamma \in V^+, 1 \leq |\gamma| \leq p\}.$$
$$pHR_k(w) = pHR\circlearrowleft_k(w) \cup pHR\circlearrowright(w).$$

The *p-bounded hairpin reduction* of w is defined by

$$pHR(w) = \bigcup_{k\geq 1} pHR_k(w).$$

The bounded hairpin reduction is naturally extended to languages by

$$pHR_k(L) = \bigcup_{w\in L} pHR_k(w) \qquad pHR(L) = \bigcup_{w\in L} pHR(w).$$

The iterated bounded hairpin reduction is defined analogously to the iterated bounded hairpin completion.

We recall that the problem of whether or not the iterated unbounded hairpin reduction of a regular language is recursively decidable is left open in [13]. The same problem for the iterated bounded hairpin reduction is now completely solved by the next more general result. Before stating the result, we need to recall a few notions about string-rewriting systems. To this aim, we follow the standard notations for string rewriting as in [2]. A string-rewriting system (SRS) over an alphabet V is a finite relation $R \subset V^* \times V^*$, and the rewrite relation induced by R is denoted by $\longrightarrow_R$. That is, we write $x \longrightarrow_R y$ if $x = uvw, y = uzw$, for some $u, v, z, w \in V^*$, and $(v, z) \in R$. As usual every pair $(v, z) \in R$ is referred to as a rule $v \to z$. The reflexive and transitive closure of $\longrightarrow_R$ is denoted by $\longrightarrow_R^*$. We use $R^*(L)$ for the closure of the language L under the string-rewriting system R. Formally, $R^*(L) = \{w \mid x \longrightarrow_R^* w, \text{ for some } x \in L\}$. A rule $v \to z$ is said to be *monadic* if it is length-reducing ($|v| > |z|$) and $|z| \leq 1$. A SRS is called monadic if all its rules are monadic. A class of languages $\mathcal{F}$ is closed under monadic SRS if for any language $L \in \mathcal{F}$ over some alphabet V and any monadic SRS R over V, $R^*(L) \in \mathcal{F}$ holds.

Theorem 4. *Every trio closed under circular permutation and monadic string-rewriting systems is closed under iterated bounded hairpin reduction.*

Proof. Let $\mathcal{F}$ be a trio and k, p be two positive integers. The central idea of the proof is as follows. We permute every word of a language in $\mathcal{F}$ in a circular way. Then the last and first letters are next to each other. Thus the hairpin reduction becomes a local operation and can be simulated by monadic string-rewriting rules. By our hypothesis, these are known to preserve the membership in $\mathcal{F}$.

To start with, we attach a new symbol X to the end of every word of a given $L \in \mathcal{F}$, $L \subseteq V^*$. Then we obtain the language L' by doing a circular permutation to all words in $L\{X\}$. Note that X marks the end and beginning of

the original words. On this language we apply a gsm-mapping g that introduces redundancy by adding to every letter information about its neighboring letters in the following way:

1. The letter containing the X contains also the $k+p$ letters to the left and to the right of X in order.
2. Every letter left of X contains the letter originally at that position and the $k+p$ letters left of it in order.
3. Every letter right of X contains the letter originally at that position and the $k+p$ letters right of it in order.

At the word's end and its beginning, where there are not enough letters to fill the symbols, some special symbol signifying a space is placed inside the compound symbols.

Now we can simulate a step of p-bounded k-hairpin reduction by a string-rewriting rule with a right-hand side of length one, i.e. a monadic one. A straightforward approach would be to use rules of the form $u\overline{v}^R X v\overline{u}^R \to uXv\overline{u}^R$. But we see that u and $Xv\overline{u}^R$ are basically not changed, they only form a context whose presence is necessary. Through our redundant representation of the word, their presence can be checked by looking only at the corresponding image of X under g. Further, since the symbols of the image of u under g contain only information about symbols to their left, they do not need to be updated after the deletion of $\overline{v}^R$ to preserve the properties 1 to 3. The same is true for $v\overline{u}^R$. Only in the symbol corresponding to X some updating needs to be done and thus it is the one that is actually rewritten. So the string rewriting rules are

$$g_{\mathsf{left}}(z_0 z_1 u\overline{v})[1\ldots|v|][z_1 u\overline{v}Xv\overline{u}z_2] \to [z_0 z_1 uXv\overline{u}z_2],$$

where g_{left} does the part of g described by property 2, and where $z_0, z_1 \in V^*$, $u, v \in V^+$, $|u| = k$, $|v| \leq p$, $|z_0 z_1 u| = p+k$. Analogously, rules that delete symbols to the right of X are defined. Let R be the string-rewriting system consisting of all such rules. It is immediate that $w' \in pHR_k(w) \Leftrightarrow g(cp(wX)) \to_R g(cp(w'X))$ and by induction $w' \in pHR_k^*(w) \Leftrightarrow g(cp(wX)) \to_R^* g(cp(w'X))$.

Therefore, at this point we have all circular permutations of words that can be reached by p-bounded k-hairpin reduction from words in L coded under g. To obtain our target language we first undo the coding of g by the gsm-mapping g' that projects all letters to the left of X to their last component, all letters to the right of X to their first component, and the symbol containing X to just X. This mapping is letter-to-letter, the gsm only needs to remember in its state whether is has already passes over the symbol containing X. Of the result of this we take again the circular permutation.

Now we filter out the words that have X at the last position and therefore are back in the original order of L and delete X. By the closure properties of $\mathcal{F}$, the result of this process lies in $\mathcal{F}$, which completes the proof. □

As monadic SRSs are known to preserve regularity (see [8]) we immediately infer that

Theorem 5. *The class of regular languages is closed under iterated bounded hairpin reduction.*

In [12] one considers another concept that seems attractive to us, namely the *primitive hairpin root* of a word and of a language. Given a word $x \in V^*$ and a positive integer k, the word y is said to be the primitive k-hairpin root of x if the following two conditions are satisfied:

$$(i)\ \ y \in HR_k^*(x) (\text{or, equivalent, } x \in HC_k^*(y)),$$
$$(ii)\ \ HR_k(y) = \emptyset.$$

Here $HR_k^*(z)$ delivers the iterated unbounded hairpin reduction of the word z. In other words, y can be obtained from x by iterated k-hairpin reduction (maybe in zero steps) and y cannot be further reduced by hairpin reduction. The primitive bounded hairpin root is defined analogously. Clearly, a word may have more than one primitive bounded hairpin root; the set of all primitive p-bounded k-hairpin roots of a word x is denoted by $pH_k root(x)$. Naturally, the primitive p-bounded k-hairpin root of a language L is defined by $pH_k root(L) = \bigcup_{x \in L} pH_k root(x)$.

Clearly, to see whether a word is reducible, one has to look only at the first and last $k + p$ symbols. By Theorem 5 we have:

Theorem 6. *$pH_k root(L)$ is regular for any regular language L and any $p, k \geq 1$.*

Proof. For the regular language $L' \subseteq V^*$ obtained in the proof of Theorem 5 it suffices to consider the language

$$\{w \in L \mid |w| \leq 2k+2\} \cup (pHR_k^*(L) \cap \{\alpha x \beta \mid |\alpha| = |\beta| = k+1, \alpha \neq \overline{\beta}^R, x \in V^+\})$$

which is regular and equals $pH_k root(L)$. □

6 Final Remarks

We have considered a restricted version of the hairpin completion operation by imposing that the prefix or suffix added by the hairpin completion are bounded by a constant. In some sense, this is the lower extreme case the upper extreme being the unbounded case that might be viewed as a linearly bounded variant. We consider that bounded variants by other sublinear mappings would be of theoretical interest.

Last but not least we would like to mention that hairpin completion and reduction resemble some language generating mechanisms considered in the literature like external contextual grammars with choice [15] or dipolar contextual deletion [9], respectively.

References

1. Baker, B.S.: Context-sensitive grammars generating context-free languages. In: Automata, Languages and Programming ICALP 1972, pp. 501–506. North-Holland, Amsterdam (1972)
2. Book, R., Otto, F.: String-Rewriting Systems. Springer, Heidelberg (1993)

3. Bottoni, P., Labella, A., Manca, V., Mitrana, V.: Superposition based on Watson-Crick-like complementarity. Theory of Computing Systems 39, 503–524 (2006)
4. Cheptea, D., Martín-Vide, C., Mitrana, V.: A new operation on words suggested by DNA biochemistry: hairpin completion. In: Transgressive Computing, pp. 216–228 (2006)
5. Deaton, R., Murphy, R., Garzon, M., Franceschetti, D.R., Stevens, S.E.: Good encodings for DNA-based solutions to combinatorial problems. In: Proc. of DNA-based computers II. DIMACS Series, vol. 44, pp. 247–258 (1998)
6. Garzon, M., Deaton, R., Neathery, P., Murphy, R.C., Franceschetti, D.R., Stevens, S.E.: On the encoding problem for DNA computing. In: The Third DIMACS Workshop on DNA-Based Computing, pp. 230–237. Univ. of Pennsylvania (1997)
7. Garzon, M., Deaton, R., Nino, L.F., Stevens, S.E., Wittner, M.: Genome encoding for DNA computing. In: Proc. Third Genetic Programming Conference, Madison, MI, pp. 684–690 (1998)
8. Hofbauer, D., Waldmann, J.: Deleting string-rewriting systems preserve regularity. Theoretical Computer Science 327, 301–317 (2004)
9. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. Information and Computation 131, 47–61 (1996)
10. Kari, L., Konstantinidis, S., Sosík, P., Thierrin, G.: On hairpin-free words and languages. In: De Felice, C., Restivo, A. (eds.) DLT 2005. LNCS, vol. 3572, pp. 296–307. Springer, Heidelberg (2005)
11. Manea, F., Martín-Vide, C., Mitrana, V.: On some algorithmic problems regarding the hairpin completion. Discrete Applied Mathematics (in press), doi:10.1016/j.dam.2007.09.022
12. Manea, F., Mitrana, V.: Hairpin completion versus hairpin reduction. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 532–541. Springer, Heidelberg (2007)
13. Manea, F., Mitrana, V., Yokomori, T.: Two complementary operations inspired by the DNA hairpin formation: completion and reduction. Theoretical Computer Science (in press), doi:10.1016/j.tcs.2008.09.049
14. Manea, F., Mitrana, V., Yokomori, T.: Some remarks on the hairpin completion. In: Proc. 12th International Conference on Automata and Formal Languages, pp. 302–313 (2008)
15. Marcus, S.: Contextual grammars. Rev. Roum. Math. Pures Appl. 14, 1525–1534 (1969)
16. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer, Heidelberg (1997)
17. Ruohonen, K.: On circular words and $(\omega^* + \omega)$-powers of words. Elektr. Inform. und Kybern. E.I.K. 13, 3–12 (1977)
18. Sakamoto, K., Gouzu, H., Komiya, K., Kiga, D., Yokoyama, S., Yokomori, T., Hagiya, M.: Molecular computation by DNA hairpin formation. Science 288, 1223–1226 (2000)
19. Salomaa, A.: Formal Languages. Academic Press, London (1973)

Rigid Tree Automata*

Florent Jacquemard[1], Francis Klay[2], and Camille Vacher[3]

[1] INRIA Saclay & LSV (CNRS/ENS Cachan)
florent.jacquemard@inria.fr
[2] FT/RD/MAPS/AMS/SLE
francis.klay@orange-ftgroup.com
[3] FT/RD & LSV (CNRS/ENS Cachan)
vacher@lsv.ens-cachan.fr

Abstract. We introduce the class of Rigid Tree Automata (RTA), an extension of standard bottom-up automata on ranked trees with distinguished states called rigid. Rigid states define a restriction on the computation of RTA on trees: RTA can test for equality in subtrees reaching the same rigid state. RTA are able to perform local and global tests of equality between subtrees, non-linear tree pattern matching, and restricted disequality tests as well. Properties like determinism, pumping lemma, boolean closure, and several decision problems are studied in detail. In particular, the emptiness problem is shown decidable in linear time for RTA whereas membership of a given tree to the language of a given RTA is NP-complete. Our main result is the decidability of whether a given tree belongs to the rewrite closure of a RTA language under a restricted family of term rewriting systems, whereas this closure is not a RTA language. This result, one of the first on rewrite closure of languages of tree automata with constraints, is enabling the extension of model checking procedures based on finite tree automata techniques. Finally, a comparison of RTA with several classes of tree automata with local and global equality tests, and with dag automata is also provided.

Introduction

Tree automata (TA) are finite representations of infinite sets of terms. In system and software verification, TA can be used to represent infinite sets of states of a system or a program (in the latter case, a term can represent the program itself), messages exchanged by a protocol, XML documents... In these settings, the closure properties of TA languages permit incremental constructions and verification problems can be reduced to TA problems decidable in polynomial time like emptiness (is the language recognized by a given TA empty) and membership (is a given term t recognized by a given TA).

Despite these nice properties, a big limitation of TA is their inability to test equalities between subterms during their computation: TA are able to detect linear patterns like $\mathsf{fst}(\mathsf{pair}(x_1, x_2))$ but not a pattern like $\mathsf{pair}(x, x)$. Several extensions of TA have been proposed to overcome this problem, by addition

* This work was partly supported by the ANR Sesur 07 project AVOTÉ.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 446–457, 2009.

of equality and disequality tests in TA transition rules (the classes [1,2] have a decidable emptiness problem), or an auxiliary memory containing a tree and memory comparison [3]. However, they are all limited to local tests, at a bounded distance from the current position.

In this paper, we define the *rigid tree automata* (RTA) by the identification of some states as *rigid*, and the condition that the subterms recognized in one rigid state during a computation are all equal. With such a formalism, it is possible to check local and global equality tests between subterms, and also the subterm relation or restricted disequalities. In Sections 2 and 3 we study issues like determinism, closure of languages under Boolean operations, comparison with related classes of automata and decision problems for RTA. RTA are a particular case of the more general class Tree Automata with General Equality and Disequality constraints (TAGED [4]). The study of the class RTA alone is motivated by the complexity results and applications mentioned below. But our most original contribution is the study of the rewrite closure of RTA languages.

Combining tree automata and term rewriting techniques has been very successful in verification see e.g. [5,6]. In this context, term rewriting systems (TRS) can describe the transitions of a system, the evaluation of a program [5], the specification of operators used to build protocol messages [7] or also the transformation of documents. In these approaches, the rewrite closure $\mathcal{R}^*\big(L(\mathcal{A})\big)$ of the language $L(\mathcal{A})$ of a TA $\mathcal{A}$ using $\mathcal{R}$ represents the set of states reachable from states described by $L(\mathcal{A})$. When $\mathcal{R}^*\big(L(\mathcal{A})\big)$ is again a TA language, the verification of a safety property amounts to check for the existence of an error state in $\mathcal{R}^*\big(L(\mathcal{A})\big)$ (either a given term t or a term in a given regular language). This technique, sometimes referred as regular tree model checking, has driven a lot of attention to the rewrite closure of tree automata languages. However, there has been very few studies of this issue for constrained TA (see e.g. [8]).

In Section 4, we show that it is decidable whether a given term t belongs to the rewrite closure of a given RTA language for a restricted class of TRS called linear invisibly, whereas this closure is generally not a RTA language. Linear invisibly TRS can typically specify cryptographic operators like $\mathsf{decrypt}(\mathsf{encrypt}(x, \mathsf{pk}(\mathsf{A})), \mathsf{sk}(\mathsf{A})) \to x$.

Using RTA instead of TA in a regular tree model checking procedure permits to handle processes with local and global memories taking their values in infinite domains and which can be written only once. For instance, our initial motivation for studying RTA was the analysis of security protocols in a model where a finite number of processes exchange messages (following a protocol) asynchronously over an insecure network controlled by an attacker who is able to tamper the messages. The messages are terms build over cryptographic operators and are interpreted modulo an invisibly TRS $\mathcal{R}$ with rules like the above one for $\mathsf{decrypt}$ [7]. It is possible to built a RTA $\mathcal{A}$ recognizing exactly the set of messages that can be exchanged by executing the protocol in presence of the active attacker. The RTA $\mathcal{A}$ models both the honest processes and the attacker, and uses one rigid state to memorize each message sent by an honest process In these settings, it is possible to express confidentiality problems as membership

modulo $\mathcal{R}$ (does $t \in \mathcal{R}^*\big(L(\mathcal{A})\big)$), and authentication like problems as emptiness of intersection with a TA (does $L(\mathcal{A}) \cap L(\mathcal{B}) = \emptyset$ for a TA $\mathcal{B}$ recognizing error traces). All the details and proofs omitted in this extended abstract due to space restrictions can be found in the long version [9].

1 Preliminaries

A *signature* Σ is a finite set of function symbols with arity. We write Σ_m for the subset of function symbols of Σ of arity m. Given an infinite set $\mathcal{X}$ of variables, the set of terms built over Σ and $\mathcal{X}$ is denoted $\mathcal{T}(\Sigma, \mathcal{X})$, and the subset of ground terms (terms without variables) is denoted $\mathcal{T}(\Sigma)$. The set of variables occurring in a term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is denoted $vars(t)$. A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ is called *linear* if every variable of $vars(t)$ occurs at most once in t. A *substitution* σ is a mapping from $\mathcal{X}$ to $\mathcal{T}(\Sigma, \mathcal{X})$.The application of a substitution σ to a term t is the homomorphic extension of σ to $\mathcal{T}(\Sigma, \mathcal{X})$.

A term t can be seen as a function from its set of *positions* $\mathcal{P}os(t)$ into function symbols of variables of $\Sigma \cup \mathcal{X}$. The positions of $\mathcal{P}os(t)$ are sequences of positive integers (ε, the empty sequence, is the root position). Position are compared wrt the prefix ordering: $p_1 < p_2$ iff there exists $p \neq \varepsilon$ such that $p_2 = p_1.p$. In this case, p is denoted $p_2 - p_1$. A subterm of t at position p is written $t|_p$, and the replacement in t of the subterm at position p by u is denoted $t[u]_p$. The depth $d(t)$ of t is the length of its longest position. A *n-context* is a linear term of $\mathcal{T}(\Sigma, \{x_1, \ldots, x_n\})$. The application of a *n-context* C to n terms $t_1, \ldots, t_n$, denoted by $C[t_1, \ldots, t_n]$, is defined as the application to C of the substitution $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$.

Term Rewriting. A *term rewrite system* (TRS) over a signature Σ is a finite set of rewrite rules $\ell \to r$, where $\ell \in \mathcal{T}(\Sigma, \mathcal{X})$ (it is called the left-hand side (lhs) of the rule) and $r \in \mathcal{T}(\Sigma, vars(\ell))$ (it is called right-hand-side (lhs)). A term $t \in \mathcal{T}(\Sigma, \mathcal{X})$ rewrites to s by a TRS $\mathcal{R}$ (denoted $t \xrightarrow[\mathcal{R}]{} s$) if there is a rewrite rule $\ell \to r \in \mathcal{R}$, a position $p \in \mathcal{P}os(t)$ and a substitution σ such that $t|_p = \sigma(\ell)$ and $s = t[\sigma(r)]_p$. In this case, t is called *reducible*. An irreducible term is also called an $\mathcal{R}$*-normal-form*. The transitive and reflexive closure of $\xrightarrow[\mathcal{R}]{}$ is denoted $\xrightarrow[\mathcal{R}]{*}$. Given $L \subseteq \mathcal{T}(\Sigma, \mathcal{X})$, we note $R^*(L) = \{t \mid \exists s \in L, s \xrightarrow[R]{*} t\}$. A TRS is called *linear* if all the terms in its rules are linear and *collapsing* if every rhs of rule is a variable.

Tree Automata. A *tree automaton* (TA) $\mathcal{A}$ on a signature Σ is a tuple $\langle Q, F, \Delta \rangle$ where Q is a finite set of nullary state symbols, disjoint from Σ, $F \subseteq Q$ is the subset of final states and Δ is a set of transition rules of the form: $f(q_1, \ldots, q_n) \to q$ where $n \geq 0$, $f \in \Sigma_n$, and $q_1, \ldots, q_n, q \in Q$. The *size* of $\mathcal{A}$ is the number of symbols in Δ. A *run* of the TA $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$ is a function $r : \mathcal{P}os(t) \to Q$ such that for all $p \in \mathcal{P}os(t)$ with $t(p) = f \in \Sigma_n$ $(n \geq 0)$, $f\big(r(p.1), \ldots, r(p.n)\big) \to r(p) \in \Delta$. We will sometimes use term-like notation for runs. For instance, a run $\{\varepsilon \mapsto q, 1 \mapsto q_1, 2 \mapsto q_2\}$ will be denoted $q(q_1, q_2)$.

The *language* $L(\mathcal{A}, q)$ of a TA $\mathcal{A}$ in state q is the set of ground terms for which there exists a run r of $\mathcal{A}$ such that $r(\varepsilon) = q$. If $q \in F$ then this run r is called *successful*. The language $L(\mathcal{A})$ of $\mathcal{A}$ is $\bigcup_{q \in F} L(\mathcal{A}, q)$, and a set of ground terms is called *regular* if it is the language of a TA.

A TA $\mathcal{A} = \langle Q, F, \Delta \rangle$ on Σ is *deterministic* (DTA), resp. *complete*, if for every $f \in \Sigma_n$, and every $q_1, \ldots, q_n \in Q$, there exists at most, resp. at least, one rule $f(q_1, \ldots, q_n) \to q \in \Delta$. In the deterministic (resp. complete) cases, given a tree t, there is at most (resp. at least) one run r of $\mathcal{A}$ on t.

2 RTA: Definition and First Properties

2.1 Definition and Examples

Definition 1. *A* rigid tree automaton *(RTA) $\mathcal{A}$ on a signature Σ is a tuple $\langle Q, R, F, \Delta \rangle$ where $\langle Q, F, \Delta \rangle$ is a tree automaton denoted $ta(\mathcal{A})$ and $R \subseteq Q$ is the subset of* rigid *states.*

A *run* of the RTA $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$ is a run r of the underlying TA $ta(\mathcal{A})$ on t with the additional condition that: for all positions $p_1, p_2 \in \mathcal{P}os(t)$, if $r(p_1) = r(p_2) \in R$ then $t|_{p_1} = t|_{p_2}$. Languages of RTA are defined the same way as for TA. Note that with these definitions, every regular language is a RTA language. We shall write below TA and RTA for the classes of TA and RTA languages.

Example 1. Let $\Sigma = \{a : 0, b : 0, f : 2\}$. The set $\{f(t, t) \mid t \in \mathcal{T}(\Sigma)\}$ is recognized by the RTA on Σ $\mathcal{A} = \big\langle \{q, q_r, q_\mathsf{f}\}, \{q_r\}, \{q_\mathsf{f}\}, \{a \to q|q_r, b \to q|q_r, f(q, q) \to q|q_r, f(q_r, q_r) \to q_\mathsf{f}\} \big\rangle$, where $a \to q|q_r$ is an abbreviation for $a \to q$ and $a \to q_r$. A successful run of $\mathcal{A}$ on $f(f(a, b), f(a, b))$ is $q_\mathsf{f}\big(q_r(q, q), q_r(q, q)\big)$. ◇

Note that the above RTA language is not regular; RTA generalize to non-linear pattern the (linear) pattern matching ability of TA.

Example 2. Let us extend the RTA of Example 1 with the transitions rules $f(q, q_\mathsf{f}) \to q_\mathsf{f}$, $f(q_\mathsf{f}, q) \to q_\mathsf{f}$ ensuring the propagation of the final state q_f up to the root. The RTA obtained recognizes the set of terms of $\mathcal{T}(\Sigma)$ containing the pattern $f(x, x)$. ◇

Proposition 1. *For all term $t \in \mathcal{T}(\Sigma, \mathcal{X})$, there exists a RTA recognizing the terms of $\mathcal{T}(\Sigma)$ which have a ground instance of t as a subterm.*

But RTA are not limited to testing equalities. Using rigid states permits to test disequality and inequality as well, like the subterm relation.

Example 3. Let $\Sigma = \{a : 0, b : 0, f : 2, < : 2\}$. The set of terms $<(s, t)$ such that $s, t \in \mathcal{T}\big(\Sigma \setminus \{<\}\big)$ and s is a subterm of t is recognized by the RTA on Σ $\langle \{q, q_r, q', q_\mathsf{f}\}, \{q_r\}, \{q_\mathsf{f}\}, \Delta \rangle$ with $\Delta = \{a \to q|q_r, b \to q|q_r, f(q, q) \to q|q_r, f(q, q_r) \to q', f(q_r, q) \to q', f(q, q') \to q', f(q', q) \to q', <(q_r, q') \to q_\mathsf{f}\}$. For instance, a successful run on $<\big(a, f(a, b)\big)$ is $q_\mathsf{f}\big(q_r, q'(q_r, q)\big)$. The idea is that

in a successful run, the rigid state q_r identifies (by a non-deterministic choice) the subterm s on the left side of $<$, and the state q' is reached immediately above q_r and propagated up to the root, in order to express that the right side t of $<$ is a superterm of s. ◇

RTA can also test disequalities between subterms built only with unary and constant symbols.

Example 4. Let $\Sigma = \{c : 0, a : 1, b : 1, \neq : 2\}$. The set of terms of $\mathcal{T}(\Sigma)$ of the form $\neq (s, t)$, where $s, t \in \mathcal{T}\big(\Sigma \setminus \{\neq\}\big)$ and s is distinct from t is recognized by the RTA $\langle \Sigma, \{q, q_r, q_a, q_b, q_{\mathsf{f}}\}, \{q_r\}, \{q_{\mathsf{f}}\}, \Delta\rangle$ with $\Delta = \{c \to q | q_r, a(q) \to q | q_r, b(q) \to q | q_r, a(q_r) \to q_a, b(q_r) \to q_b\} \cup \big\{a(q_x) \to q_x, b(q_x) \to q_x \mid q_x \in \{q_a, q_b\}\big\} \cup \big\{\neq (q_1, q_2) \to q_{\mathsf{f}} \mid q_1, q_2 \in \{q_a, q_b, q_r\}, q_1 \neq q_2\big\}$. A successful run on $\neq \big(a(a(c)), b(a(c))\big)$ is $q_{\mathsf{f}}\big(q_a(q_r(q)), q_b(q_r(q))\big)$. The rigid state q_r will be placed at the position of the largest common postfix of s and t and q_a or q_b are used to memorize the letters immediately above this position (in order to check that s and t differ when reaching the symbol $\neq$). ◇

2.2 Pumping Lemma

We propose here a weak form, adapted to RTA, of the pumping (or iteration) lemma for TA. Pumping on runs of RTA is not as easy as for standard TA. Indeed, we must take care of the position of rigid states in order to preserve recognizability. For this reason, the transformation of a subterm must be performed in several branches in parallel (instead of one single branch for TA) in order to preserve the equality condition for rigid states. Moreover, we cannot repeat a term containing a rigid state, because the same rigid state cannot label two different positions on the same branch. The proof of the following lemma can be found in [9].

Lemma 1. *For all RTA $\mathcal{A} = \langle Q, R, F, \Delta\rangle$, for all term $t \in L(\mathcal{A})$ such that $d(t) > (|Q| + 1)|R|$, there exist a context C, two 1-contexts C' and D, with D non-trivial (non-variable), and a term u such that $t = C\big[C'[D[u]], \ldots, C'[D[u]]\big]$ and for all $n \geq 0$, $C\big[C'[D^n[u]], \ldots, C'[D^n[u]]\big] \in L(\mathcal{A})$.*

Example 5. The set $\mathcal{B}$ of balanced binary trees built over the signature $\{a : 0, f : 2\}$ is not a RTA language. Assume indeed that it is recognized by a RTA $\mathcal{A} = \langle Q, R, F, \Delta\rangle$ and let $t \in L(\mathcal{A})$ such that $d(t) > (|Q| + 1)|R|$ and C, C', D, u be as in Lemma 1. By hypothesis, $C'[D[u]]$ is balanced, but for any $n > 1$, $C'[D^n[u]]$ is not balanced since C' and D are not trivial. It contradicts $C\big[C'[D^n[u]], \ldots, C'[D^n[u]]\big] \in L(\mathcal{A})$. ◇

2.3 Related Classes of Tree Automata

We shall briefly present below some classes of extended TA and compare them to RTA. The decidability and complexity results presented in Section 3 and summarized in Table 1 also offer a base of comparison.

TAGED [4] were introduced in the context of spatial logics for XML querying [10]. They are defined, like RTA, by an underlying TA, but instead of having simply a set of rigid state for testing equality, they have two binary relations on states: $R_=$ for testing equalities and $R_\neq$ for disequalities. More precisely, a run r of a TAGED on a term t is a run of the underlying TA on t with the additional condition that for all $p_1, p_2 \in \mathcal{P}os(t)$, if $\langle r(p_1), r(p_2)\rangle \in R_=$ then $t|_{p_1} = t|_{p_2}$ and if $\langle r(p_1), r(p_2)\rangle \in R_\neq$ then $t|_{p_1} \neq t|_{p_2}$. TAGED are strictly more general than RTA. The decidability of the emptiness problem is open for the whole TAGED class. A decidable subclass of TAGED is identified in [10] where the number of equality tested in every run is bounded. The fragment of positive TAGED (with $R_\neq = \emptyset$, denoted TAGED+) has the same expressiveness as RTA. This is shown in [4] where a TAGED+ is transformed implicitely into a RTA in order to decide emptiness, at the price of an exponential blowup. The emptiness is EXPTIME-complete for TAGED+, and PTIME for RTA (see Section 3). To our knowledge, the rewrite closure of TAGED has not been studied so far.

TA with equality constraints (TAC) are TA whose transitions can perform local equality and disequality tests on the subterms of the term in input (e.g. [1,2]). In contrast, the equality tests of RTA can be global. For instance, the language of terms t over $\{f:2, g:1, a:0\}$ such that $s_1 = s_2$ for every two subterms $g(s_1)$, $g(s_2)$ of t is recognizable by a RTA, but not by a TAC. The RTA language of Examples 1 and 2 are recognizable by TAC, but not the one of Example 3. The language $\mathcal{B}$ of Example 5, which is not recognizable by RTA, is recognizable by TAC.

DAG automata (DA) [11] are defined as TA computing on DAG representation of terms with maximal sharing. Somehow, DA are the dual of RTA in the sense that in their runs, a unique state is associated to equal subtrees (which are rooted by the same node in the DAG representation) whereas for RTA, a unique subtree is associated to every occurrence of the same rigid state. However, the classes of languages defined by these two formalisms are orthogonal. On one hand, one can observe that the RTA language of Example 1 (terms $f(t,t)$) is not recognizable by a DA. On the other hand, the emptiness problem is PTIME for RTA and NP-complete for DA [12]. Actually, DA and RTA are defined for different purposes: DA are proposed for computing on compressed trees, and not for checking equalities like RTA. Moreover, deterministic DA coincide with DTA, and, as we show in Section 2.4, it is not the case for DRTA.

2.4 Determinism and Completeness

A *deterministic rigid tree automaton* (DRTA) (resp. complete RTA) on a signature Σ is a RTA $\mathcal{A}$ whose underlying TA $ta(\mathcal{A})$ is deterministic (resp. complete).

Like standard TA, every RTA can be completed into a complete RTA, by the addition of a trash state. Unlike standard TA, it is not true in general that for a complete RTA $\mathcal{A}$, for every term t there exists at least one run of $\mathcal{A}$ on t. Indeed, a given run of $ta(\mathcal{A})$ on t might not be a run of $\mathcal{A}$ on t because of the rigidity condition.

Example 6. The RTA $\mathcal{A} = \langle \{q, q_r\}, \{q_r\}, \{q\}, \{a \to q, g(q) \to q_r, g(q_r) \to q\} \rangle$, is deterministic and complete. The term $t = g(g(g(a)))$ is in $L\big(ta(\mathcal{A}), q_r\big)$, with a unique run $r = q_r(q(q_r(q)))$. However, r is not a run of $\mathcal{A}$, because the two subterms at the positions of q_r are distinct. ◇

It is well-known that DTAs are as expressive as TAs. We show below that it is not the case for RTA.

Theorem 1. *DRTA $\subsetneq$ RTA and TA $\subsetneq$ DRTA.*

Proof. We show in [9] that the language of Example 1 is not recognized by a DRTA. The inclusion TA $\subset$ DRTA is immediate since DTA $\equiv$ TA and DTA are particular cases of DRTA. Let $\Sigma = \{f{:}2, g{:}1, a{:}0\}$. The language $\big\{f(g(t), g(t)) \mid t \in \mathcal{T}(\Sigma \setminus \{g\})\big\}$ is recognized by a DRTA but not by a TA. □

2.5 Boolean Closure

We show below that the class of RTA languages is closed under union and intersection but not under complement.

Theorem 2. *Given two RTA $\mathcal{A}_1$ and $\mathcal{A}_2$, there exist two RTAs of respective sizes $O(|\mathcal{A}_1| + |\mathcal{A}_2|)$ and $O(2^{|\mathcal{A}_1||\mathcal{A}_2|})$ recognizing respectively $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ and $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.*

Proof. Let $\mathcal{A}_i = \langle Q_i, R_i, F_i, \Delta_i \rangle$ with $i = 1, 2$. For $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$, we do a classical disjoint union of automata. For $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, it is easy to construct a positive TAGED $\mathcal{B}$ recognizing $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ by a product operation like for standard TA. The state set of $\mathcal{B}$ is $Q_1 \times Q_2$, its final state set $F_1 \times F_2$ and its transition rules $\{f\big(\langle q_{11}, q_{21}\rangle, \ldots, \langle q_{1n}, q_{2n}\rangle\big) \to \langle q_1, q_2\rangle \mid q_{i1} \ldots q_{in}, q_i \in Q_i f(q_{i1}, \ldots, q_{in}) \to q_i \in \Delta_i, i = 1, 2\}$. Moreover, the equality relation of $\mathcal{B}$ is $R_= = \big\{\big\langle\langle q_{r_1}, q_2\rangle, \langle q_{r_1}, q_2'\rangle\big\rangle \mid q_{r_1} \in R_1, q_2, q_2' \in Q_2\big\} \cup \big\{\big\langle\langle q_1, q_{r_2}\rangle, \langle q_1', q_{r_2}\rangle\big\rangle \mid q_1, q_1' \in Q_1, q_{r_2} \in R_2\big\}$. A construction is proposed in [4] for transforming any positive TAGED into an RTA (i.e. a TAGED with a reflexive state relation). This transformation causes an exponential blowup. It cannot be described here. Combining the two above steps results in an exponential construction for the intersection of RTA. □

Note that the construction for the intersection of RTA preserves determinism but not for the union. The following lemma (its proof can be found in [9]) shows that the exponential time complexity for the construction of the intersection automaton constructed in Theorem 2 is a lower bound, by a reduction of the EXPTIME-complete problem of the non-emptiness of the intersection of n TA.

Lemma 2. *Given n RTA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ on Σ, we can compute in polynomial time two RTA $\mathcal{A}_\times$ and $\mathcal{A}_r$, both of size $O\big(\|\mathcal{A}_1\| + \ldots + \|\mathcal{A}_n\|\big)$, and such that $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n) = \emptyset$ iff $L(\mathcal{A}_\times) \cap L(\mathcal{A}_r) = \emptyset$.*

Theorem 3. *The class of RTA languages is not closed under complement.*

Proof. We have seen in Example 5 that the set $\mathcal{B}$ of balanced binary trees over $\Sigma := \{a : 0, f : 2\}$ is not a RTA language. We show that its complement $\overline{B}$ in $\mathcal{T}(\Sigma)$ is an RTA language. The idea is similar to the construction for the subterm relation in Example 3: one rigid state q_r is used to choose non-deterministically a subterm, and it is checked that the sibling of q_r contains q_r at depth more than one (such subterms are characterised by the state q' below). More precisely, the RTA for $\overline{B}$ is $\langle\{q, q_r, q', q_\mathsf{f}\}, \{q_r\}, \{q_\mathsf{f}\}, \Delta\rangle$ with $\Delta = \{a \to q | q_r, f(q, q) \to q | q_r, f(q, q_r) \to q', f(q_r, q) \to q', f(q, q') \to q', f(q', q) \to q', f(q_r, q') \to q_\mathsf{f}, f(q', q_r) \to q_\mathsf{f}, f(q_\mathsf{f}, q) \to q_\mathsf{f}, f(q, q_\mathsf{f}) \to q_\mathsf{f}\}$. The last two transition rules ensure the propagation of the final state q_f up to the root, like in Example 2. □

3 Decision Problems

We study in this section several decision problems for RTA; *emptiness*: given a RTA $\mathcal{A}$ on Σ, does $L(\mathcal{A}) = \emptyset$, *universality*: does $L(\mathcal{A}) = \mathcal{T}(\Sigma)$, *finiteness*: is $L(\mathcal{A})$ finite, *membership*: given additionally $t \in \mathcal{T}(\Sigma)$, is t in $L(\mathcal{A})$; *inclusion*: given two RTA $\mathcal{A}_1$ and $\mathcal{A}_2$, does $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$, *equivalence*: does $L(\mathcal{A}_1) = L(\mathcal{A}_2)$, and *intersection emptiness*: given n RTA $\mathcal{A}_1, \ldots \mathcal{A}_n$, does $L(\mathcal{A}_1) \cap \ldots \cap L(\mathcal{A}_n) = \emptyset$. Table 1 provides a summary of closure and decision results and a comparison with other classes of extended TA mentioned in Section 2.3.

Table 1. Summary of closure and decision results

	TA	**RTA**	**TAGED+**	**DA**
$\cup$	PTIME	PTIME	PTIME	PTIME
$\cap$	PTIME	EXPTIME	EXPTIME	not [12]
$\neg$	EXPTIME	not	not	not
emptiness	linear-time	linear-time	EXPTIME-complete	NP-complete
membership	PTIME	NP-complete	NP-complete	NP-complete
$\cap$-emptiness	EXPTIME-complete	EXPTIME-complete	EXPTIME-complete	
universality	EXPTIME-complete	undecidable	undecidable	undecidable
inclusion	EXPTIME-complete	undecidable	undecidable	undecidable
finiteness	PTIME	PTIME		

Theorem 4. *The emptiness problem is decidable in linear time for RTA.*

Proof. We show [9] that the emptiness of $L(\mathcal{A})$ and $L\big(ta(\mathcal{A})\big)$ are equivalent. The latter problem (emptiness for standard TA) is known to be decidable in linear-time (see e.g. [13]). The idea is that if $L(ta(\mathcal{A}))$ is not empty, then the classical "state marking" algorithm builts a witness which respects the rigidity condition for all states, and is therefore a witness for $L(\mathcal{A})$ non-emptiness. □

Theorem 5. *Membership is* NP*-complete for RTA (*PTIME *for DRTA).*

Proof. A non-deterministic algorithm for this problem consist in, given a RTA $\mathcal{A}$ and a term t, guessing a labelling of the nodes of t with states of $\mathcal{A}$ and checking

that this labelling is a run of $\mathcal{A}$ on t. The checking operation can be performed in polynomial time. In the deterministic case, there is at most one labelling of the term t compatible with the transition rules.

In order to show NP-hardness, we propose [9] a reduction from 3-SAT for a formula ϕ into the membership to an RTA $\mathcal{A}$ of a term t_ϕ representing ϕ. Each variable x of ϕ is represented in t_ϕ by a subterm $x(0,1)$, where x is a binary symbol and $0, 1$ constants. The most important transitions of $\mathcal{A}$ are $0, 1 \to q_x|q_{\neg x}$ for each variable x of ϕ and $x(q_x, q_{\neg x}) \to q_0$, $x(q_{\neg x}, q_x) \to q_1$, where the states q_x and $q_{\neg x}$ are rigid. The states q_0 and q_1 represent the value associated to x (they are propagated bottom-up along t_ϕ) and the rigidity condition ensures that the same value is associated to all occurrences of the variable x in ϕ. □

Theorem 6. *Intersection non-emptiness is* EXPTIME*-complete for RTA.*

Proof. The upper-bound is a consequence of Lemma 2 and Th. 2 & 4. The lower-bound follows from the EXPTIME-hardness of the problem for TA [14]. □

Theorem 7. *Universality is undecidable for RTA.*

Proof. In [9] we reduce the non-existence of a solution of an instance P of the Post Correspondence Problem to the universality of a RTA. This RTA recognizes the set of terms which do not represent a solution of P. It is defined as a disjoint union of RTAs, one for each case. Some cases involve the construction of a RTA testing disequalities between unary subterms like in Example 4. □

Theorem 8. *Inclusion and equivalence are undecidable for RTA.*

Proof. The equivalence problem is reducible to inclusion. Hence both are undecidable as universality is a particular case of equivalence. □

For an RTA $\mathcal{A}$, the finiteness of $L(ta(\mathcal{A}))$ implies the finiteness of $L(\mathcal{A})$, but the converse is not true: the language of the RTA of Example 6 is $\{a, g(g(a))\}$ whereas the language of its underlying TA is $\{a, g^2(a), g^4(a), \ldots\}$.

Theorem 9. *Finiteness is decidable in* PTIME *for RTA.*

Like for TA, checking finiteness amounts to detect (in PTIME) some loops and paths in the accessibility graph of an RTA (see [9] for details).

4 Rewrite Closure

The closure of a RTA language under rewriting is unfortunately not a RTA language, even for a TRS as simple as $\mathcal{R} = \{f(g(x)) \to x\}$. Let $\Sigma = \{h : 2, f : 1, g : 1, 0 : 0\}$, and let $\mathcal{A} = \langle Q, R, \Delta \rangle$ be the RTA on Σ with $Q = \{q_0, q_1, q_2, q_r, q_\mathsf{f}\}$, $R = \{q_r\}$, $F = \{q_\mathsf{f}\}$, and $\Delta = \{0 \to q_0, g(q_0) \to q_0|q_r, f(q_r) \to q_1, f(q_1) \to q_1, h(q_r, q_{1,2}) \to q_\mathsf{f}, h(q_{1,2}, q_{1,2}) \to q_2, h(q_\mathsf{f}, q_{1,2}) \to q_\mathsf{f}, \}$ where $q_{1,2}$ is either q_1 or q_2. Every term of $L(\mathcal{A})$ has the form $H\left[g^m(0), f^*(g^m(0)), \ldots, f^*(g^m(0))\right]$ where H is a k-context made of the symbol h only, and g^m and f^* represent nesting of m symbol g and an arbitrary number of f, respectively. The rigid state q_r

enforces that each argument has the same number of g. The terms of the closure $\mathcal{R}^*(L(\mathcal{A}))$ of $L(\mathcal{A})$ by $\mathcal{R}$ have a similar form except that the number of g in the different arguments might not be equal. They only have to be all less than or equal to the number of g on the leftmost argument. We show in [9] that it is not a RTA language, with arguments similar to those of Section 2.2. The rewrite closure of a RTA under a linear collapsing TRS is even not recursive.

Theorem 10. *The problem to know whether $t \in \mathcal{R}^*(L(\mathcal{A}))$ or not given a RTA $\mathcal{A}$, a collapsing and linear TRS $\mathcal{R}$ and a term t, is undecidable.*

Proof. Let $u_1, v_1 \dots u_n, v_n$ be words on an alphabet Γ seen as a PCP instance P. Let us consider the signature $\Sigma = \{g_i : 1, f_i : 1 \mid i \leq n\} \cup \{a : 1 \mid a \in \Gamma\} \cup \{0 : 0, k : 1, h : 2\}$, and $L = \{h(s, k(s)) \mid s = f_{i_1}(g_{i_1}(\dots f_{i_m}(g_{i_m}(w(0))))), m > 0, w \in \Gamma^*\}$[1] where $1 \leq i_1, \dots, i_m \leq n$. Let $\mathcal{R}$ be a TRS on Σ with the rules $f_i(g_i(u_i(x))) \to x$ $(i \leq n)$, $g_i(x) \to x$ $(i \leq n)$, $g_j(f_i(v_i(x))) \to x$ $(i, j \leq n)$, and $k(f_i(v_i(x))) \to x$ $(i \leq n)$. The tree language L is recognizable by a RTA on Σ and we show [9] that $h(0, 0) \in \mathcal{R}^*(L)$ iff P has a solution. □

The problem of Theorem 10, *membership modulo*, becomes decidable with some further syntactical restrictions on $\mathcal{R}$ based on the theory of visibly pushdown automata (VPA) [15]. VPA define a subset of context-free languages closed under intersection, and were generalized to tree recognizers in [16,17]. The idea in these works is that the signature Σ is partitioned into $\Sigma = \Sigma_c \uplus \Sigma_r \uplus \Sigma_\ell$ and the operation performed by the VPA on the stack depends on the current symbol in input: if it is a *call* symbol of Σ_c, the VPA can only do a push, for a *return* symbol of Σ_r it can do a pop and it must leave the stack untouched for a *local* symbol of Σ_ℓ.

In [16], Chabin and Rety show that the class of visibly pushdown tree automata (VPTA) languages is closed under rewriting with so called linear context-free visibly TRS. We use a similar definition in order to characterize a class of TRS for which membership modulo is decidable.

Definition 2. *A collapsing TRS $\mathcal{R}$ is called* inverse-visibly *(invisibly) if for every rule $\ell \to x \in \mathcal{R}$, $d(\ell) \geq 1$, x occurs once in ℓ, and if x occurs at depth 1 in ℓ then $\ell \in \mathcal{T}(\Sigma_\ell, \mathcal{X})$, otherwise, $\ell(\varepsilon) \in \Sigma_c$, the symbol immediately above x is in Σ_r and all the other symbols of ℓ are in Σ_ℓ.*

Example 7. The TRS $\mathcal{R} = \{\mathsf{fst}(\mathsf{pair}(x_1, x_2)) \to x_1, \mathsf{snd}(\mathsf{pair}(x_1, x_2)) \to x_2, \mathsf{decrypt}(\mathsf{encrypt}(x, \mathsf{pk}(\mathsf{A})), \mathsf{sk}(\mathsf{A})) \to x\}$ is linear and invisibly with $\Sigma_c = \{\mathsf{fst}, \mathsf{snd}, \mathsf{decrypt}\}$ and $\Sigma_r = \{\mathsf{pair}, \mathsf{encrypt}\}$, $\Sigma_\ell = \{\mathsf{pk}, \mathsf{sk}, \mathsf{A}\}$. ◇

The TRS $\{f(g(x)) \to x\}$ is invisibly but not the one for Theorem 10.

Theorem 11. *The problem to know whether $t \in \mathcal{R}^*(L(\mathcal{A}))$ or not, given a RTA $\mathcal{A}$, a linear and invisibly TRS $\mathcal{R}$ and a term t, is decidable.*

Proof. The proof [9] is long and technical, and due to space restrictions, we only sketch it below for a TRS $\mathcal{R}$ containing the two first rewrite rules of Example 7.

[1] For all $w = a_1, \dots, a_p \in \Gamma^*$, the term $a_1(\dots a_p(t))$ is written $w(t)$.

Let $\mathcal{A} = \langle Q, R, F, \Delta \rangle$ be a RTA on $\Sigma = \{\mathsf{f} : 2, \mathsf{fst} : 1, \mathsf{snd} : 1, \mathsf{pair} : 2, 0 : 0\}$ with $Q = \{q_0, q_r, q_1, q_\mathsf{f}\}$, $R = \{q_r\}$, $F = \{q_\mathsf{f}\}$, and $\Delta = \{0 \to q_0, \mathsf{pair}(q_0, q_0) \to q_0 | q_r, \mathsf{fst}(q_r | q_1) \to q_1, \mathsf{snd}(q_r | q_1) \to q_1, \mathsf{f}(q_1, q_1) \to q_\mathsf{f}\}$ and let $t = \mathsf{f}(\mathsf{pair}(0, 0), 0)$. Very roughly, the decision algorithm guesses the existence of one tree $t' \in L(\mathcal{A})$ such that $t' \xrightarrow[\mathcal{R}]{*} t$, by application of $\mathcal{R}$ backwards starting from t, expanding subterms into lhs of rules. In order to ensure that $t' \in L(\mathcal{A})$, we consider pairs of states $\frac{q_\varepsilon}{q_x}$ which intuitively correspond to a run r of $\mathcal{A}$ on ℓ (for $\ell \to x \in \mathcal{R}$) such that $q_\varepsilon = r(\varepsilon)$ and $q_x = r(p_x)$ where $\ell(p_x) = x$ (this position p_x is unique by hypothesis). If $q_\varepsilon = q_x$, the pair is simply denoted q_ε. In a first step, we label the lhs of $\mathcal{R}$ with such pairs. For both $\mathsf{fst}(\mathsf{pair}(x_1, x_2))$ and $\mathsf{snd}(\mathsf{pair}(x_1, x_2))$, the only possible labelling is $\ell_1 := q_1(\frac{q_1}{q_0}(q_0, q_0))$. The condition for such a labelling is indeed that there exists a transition in $\mathcal{A}$ from the first components of labels at sibling positions into the second component of the label at the father position, like $\mathsf{fst}(q_1) \to q_1$ and $\mathsf{pair}(q_0, q_0) \to q_0$ for ℓ_1 above. Intuitively, ℓ_1 describes a nesting of runs on lhs of $\mathcal{R}$ which permits to recover a run r' of $\mathcal{A}'$ on t'. In other terms, t' can be generated by a context-free tree grammar with non-terminals from Q (nullary) or of the form $\frac{q_\varepsilon}{q_x}$ (unary). For instance, the production rule $\frac{q_1}{q_0}(x) := \mathsf{fst}(\frac{q_1}{q_0}(\mathsf{pair}(x, q_0))$ corresponds to $\mathsf{fst}(\mathsf{pair}(x_1, x_2)) \to x_1$. The grammar generates a visibly pushdown tree language, thanks to the hypotheses on $\mathcal{R}$.

Next, in order to guess t', we label the positions of t by pairs of states. We obtain $q_\mathsf{f}(p(q_0, q_0), \frac{q_1}{q_0})$ where p is either q_r or $\frac{q_1}{q_0}$. We can observe that the rigid state q_r occurs, possibly at nested depth bigger than one, in both cases for p. The tricky part of the algorithm it to check that there exists at least one term in the intersection of the languages (generated by grammars as above) corresponding to the distinct occurrences of q_r. We use the fact that the emptiness of intersection is decidable for visibly context free tree grammars. □

5 Conclusion and Further Work

We want to use RTA for the automatic verification of traces or equivalence properties of security protocols, using regular tree model checking like techniques. In this context, we are planning to extend the result of Theorem 11 to invisibly (non-linear) TRS, in order to handle axioms as $\mathsf{decrypt}(\mathsf{encrypt}(x, y), y) = x$. We are also interested about the symmetric form of the TRS of [16], whose rhs are not single variables but have the form $f(x_1, \ldots, x_n)$.

One may also study the extension of RTA to equality tests modulo equational theories like in [8], or the addition of disequality constraints in order to obtain closure under complement and correspondence with logics.

References

1. Bogaert, B., Tison, S.: Equality and Disequality Constraints on Direct Subterms in Tree Automata. In: Finkel, A., Jantzen, M. (eds.) STACS 1992. LNCS, vol. 577, pp. 161–171. Springer, Heidelberg (1992)
2. Dauchet, M., Caron, A.C., Coquidé, J.L.: Automata for Reduction Properties Solving. Journal of Symbolic Computation 20(2), 215–233 (1995)

3. Comon, H., Cortier, V.: Tree automata with one memory, set constraints and cryptographic protocols. Theoretical Computer Science 331(1), 143–214 (2005)
4. Filiot, E., Talbot, J.M., Tison, S.: Tree automata with global constraints. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 314–326. Springer, Heidelberg (2008)
5. Bouajjani, A., Touili, T.: On computing reachability sets of process rewrite systems. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 484–499. Springer, Heidelberg (2005)
6. Genet, T., Klay, F.: Rewriting for Cryptographic Protocol Verification. In: McAllester, D. (ed.) CADE 2000. LNCS, vol. 1831. Springer, Heidelberg (2000)
7. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL, pp. 104–115 (2001)
8. Jacquemard, F., Rusinowitch, M., Vigneron, L.: Tree automata with equality constraints modulo equational theories. Journal of Logic and Algebraic Programming 75(2), 182–208 (2008)
9. Jacquemard, F., Klay, F., Vacher, C.: Rigid tree automata. Technical Report RRLSV-0827, Laboratoire Spécification et Vérification (2008)
10. Filiot, E., Talbot, J.M., Tison, S.: Satisfiability of a spatial logic with tree variables. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 130–145. Springer, Heidelberg (2007)
11. Charatonik, W.: Automata on dag representations of finite trees. Technical Report Technical Report MPI-I-99-2-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany (1999)
12. Anantharaman, S., Narendran, P., Rusinowitch, M.: Closure properties and decision problems of dag automata. Inf. Process. Lett. 94(5), 231–240 (2005)
13. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Löding, C., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2007), http://tata.gforge.inria.fr
14. Seidl, H.: Haskell overloading is DEXPTIME-complete. Information Processing Letters 52(2), 57–60 (1994)
15. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: 36th Annual ACM Symposium on Theory of Computing, STOC, pp. 202–211. ACM, New York (2004)
16. Chabin, J., Réty, P.: Visibly pushdown languages and term rewriting. In: Konev, B., Wolter, F. (eds.) FroCos 2007. LNCS, vol. 4720, pp. 252–266. Springer, Heidelberg (2007)
17. Comon-Lundh, H., Jacquemard, F., Perrin, N.: Tree automata with memory, visibility and structural constraints. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 168–182. Springer, Heidelberg (2007)

Converting Self-verifying Automata into Deterministic Automata

Galina Jirásková[1,⋆] and Giovanni Pighizzini[2,⋆⋆]

[1] Mathematical Institute, Slovak Academy of Sciences,
Grešákova 6, 040 01 Košice, Slovakia
jiraskov@saske.sk
[2] Dipartimento di Informatica e Comunicazione,
Università degli Studi di Milano
via Comelico 39, I-20135 Milano, Italy
pighizzini@dico.unimi.it

Abstract. Self-verifying automata are a special variant of finite automata with a symmetric kind of nondeterminism. In this paper, we study the transformation of self-verifying automata into deterministic automata from a descriptional complexity point of view. The main result is the exact cost, in terms of the number of states, of such a simulation.

1 Introduction

In automata theory and theoretical computer science, several kinds of devices able to recognize formal languages have been proposed and investigated.

Different classes of devices can be compared, first of all, from the point of view of their recognition powers. We mention just two examples of classical results in this area: the equivalence between deterministic and nondeterministic finite automata, and the fact that deterministic pushdown automata are strictly less powerful than nondeterministic ones. With a deeper investigation, classes of devices can be compared from the point of view of their descriptional complexity [1]. The classical example is the simulation of n-state nondeterministic automata by deterministic automata that can be done using 2^n states [2], and cannot be done, in the worst case, with less than 2^n states [3,4,5].

In this paper, we continue this line of research, by considering *self-verifying automata*, a special kind of finite automata, introduced in [6], with a symmetric form of nondeterminism, called *self-verifying nondeterminism* [7]. This kind of nondeterminism was mainly considered in connection with randomized Las Vegas computations, but as pointed out in [8], it is interesting also *per se*.

Roughly speaking, in self-verifying nondeterminism, computation paths can give three types of answers: *yes*, *no*, and *I do not know*. On each input string,

⋆ Supported by VEGA grant 2/0111/09.
⋆⋆ Partially supported by MIUR under the project PRIN "Aspetti matematici e applicazioni emergenti degli automi e dei linguaggi formali: metodi probabilistici e combinatori in ambito di linguaggi formali".

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 458–468, 2009.

at least one path must give answer "yes" or "no". Furthermore, on the same string, two paths cannot give contradictory answers, namely both the answers "yes" and "no" are not possible.

It is not difficult to observe that self-verifying automata are as powerful as deterministic automata. In particular, the standard subset construction can be used to convert them to deterministic automata. Hence, the question arises of investigating this equivalence from the descriptional point of view.

This problem was previously considered by Assent and Seibert in [9], who proved that in the deterministic automaton obtained by applying the standard subset construction to a self-verifying automaton certain states must be equivalent. As a consequence, they were able to show that each n-state self-verifying automaton can be simulated by a deterministic automaton with $O(\frac{2^n}{\sqrt{n}})$ states.

In this paper, we further deepen this investigation by showing that such an upper bound can be lowered to a function $g(n)$ which grows like $3^{\frac{n}{3}}$ (we give the *exact* value of $g(n)$ in the paper). In particular, we associate with each n-state self-verifying automaton A a certain graph with n vertices, and we prove that there exists a deterministic automaton equivalent to A whose state set is isomorphic to the set of the maximal cliques of such a graph. Using a result from graph theory stating the number of possible maximal cliques in a graph [10], we get the upper bound $g(n)$.

In the second part of the paper, we prove the optimality of such an upper bound. In fact, we are able to show that for each positive integer n, there exists a binary language L accepted by an n-state self-verifying automaton, such that the minimal equivalent deterministic automaton must have *exactly* $g(n)$ states.

We conclude the paper with some considerations concerning the cases of automata defined over a one-letter alphabet and of automata with multiple initial states.

2 Preliminaries

We fix an alphabet Σ. Given a language $L \subseteq \Sigma^*$, we denote by L^c the complement of L, namely the set $\Sigma^* - L$.

We assume that the reader is familiar with the notions of deterministic and nondeterministic finite automata (denoted respectively as dfa's and nfa's, for short).

Definition 1. *A* self-verifying finite automaton *(svfa) A is a 6-tuple $(Q, \Sigma, \delta, q_0, F^a, F^r)$, where Q, Σ, δ, q_0 are defined as for standard nondeterministic automata, $F^a, F^r \subseteq Q$ are, respectively, the sets of* accepting *and* rejecting *states, while the remaining states, namely the states belonging to $Q - (F^a \cup F^r)$, are called* neutral *states.*

It is required that for each input string w in Σ^*, there exists at least one computation ending in an accepting or in a rejecting state, that is, $\delta(q_0, w) \cap (F^a \cup F^r) \neq \emptyset$, and there are no strings w such that both $\delta(q_0, w) \cap F^a$ and $\delta(q_0, w) \cap F^r$ are nonempty.

The language accepted by A, denoted as $L^a(A)$, is the set of all input strings having a computation ending in an accepting state, while the language rejected by A, denoted as $L^r(A)$, is the set of all input strings having a computation ending in a rejecting state.

It follows directly from the definition that $L^a(A) = (L^r(A))^c$ for each svfa A. Hence, when we will say that an svfa A *recognizes* a language L, we will mean that $L = L^a(A)$ and $L^c = L^r(A)$.

From an svfa A recognizing a language L, we can immediately get two nfa's accepting, respectively, L and L^c. These automata are defined like the svfa A, except that the sets of final states are replaced, respectively, by F^a and F^r. Hence, svfa's can be seen as a special case of nfa's.

On the other hand, having two nfa's accepting languages L and L^c, respectively, we can construct an svfa for the language L by adding a new initial state connected via ε-transitions to the initial states of the two nfa's. The set of accepting states of this svfa consists of all final states of the nfa for L, while the set of rejecting states contains all final states of the nfa for L^c.

The above observations give the following relationships between the sizes of svfa's and nfa's recognizing the same language.

Theorem 1 ([8, Observation 4.2]). *For each regular language L, let* $\mathrm{ns}(L)$ *and* $\mathrm{svs}(L)$ *denote, respectively, the minimum number of states of any nfa and of any svfa recognizing L. Then* $\max\{\mathrm{ns}(L), \mathrm{ns}(L^c)\} \leqslant \mathrm{svs}(L) \leqslant 1+\mathrm{ns}(L)+\mathrm{ns}(L^c)$.

Let $G = (V, E)$ be an undirected graph. We recall that each complete subgraph of G is called a *clique*. We also say that a subset $\alpha \subseteq V$ forms a clique if the subgraph of G induced by α, namely the graph $(\alpha, E \cap (\alpha \times \alpha))$, is a clique. Furthermore, a clique $\alpha \subseteq V$ is *maximal* if each subset properly containing α does not form a clique. In [10], Moon and Moser stated the following exact bound for the number of maximal cliques in a graph.

Theorem 2 (Moon, Moser [10]). *Let $f(n)$ denotes the maximum number of possible maximal cliques in a graph with n nodes. If $n \geqslant 2$, then*

$$f(n) = \begin{cases} 3^{\lfloor \frac{n}{3} \rfloor}, & \text{if } n \equiv 0 \pmod 3, \\ 4 \cdot 3^{\lfloor \frac{n}{3} \rfloor - 1}, & \text{if } n \equiv 1 \pmod 3, \\ 2 \cdot 3^{\lfloor \frac{n}{3} \rfloor}, & \text{if } n \equiv 2 \pmod 3. \end{cases}$$

3 Conversion of svfa's into dfa's

Self-verifying automata are a special case of nondeterministic automata. Hence, the standard simulation of nfa's by dfa's given by the subset construction can be used also to simulate svfa's. In this section, by investigating some properties of svfa's and of the corresponding subset automata, we will show how it is possible to reduce the number of states of resulting dfa's. More precisely, the main result of this section is an upper bound on the number of states of minimal dfa's equivalent to n-state svfa's.

Throughout the section, consider a fixed svfa $A = (Q, \Sigma, \delta, q_0, F^a, F^r)$ with n states. Given a state $q \in Q$, we denote by L^a_q and L^r_q, respectively, the set of strings accepted and the set of strings rejected starting from q, that is, $L^a_q = \{x \in \Sigma^* \mid \delta(q, x) \cap F^a \neq \emptyset\}$ and $L^r_q = \{x \in \Sigma^* \mid \delta(q, x) \cap F^r \neq \emptyset\}$. We observe that as a consequence of the definition of svfa's, the following statements hold for the initial state q_0:

- $L^a_{q_0} \cup L^r_{q_0} = \Sigma^*$, namely the automaton A must give an answer on each string (completeness).
- $L^a_{q_0} \cap L^r_{q_0} = \emptyset$, namely the automaton A cannot give two contradictory answers on the same string (consistency).

Note that the conjunction of the two statements is equivalent to $L^a_{q_0} = (L^r_{q_0})^c$.

If $q \in Q$ is a reachable state of A, namely there exists a string x in Σ^* such that $q \in \delta(q_0, x)$, then $L^a_q \cap L^r_q = \emptyset$. Otherwise, if q is not reachable, it is possible that the languages L^a_q and L^r_q are not disjoint. However, we can remove all unreachable states from A, without affecting the accepted and the rejected languages. Hence, in the following we assume that *each state q of A is reachable* and so $L^a_q \cap L^r_q = \emptyset$.

Applying the standard subset construction to the svfa A, we get an equivalent dfa. Such a dfa, *restricted to its reachable states*, i.e., to states $\alpha \subseteq Q$ such that there is at least one string $x \in \Sigma^*$ with $\delta(q_0, x) = \alpha$, will be called in the following the *subset automaton* associated with A.

While for a state $q \neq q_0$ of an svfa it can happen that starting from q and reading a string x neither accepting nor rejecting states are reachable, that is, $x \notin L^a_q \cup L^r_q$, for the states of the subset automaton we can prove a completeness condition. In particular, for each $\alpha \subseteq Q$, we consider the languages accepted and rejected starting from α given by $L^a_\alpha = \bigcup_{q \in \alpha} L^a_q$ and $L^r_\alpha = \bigcup_{q \in \alpha} L^r_q$, and we prove the following lemma.

Lemma 1. *Let $\alpha \subseteq Q$ be a state of the subset automaton. Then $L^a_\alpha \cap L^r_\alpha = \emptyset$ and $L^a_\alpha \cup L^r_\alpha = \Sigma^*$, i.e., $L^a_\alpha = (L^r_\alpha)^c$.*

Proof. Since α is a state of the subset automaton, it must be reachable. Let x be a string in Σ^* such that $\delta(q_0, x) = \alpha$. Suppose that $y \in L^a_\alpha \cap L^r_\alpha$. Then $\delta(q_0, xy) \cap F^a \neq \emptyset$ and $\delta(q_0, xy) \cap F^r \neq \emptyset$, which is a contradiction.

In a similar way, if $y \notin L^a_\alpha \cup L^r_\alpha$, then we get that $xy \notin L^a_{q_0} \cup L^r_{q_0}$, which is a contradiction. □

In order to state an upper bound on the number of states of the minimal dfa equivalent to A, it is useful to count how many reachable subsets of Q, representing different languages, are possible, or to find equivalence conditions for the states of the subset automaton.

In [9] it was shown that given two reachable states $\alpha, \beta \subseteq Q$ of the subset automaton, $\alpha \subseteq \beta$ implies that α and β are equivalent. In light of Lemma 1, this is due to the fact that $\alpha \subseteq \beta$ implies that $L^a_\alpha \subseteq L^a_\beta$ and $L^r_\alpha \subseteq L^r_\beta$, which combined with the completeness conditions of the lemma, gives $L^a_\alpha = L^a_\beta$ and

$L^r_\alpha = L^r_\beta$. In the following, we will show that actually in the subset automaton the state equivalence is implied by a weaker condition.

To this aim, we introduce a *compatibility relation* on the state set Q. Intuitively, two states $q, p \in Q$ are compatible if and only if two computations starting from q and p cannot give contradictory answers on the same string. We now define this notion formally.

Definition 2. *Two states $q, p \in Q$ are* compatible *if and only if* $(L^a_q \cup L^a_p) \cap (L^r_q \cup L^r_p) = \emptyset$.

The compatibility graph *of A is the undirected graph whose vertex set is Q, and which contains an edge $\{q, p\}$ if and only if states q and p are compatible.*

By the above discussion, it is immediate to observe that if α is a state of the subset automaton, then all $q, p \in \alpha$ must be compatible. Hence, each reachable state of the subset automaton is represented by a clique in the compatibility graph. The following lemma allows us to restrict our attention only to *maximal* cliques.

Lemma 2. *Let $\alpha, \beta \subseteq Q$ be two states of the subset automaton such that $\alpha \cup \beta$ is a clique in the compatibility graph of the svfa A. Then α and β are equivalent.*

Proof. Before proving the lemma, we notice that at the first glance the statement seems to be very close to Claim 3.2 in [9]. However, that claim does not apply when $\alpha \cup \beta$ is not a reachable state of the subset automaton.

By contradiction, assume that there is a string x in Σ^* which is accepted by the subset automaton from state α and rejected from state β. It follows that there is a state q in α and a state p in β such that the string x is accepted by the svfa A from state q and rejected from state p. This means that states q and p are not compatible, which contradicts our assumption that $\alpha \cup \beta$ is a clique in the compatibility graph of A. □

Now we are able to state our upper bound.

Theorem 3. *For each svfa A with n states, there exists an equivalent dfa with at most $1 + f(n-1)$ states, where $f(n)$ is the maximum number of possible maximal cliques in a graph with n nodes.*

Proof. We have already observed that each state of the subset automaton associated with A is a clique in the compatibility graph of A. Furthermore, by Lemma 2, if two states form cliques that are contained in the same maximal clique, then they are equivalent. Hence, the subset automaton can be reduced to an equivalent dfa with at most one state for each maximal clique of the compatibility graph.

To complete the proof, we count the number of such cliques.

First of all, we observe that since $L^a_{q_0} = (L^r_{q_0})^c$, two states q and p that are compatible with the initial state q_0 are compatible with each other. Hence, in the compatibility graph there is exactly one maximal clique containing state q_0, while the other maximal cliques can involve the remaining $n-1$ states. Hence, using Theorem 2, we get that the minimal dfa equivalent to A can have at most $1 + f(n-1)$ states. □

4 Optimality

In this section, we study the optimality of the simulation of svfa's by dfa's presented in Section 3. We prove that for each n, there exists a binary svfa A_n with n states whose equivalent minimal dfa has $1 + f(n-1)$ states, showing in this way a lower bound exactly matching the upper bound stated in Theorem 3.

For the sake of simplicity, let us start by considering the case $n = 1 + 3m$, with $m \geqslant 2$. In the last part of the section, we will discuss how to extend our argument to the other values of n.

Let $A_n = (Q, \{a, b\}, \delta, q_0, F^a, F^r)$ be the automaton depicted in Figure 1 and defined as follows:

- $Q = \{q_0\} \cup \{(i, j) \mid 0 \leqslant i \leqslant 2, 1 \leqslant j \leqslant m\}$,
- $\delta(q_0, a) = \delta(q_0, b) = \{(0, 1), (0, 2), \ldots, (0, m)\}$,
  and for all i, j with $0 \leqslant i \leqslant 2$ and $1 \leqslant j \leqslant m$,
  $\delta((i, j), a) = \begin{cases} \{(i, j+1)\}, & \text{if } j < m, \\ \{(0, 1)\}, & \text{otherwise,} \end{cases}$
  $\delta((i, j), b) = \{((i+1) \bmod 3, j)\}$,
- $F^a = \{q_0, (0, m)\}$, $F^r = \{(1, m), (2, m)\}$.

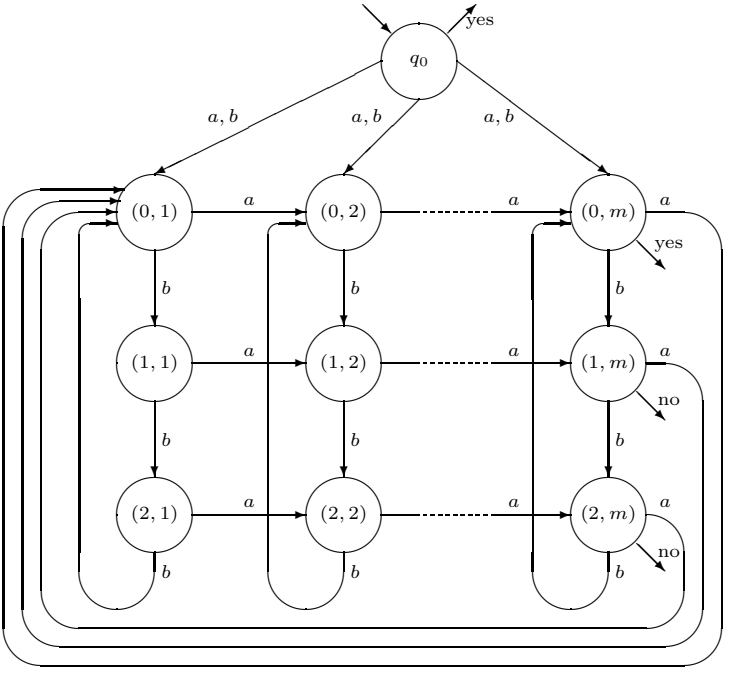


Fig. 1. The self-verifying finite automaton A_n

We claim that A_n is self-verifying. To prove this, and also for the subsequent discussion, it is useful to consider the following family Q' of subsets of Q:

$$Q' = \{\{(x_1, 1), \ldots, (x_m, m)\} \mid x_1, \ldots, x_m \in \{0, 1, 2\}\}.$$

Note that in Figure 1, the states other than q_0 are represented by a grid, according to the two components in their names. Each set belonging to Q' corresponds to the choice of one element in each column of such a grid. Notice that after applying a transition by the same letter to all the states in a set belonging to Q', we reach a set of states which still belongs to Q'. Extending this argument to strings we get the following remark.

Remark 1. For each $\alpha \in Q'$ and each $w \in \Sigma^*$, the set $\delta(\alpha, w)$ belongs to Q'.

Furthermore, from the initial state (accepting the empty string), by reading a or b, we reach the set $\{(0, 1), \ldots, (0, m)\}$ belonging to Q'. Hence, for each $x \in \Sigma^+$, the set $\delta(q_0, x)$ contains exactly one of the final states $(0, m), (1, m), (2, m)$. Thus, for each string, the automaton reaches a final state, and it cannot give two different answers on the same string. This permits us of concluding that A_n is self-verifying.

We now closely study the properties of the computations of the svfa A_n. As a consequence of the previous discussion and of Remark 1, it turns out that in the subset automaton associated with A_n, all states that can be reached besides the initial state $\{q_0\}$ belong to Q'. We will show that *all* the elements of Q' are reachable.

In the following, an element $\{(x_1, 1), \ldots, (x_m, m)\}$ belonging to Q' will be denoted simply as the vector $(x_1, \ldots, x_m)$ in $\{0, 1, 2\}^m$. With a small abuse of notation, for $(x_1, \ldots, x_m)$ and $(y_1, \ldots, y_m)$ in $\{0, 1, 2\}^m$, we will write $(y_1, \ldots, y_m) = \delta((x_1, \ldots, x_m), w)$, if by reading the string $w \in \Sigma^*$ from the state set corresponding to the vector $(x_1, \ldots, x_m)$, the state set corresponding to the vector $(y_1, \ldots, y_m)$ is reached.

Hence, for each $(x_1, \ldots, x_m)$ in $\{0, 1, 2\}^m$, the following holds:

- $\delta((x_1, \ldots, x_m), a) = (0, x_1, \ldots, x_{m-1})$,
- $\delta((x_1, \ldots, x_m), b) = ((x_1 + 1) \bmod 3, \ldots, (x_m + 1) \bmod 3)$.

From the first equality, we can see that the string a^m can be used to "reset" the "grid part" of the automaton, that is, $\delta((x_1, \ldots, x_m), a^m) = (0, \ldots, 0)$ for each $(x_1, \ldots, x_m)$ in $\{0, 1, 2\}^m$. Hence, the state $(0, \ldots, 0)$ is reachable from each state belonging to Q'. We now prove the converse.

Lemma 3. *For each $(z_1, \ldots, z_m)$ in $\{0, 1, 2\}^m$, the state set corresponding to the vector $(z_1, \ldots, z_m)$ is reachable.*

Proof. Notice that $(z_1, \ldots, z_m) = \delta((0, z_2 - z_1, \ldots, z_m - z_1), b^{z_1})$, where subtraction is modulo 3. So, it is enough to prove that all states corresponding to vectors $(0, y_1, \ldots, y_{m-1})$ with y_i in $\{0, 1, 2\}$ are reachable.

Let $y = y_1 \cdots y_k$ be a string over $\{0,1,2\}$ of length k, $0 \leqslant k \leqslant m-1$, and let us show by induction on k that for each x in $\{0,1,2\}$, the state corresponding to the vector $(0, y_1, \ldots, y_k, x, x, \ldots, x)$ is reachable.

The basis, $k = 0$, holds true since for each x, state $(0, x, x, \ldots, x)$ can be reached from state $(0, \ldots, 0)$ by $b^x a$. For the induction step, let $y = y_1 \cdots y_k$ be any string of length k. Then state $(0, y_2 - y_1, \ldots, y_k - y_1, x - y_1, \ldots, x - y_1)$, with subtraction modulo 3, is reachable by induction hypothesis, and it goes to state $(0, y_1, \ldots, y_k, x, \ldots, x)$ by the string $b^{y_1} a$. Hence state $(0, y_1, \ldots, y_k, x, \ldots, x)$ is reachable, which completes our proof. □

As a consequence of Remark 1 and Lemma 3, it turns out that in the subset automaton A_{sub} associated with the svfa A_n, the set of reachable states coincides with $\{q_0\} \cup Q'$. Hence, its cardinality is $1 + 3^m = 1 + 3^{\frac{n-1}{3}}$. Now we prove that the automaton A_{sub} is minimal.

To this aim, given two different states $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_m)$ in Q', we consider an index j, $1 \leqslant j \leqslant m$, such that $x_j \neq y_j$. We remind that $x_j, y_j \in \{0,1,2\}$. The states $(0,j), (1,j)$ of the svfa A_n are not compatible because, for instance, the string a^{m-j} is accepted by a computation starting from $(0,j)$, but rejected by a computation starting from $(1,j)$. The same string can be used to show that even the states $(0,j)$ and $(2,j)$ are not compatible, while the string $a^{m-j}b$ can be used to prove that the states $(1,j)$ and $(2,j)$ are not compatible. Hence, in all possible cases we conclude that there exists a string z witnessing the fact that the states (x_j, j) and (y_j, j) of A_n are not compatible. By the standard properties of subset automata, it follows that the same string z distinguishes x and y. Since the initial state $\{q_0\}$ of the subset automaton is accepting, it cannot be equivalent to any subset of Q' that contains either state $(1,m)$ or state $(2,m)$. A subset of Q' that contains state $(0,m)$ is distinguishable with $\{q_0\}$ by the string b. Hence, we get the following result.

Lemma 4. *The minimal dfa equivalent to the given n-state svfa A_n is the subset automaton A_{sub} associated with A_n and it has $1 + 3^{\frac{n-1}{3}}$ states.*

Now we consider the other values of n.

For $n = 3m + 2$, $m \geqslant 2$, we modify the definition of A_n, by introducing the extra state $(3,1)$. Furthermore, $\delta((2,1), b) = \{(3,1)\}$, $\delta((3,1), b) = \{(0,1)\}$, and $\delta((3,1), a) = \delta((2,1), a) = \{(2,2)\}$. The other transitions are unchanged. With respect to the automaton depicted in Figure 1, the modified A_n has one extra state in the first column. This new state is inserted in the loop joining, by transitions labeled with the letter b, the states of the first column, i.e., such a loop has length 4. Furthermore, the only transition from the new state on the letter a reaches the last state of the second column.

After this change, all the states in the first column are still not compatible with each other. Along the same lines of the previous proof, in particular of Remark 1 and Lemma 3, it can be proved that the set of reachable states in the subset automaton obtained from A_n corresponds to $\{q_0\}$ plus (an isomorphic copy of) the set $\{0,1,2,3\} \times \{0,1,2\}^{m-1}$. All these states are pairwise distinguishable. Hence the minimal dfa equivalent to A_n has $1 + 4 \cdot 3^{m-1}$ states.

For $n = 3m$, $m \geqslant 2$, the original definition of A_n is changed, by removing the state $(2,1)$ and the outgoing transitions. Furthermore, a transition on the letter b from $(1,1)$ to $(0,1)$ is added. Hence the loop on letters b in the first column is of length 2. In this case, we prove that the set of states of the minimal dfa equivalent to the given svfa is isomorphic to $\{q_0\} \cup \{0,1\} \times \{0,1,2\}^{m-1}$. We have to modify the proof of reachability in Lemma 3. The basis holds true since $(0,\ldots,0)$ goes to $(0,1,1,\ldots,1)$ by ba, and to $(0,2,2,\ldots,2)$ by bb. The induction step is the same as in Lemma 3, if $y_1 = 0$ or $y_1 = 1$. In the case of $y_1 = 2$, state $(0,2,y_2,\ldots,y_k,x,\ldots,x)$ can be reached by the string ab^2 from state $(0,y_2-2,\ldots,y_k-2,x-2,\ldots,x-2)$, which is reachable by induction. Hence the number of states is $1 + 2 \cdot 3^{m-1}$.

With the above considered cases, we get the optimality for each $n \geqslant 6$. We can also prove, using the same examples, the optimality for $n \in \{3,4,5\}$. In particular, for these values of n, we get that $n = 1 + f(n-1)$ and the svfa A_n coincides with the minimal dfa.

Finally, we consider $n = 1$ and $n = 2$. All the transitions in an svfa with only one state must be self-loops. Hence the svfa is already a dfa accepting or rejecting all Σ^*.

On the other hand, if an svfa has two states and both are accepting or both are rejecting, then they can be merged, obtaining a one-state svfa (previous case). Hence the only remaining case is that of an svfa consisting of one accepting and one rejecting state. We observe that a nondeterministic choice in such an automaton should imply both the acceptance and the rejection of the same string. Hence, also in this special case the svfa coincides with the minimal dfa.

By summarizing we have proved the following result.

Theorem 4. *For each integer $n \geqslant 1$ and each n-state self-verifying finite automaton, there exists an equivalent deterministic finite automaton with $g(n)$ states, where*

$$g(n) = \begin{cases} 1 + 3^{\frac{n-1}{3}}, & \text{if } n \equiv 1 \pmod 3 \text{ and } n \geqslant 4, \\ 1 + 4 \cdot 3^{\frac{n-2}{3}-1}, & \text{if } n \equiv 2 \pmod 3 \text{ and } n \geqslant 5, \\ 1 + 2 \cdot 3^{\frac{n}{3}-1}, & \text{if } n \equiv 0 \pmod 3 \text{ and } n \geqslant 3, \\ n, & \text{if } n \leqslant 2. \end{cases}$$

Furthermore, for each integer $n \geqslant 1$, there exists a binary n-state svfa A_n such that the minimal dfa equivalent to A_n has exactly $g(n)$ states.

5 Conclusions

We conclude the paper with some considerations concerning our main result. Theorem 4 gives a tight bound on the number of states of a dfa equivalent to a given n-state svfa. The optimality of this bound has been proved by considering a family of automata with a two-letter input alphabet. Hence, one can immediately ask whether or not the optimality holds in the case of unary automata, namely automata with a one-letter alphabet.

It is not difficult to give a negative answer to this question. In fact, as proved by Chrobak [11], the cost, in term of the number of states, of the optimal transformation of nfa's into dfa's is given by a function $F(n)$, such that $F(n) = e^{\Theta(\sqrt{n \log n})}$. Since this function grows slowly than the bound $g(n)$ given in Theorem 4 and svfa's are special nfa's, it turns out that $F(n)$ is a better upper bound for the conversion of unary n-state svfa's into dfa's.

Hence, the next natural question is whether or not this upper bound $F(n)$ is tight. Even in this case, the answer is negative. We can prove this by contradiction. Suppose that a language L is accepted by an n-state svfa A and the minimal dfa accepting L has $F(n)$ states. By Theorem 1, both L and L^c are accepted by n-state nfa's. However, in [12] it was proved that if a unary language L is accepted by an n-state nfa and the minimal dfa accepting L has $F(n)$ states, that is, the language L is a worst case for the determinization, then *each nfa* accepting L^c should have at least $F(n)$ states, which implies that the nondeterminism is useless in order to recognize L^c. This gives a contradiction. Hence, we have proved the following theorem.

Theorem 5. *For each unary n-state self-verifying finite automaton, there exists an equivalent deterministic finite automaton with less than $F(n)$ states.*

We can increase the nondeterministic capabilities of svfa's by adding the possibility of *multiple initial states*. In this case, at the beginning of the computation, the starting state is chosen from a set of possible initial states. The same requirements apply to this kind of automata, namely for each string, the automaton should have a computation reaching an accepting or a rejecting state, and the automaton cannot give two different answers. By the arguments in Section 3, we can prove that each n-state svfa with multiple initial states can be simulated by a dfa whose number of states is given by the function $f(n) = g(n+1) - 1$. We can prove that this bound is optimal, by making just a small change to the automata A_n considered in Section 4. For each integer $n \geqslant 1$, we define an automaton B_n by removing from A_{n+1} the initial state q_0 and by considering the set $\{(0,1),(0,2),\ldots,(0,m)\}$ of possible initial states. By our results in Section 4, it turns out that the minimal dfa equivalent to the svfa B_n must have $f(n)$ states.

We also notice that the transition diagram of the svfa B_n is deterministic. The only nondeterministic step is at the beginning of the computation for choosing the initial state. Hence, we actually proved that the tight cost of the simulation of n-state multiple initial state svfa's by dfa's is $f(n)$, even if the only nondeterministic step is at the beginning of the computation and all the transitions of the automaton are deterministic. A similar phenomenon happens for deterministic automata with multiple initial state. In [13] it has been proved that for each integer n, there exists a language accepted by an n-state deterministic automaton with multiple initial states such that the minimal equivalent dfa has $2^n - 1$ states.

Acknowledgments

We would like to thank the anonymous referees of LATA 2009 for their valuable comments and suggestions that help us to improve the presentation of our results.

References

1. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. UCS 8(2), 193–234 (2002)
2. Rabin, M., Scott, D.: Finite automata and their decision problems. IBM J. Res. Develop. 3, 114–125 (1959)
3. Lupanov, O.: A comparison of two types of finite automata. Problemy Kibernet 6, 321–326 (1963) (in Russian); German translation: Über den Vergleich zweier Typen endlicher Quellen, Probleme der Kybernetik 6, 329–335 (1966)
4. Moore, F.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Transactions on Computers C-20(10), 1211–1214 (1971)
5. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Proc. 12^{th} Ann. IEEE Symp. on Switching and Automata Theory, pp. 188–191 (1971)
6. Ďuriš, P., Hromkovič, J., Rolim, J., Schnitger, G.: Las Vegas versus determinism for one-way communication complexity, finite automata, and polynomial-time computations. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 117–128. Springer, Heidelberg (1997)
7. Hromkovič, J., Schnitger, G.: Nondeterministic communication with a limited number of advice bits. SIAM J. Comput. 33(1), 43–68 (2003)
8. Hromkovič, J., Schnitger, G.: On the power of Las Vegas for one-way communication complexity, OBDDs, and finite automata. Information and Computation 169(2), 284–296 (2001)
9. Assent, I., Seibert, S.: An upper bound for transforming self-verifying automata into deterministic ones. Theoretical Informatics and Applications 41(3), 261–265 (2007)
10. Moon, J., Moser, L.: On cliques in graphs. Israel J. Math. 3, 23–28 (1965)
11. Chrobak, M.: Finite automata and unary languages. Theoretical Computer Science 47, 149–158 (1986); Corrigendum. ibid 302, 497–498 (2003)
12. Mera, F., Pighizzini, G.: Complementing unary nondeterministic automata. Theoretical Computer Science 330(2), 349–360 (2005)
13. Holzer, M., Salomaa, K., Yu, S.: On the state complexity of k-entry deterministic finite automata. Journal of Automata, Languages and Combinatorics 6(4), 453–466 (2001)

Two Equivalent Regularizations for Tree Adjoining Grammars

Anna Kasprzik

Informatik FB IV, University of Trier, D-54286 Trier

Abstract. We present and compare two methods of how to make derivation in a Tree Adjoining Grammar a regular process (in the Chomsky hierarchy sense) without loss of expressive power. One regularization method is based on an algebraic operation called Lifting, while the other exploits an additional spatial dimension by transforming the components of a TAG into three-dimensional trees. The regularized grammars generate two kinds of "encoded" trees, from which the intended ones can be reconstructed by a simple decoding function. We can show the equivalence of these two two-step approaches by giving a direct translation between lifted and three-dimensional trees and proving that via this translation it is possible to switch between the encodings without losing the information necessary for the reconstruction of the intended trees.

Keywords: Tree Adjoining Grammar, Multi-Dimensional Trees, Lifting, Regularization.

1 Introduction

A Tree Adjoining Grammar (TAG) is a special kind of tree grammar which has been developed by Joshi et al. [1] in connection with studies on the formal treatment of natural languages. Joshi et al. [1] claimed the least class of formal languages containing all natural languages to be situated between the context-free and the context-sensitive languages in the Chomsky Hierarchy, and introduced the notion of *mild context-sensitivity*. The string language classes associated with TAGs are all situated in the family of mildly context-sensitive language classes, which became an important concept for computational linguistics.

As mild context-sensitivity represents a relatively high degree of complexity already, it would be of considerable use if there were a way to simplify derivation without giving up any of the expressive power. One of the simplest modes of derivation is the regular one where only the outermost components of an object (e.g. the last symbol of a string) may be rewritten. Regularizing a formalism has the obvious advantage that it makes the whole range of finite-state methods applicable ([2]), which is of interest for most areas based on formal language theory, e.g. natural language processing or grammatical inference ([3]), especially when more complex objects than strings are involved. As a matter of fact, for TAGs at least two such regularization methods exist, and the presentation and comparison of these two approaches constitute the main part of this work.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 469–480, 2009.

2 Preliminaries

We presuppose some familiarity with classical formal language theory and trees. The necessary preliminaries can be found for example in [4], [5], and also [6].

Definition 1. *A TAG is a 5-tuple* $\langle \Sigma, N, I, A, S \rangle$*, where* Σ *is the (non-ranked)* terminal labeling alphabet*,* N *is the (non-ranked)* nonterminal labeling alphabet *with* $N \cap \Sigma = \emptyset$*,* S *is the start symbol with* $S \in N$*,* I *is a finite set of* initial trees *where the root is labeled with* S*, and* A *is a finite set of* auxiliary trees*.*

Nonterminals label inner nodes and, in auxiliary trees, exactly one leaf which is referred to as the *foot node* and must carry the same symbol as the root. All other leaves are labeled by terminals or the empty word ε. New trees can be built by *adjunction*: A node in a tree is replaced by an auxiliary tree and the subtree formerly rooted at that node is attached to the foot node of the auxiliary tree.

A TAG can be enriched by associating a pair of constraints with every node, stating if adjunction is required or not (*obligatory adjunction (OA) constraint*), and which auxiliary trees may be adjoined at that node (*selective adjunction (SA) constraint*). These constraints obliterate the roles of nonterminal and terminal symbols and the start symbol, and hence the distinction between initial and auxiliary trees as well. Rogers [7] defines *non-strict* TAGs:

Definition 2. *A non-strict TAG is a pair* $\langle E, I \rangle$ *where* E *is a finite set of elementary trees in which each node is associated with a label from some alphabet, an SA constraint (a subset of* E*), and an OA constraint (Boolean valued).* $I \subseteq E$ *is a distinguished non-empty subset. Every elementary tree has a foot node.*

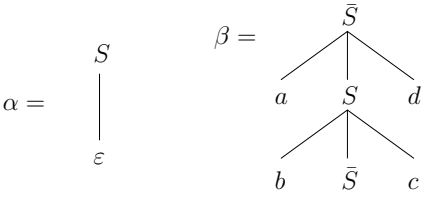


Fig. 1. A TAG generating the (non-context-free) string language $\{a^n b^n c^n d^n | n \geq 0\}$

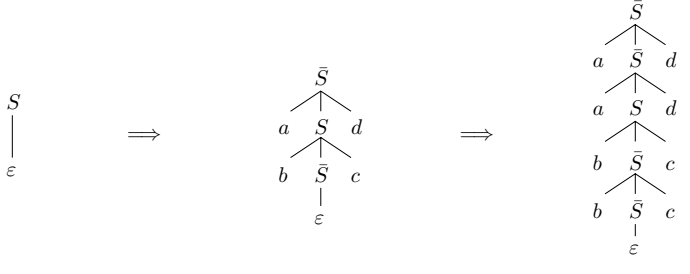


Fig. 2. A derivation of the word *aabbccdd*

Non-strict TAGs are equivalent to TAGs with adjunction constraints. We give an example for a non-strict TAG generating the language $\{a^n b^n c^n d^n | n \geq 0\}$:

Example 1. Let G with $G = \langle\{\alpha, \beta\}, \{\alpha\}\rangle$ be a non-strict TAG (over the alphabet $\{a, b, c, d,\ S\}$). The only initial tree α and the only auxiliary tree β are given in Figure 1. Constraints at the inner nodes and the foot node are: $OA = 0$ and $SA = \{\beta\}$ for the ones without a bar, and $OA = 0$ and $SA = \emptyset$ for the ones labeled with '$\bar{S}$'. The bar stands for null adjunction, i.e., no adjunction is allowed at these nodes. A derivation of the word *aabbccdd* is shown in Figure 2.

3 Lifted Trees

The technique of Lifting belongs to the realm of treating formal language classes with algebraical means. For the necessary preliminaries concerning universal algebra (esp. many-sorted algebras) and trees as terms, see [8,9].

Definition 3. *Let Σ be a ranked alphabet. The* derived $(\mathbb{N}^* \times \mathbb{N})$-indexed alphabet of Σ, *denoted by $D(\Sigma)$, is defined as follows. Let, for each $n \geq 0$, $\Sigma'_n = \{f' | f \in \Sigma_n\}$ be a new set of symbols; let for each n and each i, $1 \leq i \leq n$, π_i^n be a new symbol (the ith* projection symbol *of sort n); and let, for each $n, k \geq 0$, $c_{n,k}$ be a new symbol (the (n,k)th* composition symbol*). Then*

- $D(\Sigma)_{\varepsilon,0} = \Sigma'_0$;
- *for $n \geq 1$,* $D(\Sigma)_{\varepsilon,n} = \Sigma'_n \cup \{\pi_i^n | 1 \leq i \leq n\}$;
- *for $n, k \geq 0$,* $D(\Sigma)_{nk^n,k} = \{c_{n,k}\}$, *and*
- $D(\Sigma)_{w,s} = \emptyset$ *for $w \in \mathbb{N}^*$, $s \in \mathbb{N}$ otherwise.*

Note that all operators in Σ are treated as constants in $D(\Sigma)$. Lifting a term over some Σ just means translating it into a corresponding term over $D(\Sigma)$:

Definition 4 ([6]). *Let Σ be a ranked alphabet. For $k \geq 0$, $\mathsf{LIFT}_k^\Sigma : T_\Sigma(X_k) \longrightarrow T^k_{D(\Sigma)}$ (with $T_\Sigma(X_k)$ the set of terms over $\Sigma \cup \{x_1, \ldots, x_k\}$) is defined as follows:*

$$\begin{aligned}
\mathsf{LIFT}_k^\Sigma(x_i) &= \pi_i^k \\
\mathsf{LIFT}_k^\Sigma(f) &= c_{0,k}(f') \text{ for } f \in \Sigma_0 \\
\mathsf{LIFT}_k^\Sigma(f(t_1, \ldots, t_n)) &= c_{n,k}(f', \mathsf{LIFT}_k^\Sigma(t_1), \ldots, \mathsf{LIFT}_k^\Sigma(t_n)) \\
&\quad \text{for } n \geq 1, f \in \Sigma_n \text{ and } t_1, \ldots, t_n \in T(\Sigma, X_k) \ .
\end{aligned}$$

Lifting has a very useful side effect: If you note a term as a tree and lift it, all inner nodes become leaves. This obviously makes the operation of replacing those nodes by bigger structures a much "simpler" process – speaking in terms of formal language theory, a regular process. Consequently any context-free tree grammar (CFTG, based on the rewriting of inner nodes) over a signature Σ can be translated into a regular tree grammar (RTG, rewriting of leaves only) over the signature $D(\Sigma)$ by lifting the trees on the right hand sides of the productions and, since all nonterminals have become constants (i.e., leaf labels), by

deleting the variables representing daughters on the left hand sides. The intended trees over the original signature can then be reconstructed using the information contained in the "encoded" trees the RTG generates via the following function:

$$\begin{aligned} rec(f') &= f(x_1, \ldots, x_n) \text{ for } f \in \Sigma_n \\ rec(\pi_i^n) &= x_i \\ rec(c(t, t_1, \ldots, t_n)) &= rec(t)[rec(t_1), \ldots, rec(t_n)] \ . \end{aligned}$$

The proof of the following lemma (contained implicitly in [8]) is given in [10]:

Lemma 1. *Suppose $\Gamma = (\Sigma, \mathsf{F}, S, X, P)$ is a CFTG and $L(\Gamma)$ the tree language it generates. Then there is a derived regular tree grammar $\Gamma^L = (D(\Sigma), D(\mathsf{F}), S', P^L)$ such that $L(\Gamma)$ is the image of $L(\Gamma^L)$ under the mapping rec.*

The individual derivation steps also correspond:

Lemma 2. *A tree t' is derived in Γ^L from t in k steps, i.e., $t \Rightarrow t'$ via the productions $p_1^L, \ldots, p_k^L$ in P^L if and only if there are corresponding productions $p_1, \ldots, p_k$ in P such that $rec(t')$ is derived in Γ from $rec(t)$ via those productions.*

Morawietz [6] has collected some properties of trees generated by a lifted CFTG over some signature $D(\Sigma)$: a) All inner nodes are and no leaf is labeled by some composition symbol $c_{n,k}$, b) any node labeled with a symbol in Σ'_n, $n \geq 1$, is on a leftmost branch, and c) for any node p labeled with some projection symbol π_i^n there is a unique node μ which properly dominates p and whose ith sister will eventually evaluate to the value of π_i^n under the mapping *rec*. Moreover, μ will be the first node properly dominating p on a left branch. We will call lifted trees that fulfil condition c) *closed* lifted trees.

Lifting can be used to regularize TAGs as well. However, since TAGs function a little differently from CFTGs (which is linked to the fact that TAG trees are labeled by non-ranked symbols), this is not possible without some transformation. It has been proven by Fujiyoshi and Kasai [11] that every TAG can be translated into a spine grammar, which is a special kind of CFTG, where in the tree on the right-hand side of every production there exists a path from the root to a variable-labeled leaf and every other variable is the child of a node on that path. See [11] for the exact definition, construction, and proof of weak equivalence to TAGs. Their method can be easily adapted to non-strict TAGs as well. Example 2 shows the result of applying the translation of Fujiyoshi and Kasai [11] to the TAG of Example 1, and its lifted version.

Example 2. The new CFTG $G' = (\Sigma_0 \cup \Sigma_1 \cup \Sigma_3, \mathsf{F}_0 \cup \mathsf{F}_1 \cup \mathsf{F}_3, S', X, P)$ obtained from the TAG G in Example 1 is defined as follows: $\Sigma_0 = \{\varepsilon, a, b, c, d\}$, $\Sigma_1 = \{s_1\}$, $\Sigma_3 = \{s_3\}$, $X = \{x_1, x_2, x_3\}$, $\mathsf{F}_0 = \{S'\}$, $\mathsf{F}_1 = \{S_1\}$, $\mathsf{F}_3 = \{S_3\}$, and

$$P = \left\{ \begin{array}{c} S' \longrightarrow s_1(\varepsilon) \\ S' \longrightarrow S_1(\varepsilon) \\ S_1(x_1) \longrightarrow s_3(a, s_3(b, s_1(x_1), c), d) \\ S_1(x_1) \longrightarrow s_3(a, S_3(b, s_1(x_1), c), d) \\ S_3(x_1, x_2, x_3) \longrightarrow s_3(a, s_3(b, s_3(x_1, x_2, x_3), c), d) \\ S_3(x_1, x_2, x_3) \longrightarrow s_3(a, S_3(b, s_3(x_1, x_2, x_3), c), d) \end{array} \right\}$$

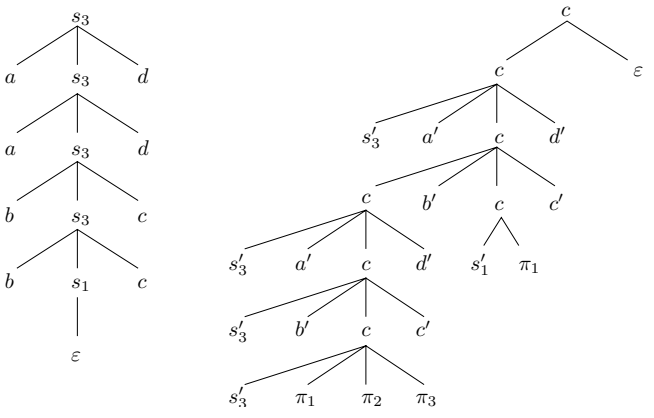


Fig. 3. Two trees, generated via corresponding rules of G' and G'^L

In lifted form, this grammar then looks like this: $G'^L = (D(\Sigma), D(\mathsf{F}), S'', P^L)$ with $D(\Sigma)_{\varepsilon,0} = \{\varepsilon, a', b', c', d'\}$, $D(\Sigma)_{\varepsilon,1} = \{s'_1\}$, $D(\Sigma)_{\varepsilon,3} = \{s'_3, \pi^3_1, \pi^3_2, \pi^3_3\}$, $D(\Sigma)_{nk^n,k} = \{c\}$, $D(\mathsf{F})_{\varepsilon,0} = \{S''\}$, $D(\mathsf{F})_{\varepsilon,1} = \{S'_1\}$, $D(\mathsf{F})_{\varepsilon,3} = \{S'_3\}$, and

$$P^L = \left\{ \begin{array}{l} S'' \longrightarrow c(s'_1, \varepsilon) \\ S'' \longrightarrow c(S'_1, \varepsilon) \\ S'_1 \longrightarrow c(s'_3, a', (s'_3, b', c(s'_1, \pi_1), c'), d') \\ S'_1 \longrightarrow c(s'_3, a', (S'_3, b', c(s'_1, \pi_1), c'), d') \\ S'_3 \longrightarrow c(s'_3, a', (s'_3, b', c(s'_3, \pi_1, \pi_2, \pi_3), c'), d') \\ S'_3 \longrightarrow c(s'_3, a', (S'_3, b', c(s'_3, \pi_1, \pi_2, \pi_3), c'), d') \end{array} \right\}$$

Figure 3 shows two corresponding trees generated by G' and G'^L – note how the elementary trees of the original TAG are still distinguishable. In fact we can state the following: In a tree generated by a lifted TAG the elementary trees are represented by the tree parts between one composition symbol on a leftmost branch and the next, and the expanded nodes correspond to the mother nodes of these composition symbols. In order to see this, consider the following observations: a) An adjunction operation in a TAG corresponds to the rewriting of an inner node in the corresponding CFTG, and consequently to the rewriting of a leaf in the corresponding lifted CFTG (see Lemma 2) and b) a composition symbol on a leftmost branch is certain evidence that at this node a production of the lifted CFTG has been applied.[1] It follows from this that the parts

[1] This can be explained as follows: Lifting a CFTG comprises lifting all the trees on the right-hand sides of the rules. In such a lifted tree all leftmost daughters of any node are labeled with symbols in Σ'_n (for some n), and never by composition symbols. Consequently, a composition symbol on a leftmost branch indicates that the tree cannot be contained as a whole in the rules of the grammar (i.e., as the right-hand side of a rule starting with '$S' \longrightarrow \ldots$') but that some production must have been applied that licenses the rewriting of a leaf by the subtree now rooted at the node in question (which, however, can contain other rewritings itself).

separated by composition symbols on leftmost branches must match the elementary components of the original TAG. Foot nodes are represented by nodes whose children except the leftmost are all labeled by projection symbols.

4 Three-Dimensional Trees and Their Yields

In this section we will consider a method based on a generalization by Rogers [12,7] of the concept of trees. Starting from ordinary trees based on two-dimensional tree domains Rogers extends the concept both downwards (to strings and points) and upwards and defines *labeled multi-dimensional trees*:

Definition 5. *Let* ${}^{d}1$ *be the class of all* d*th-order sequences of* 1*s:* ${}^{0}1 := \{1\}$*, and* ${}^{n+1}1$ *is the smallest set satisfying (i)* $\langle\rangle \in {}^{n+1}1$*, and (ii) if* $\langle x_1, \ldots, x_l\rangle \in {}^{n+1}1$ *and* $y \in {}^{n}1$*, then* $\langle x_1, \ldots, x_l, y\rangle \in {}^{n+1}1$*. Let* $\mathbb{T}^0 := \{\emptyset, \{1\}\}$ *(point domains). A* $(d{+}1)$*-dimensional tree domain is a set of hereditarily prefix closed* $(d{+}1)$*st-order sequences of* 1*s, i.e.,* $\mathsf{T} \in \mathbb{T}^{d+1}$ *iff*

- $\mathsf{T} \subseteq {}^{d+1}1$,
- $\forall s, t \in {}^{d+1}1 : s \cdot t \in \mathsf{T} \Rightarrow s \in \mathsf{T}$,
- $\forall s \in {}^{d+1}1 : \{w \in {}^{d}1 | s \cdot \langle w\rangle \in \mathsf{T}\} \in \mathbb{T}^d$.

A Σ*-labeled* $\mathsf{T}d$ *(*d-dimensional tree*) is a pair* (T, τ) *where* T *is a* d*-dimensional tree domain and* $\tau : T \longrightarrow \Sigma$ *is an assignment of labels in the (non-ranked) alphabet* Σ *to nodes in* T*. We will denote the class of all* Σ*-labeled* $\mathsf{T}d$ *as* $\mathbb{T}^d_\Sigma$*.*

Every d-dimensional tree can be conceived to be built up from d-dimensional *local trees*, that is, trees of depth at most one in their major dimension. Each of these smaller trees consists of a root and an arbitrarily large $(d-1)$-dimensional "child tree" consisting of the root's children. Composite trees can then be built from local ones by identifying the root of one local tree with a node in the child tree of another (see Figure 4 for an illustration). Rogers [7] also defines automata for multi-dimensional trees based on the notion of local trees.

For this paper, the most important concept to adapt to multi-dimensionality is that of the *yield* of a tree. The yield of a two-dimensional tree is the string formed by its leaf labels. In Rogers' words, it is a projection of the tree onto the next lower level, i.e., its dimensions are reduced by one. So d-dimensional trees with $d \geq 3$ have several yields, one for each dimension that is taken away, down to the one-dimensional string yield. Note that when taking the yield of a tree with $d \geq 3$, some thought has to go into the question of how to interweave the child trees of its local components to form a coherent $(d-1)$-dimensional tree, since there are often several possibilities. Rogers solves this by a construction quite similar to foot nodes in TAGs. See [7] for the exact definition.

Rogers [7] has established a link between three-dimensional trees and TAGs – he has proven the equivalence of $\mathsf{T}3$ recognizing automata and non-strict TAGs:

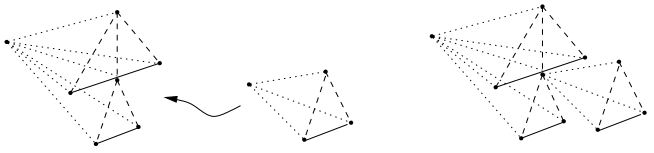


Fig. 4. Adjunction in TAG expressed via three-dimensional trees

Theorem 1. *A set of Σ-labeled two-dimensional trees is the yield of a recognizable set of Σ-labeled* T3 *trees iff it is generated by a non-strict TAG.*

From a certain perspective, trees accepted by a T3 automaton derived from a non-strict TAG can be seen as a special sort of derivation trees for that TAG in which one does not have to resort to tree names as node labels since both the elementary trees in question and the way they are combined can be displayed explicitly in the same object. Their direct yield is the set of the trees generated by that TAG, their one-dimensional yield is the corresponding string language.

The representation of a TAG via three-dimensional trees obviously also constitutes a regularization: Trees are now constructed by adding local trees at the frontier of another tree (see Figure 4), which is a regular process, instead of inserting trees at the interior. As stated above, the trees generated by the original TAG can be extracted from the three-dimensional trees via the yield function. We have thus described the second regularization method for TAG.

We introduce an a bit more "term-like" representation for three-dimensional trees, which will include the concept of rank (in the second dimension), in order to facilitate the comparison to lifted trees. Let Σ be an arbitrary ranked alphabet. We define the set $3\mathbb{D}_\Sigma$ of three-dimensional trees over Σ:

Definition 6. $(f, t) \in 3\mathbb{D}_\Sigma$ *if* $f \in \Sigma_0$ *(f is the root label) and* $t \in 3\mathbb{D}^+_\Sigma$ *(t is the structure formed by the nodes properly dominated by the root in the third dimension).* $3\mathbb{D}^+_\Sigma$ *is the set of* antitrunks *(three-dimensional trees without a root):*

- $(f, t, \langle t_1, \ldots, t_n\rangle) \in 3\mathbb{D}^+_\Sigma$ *if* $f \in \Sigma_n$, $t \in 3\mathbb{D}^+_\Sigma$ *and* $t_i \in 3\mathbb{D}^+_\Sigma$ *for* $0 \leq i \leq n$ *(the t_i are the daughter antitrunks of f in the second dimension).*
- $(f, \langle t_1, \ldots, t_n\rangle) \in 3\mathbb{D}^+_\Sigma$ *if* $f \in \Sigma_n$ *and* $t_i \in 3\mathbb{D}^+_\Sigma$ *for* $0 \leq i \leq n$.

In addition, we use a binary feature to indicate if a node is a foot node or not. For example, $(f, \langle\rangle, 1)$ for some label $f \in \Sigma$ and $t \in 3\mathbb{D}^+_\Sigma$ is a foot node, $(f, t, \langle t_1, t_2\rangle, 0)$ for $t_1, t_2 \in 3\mathbb{D}^+_\Sigma$ is not, and $(f, t, \langle t_1, t_2\rangle, 1)$ is not well-formed. We postulate the following conditions for foot nodes: a) Foot nodes are leaves in the second and third dimension, b) every contiguous two-dimensional tree in an antitrunk has to contain exactly one foot node, c) leaves in the second dimension are also leaves in the third dimension.

We will now define our own yield function $yd_\Sigma : 3\mathbb{D}^+_\Sigma \times \mathbb{N} \to T_{\Sigma^0}$ where Σ^0 is a ranked alphabet with $\Sigma^0_n = \Sigma_n \cup \Sigma_0$ for every $n \geq 0$ and T_{Σ^0} is the set of all two-dimensional trees over Σ^0. Let $x_1, \ldots, x_l$ be elements of a countable set of variables. Let $t_v, t_1, \ldots, t_m \in 3\mathbb{D}^+_\Sigma$. The function $g_{rk} : \Sigma_0 \times \mathbb{N} \to \Sigma^0$ takes a

label of rank 0 and yields another label consisting of the same symbol, but with the rank given in the second argument.

$$yd_{\Sigma}(t,l) = \begin{cases} f(yd_{\Sigma}(t_1,l),\ldots,yd_{\Sigma}(t_m,l)) & \text{if } f \in \Sigma_m,\ m \geq 0,\\ & t = (f,(t_1,\ldots,t_m),0)\\ yd_{\Sigma}(t_v,m)[(yd_{\Sigma}(t_1,l),\ldots,yd_{\Sigma}(t_m,l)] & \text{if } f \in \Sigma_m,\ m \geq 1,\ t = \\ & (f,t_v,(t_1,\ldots,t_m),0)\\ f_l(x_1,\ldots,x_l) & \text{if } t = (f,(),1) \text{ and}\\ \quad \text{for } f_l = g_{rk}(f,l) & f \in \Sigma_0 \end{cases}$$

The function has three cases because we have to distinguish between foot nodes and non-foot nodes, and among the latter between nodes that have an extension in the third dimension and nodes that do not. The second argument keeps track of the number of daughters of the roots in the third dimension so that when a foot node is reached the correct number of variables can be attached (which are then substituted by the direct subtrees of the corresponding root). The function yd works on antitrunks. In order to obtain the yield of a T3 tree we first have to apply a function $yd^{pre}_{\Sigma} : 3\mathbb{D}_{\Sigma} \rightarrow T_{\Sigma^0}$ to detach the root and initialize the second argument: $yd^{pre}_{\Sigma}(t_a) = yd_{\Sigma}(t,0)$ for a tree $t_a = (f,t)$ with $f \in \Sigma$ and $t \in 3\mathbb{D}^{+}_{\Sigma}$.

5 Equivalence

We would like to establish the equivalence of the two regularizations for TAGs presented above, based on either the transformation into a lifted spine grammar or into a T3 automaton. In order to do this, it is important to note several structural similarities between the tree-like objects defined by these devices: Both types of objects still include two kinds of information about the originally intended trees, namely which individual components, i.e., elementary trees they are composed of, and how these are put together, which is precisely the information needed to reconstruct the intended trees from the "encoded" ones.

With the T3 method the elementary trees are the child structures of the local trees of the T3 tree, and the points where the local trees are joined together are the nodes that are expanded by these child structures. In a tree generated by a lifted TAG the elementary trees are represented by the tree parts between one composition symbol on a leftmost branch and the next, and the expanded nodes correspond to the mother nodes of these composition symbols (see Section 3).

We will give a direct formal translation between lifted and T3 trees that exploits these structural similarities by finding corresponding points and making them match. Let us start by giving the function h_{li} translating lifted into T3 trees. For this, let $\mathbb{L}_{D(\Sigma)}$ be a set of trees over $D(\Sigma)$ characterized by:

- $\bigcup_{n\geq 0} D(\Sigma)_{\varepsilon,n} \in \mathbb{L}_{D(\Sigma)}$.
- $c(f,t_1,\ldots,t_n) \in \mathbb{L}_{D(\Sigma)}$ if $f \in {\Sigma_n}'$ and $t_1,\ldots,t_n \in \mathbb{L}_{D(\Sigma)} \setminus \{\pi^n_i | 1 \leq i \leq n\}$.
- $c(f,\pi_1,\ldots,\pi_n) \in \mathbb{L}_{D(\Sigma)}$ if $f \in \Sigma'_n$ and $n \geq 1$.

– $c(t, t_1, \ldots, t_n) \in \mathbb{L}_{D(\Sigma)}$ if $n \geq 1$, $t, t_1, \ldots, t_n \in \mathbb{L}_{D(\Sigma)} \setminus \{\pi_i^n | 1 \leq i \leq n\}$ and t contains projection symbols $\pi_1, \ldots, \pi_n$ that are not dominated by more than one composition symbol on a leftmost branch in t.

It is clear that all trees generated by a lifted CFTG derived from a TAG via the algorithm from [11] and all their subtrees are contained in $\mathbb{L}_{D(\Sigma)}$. We will therefore take $\mathbb{L}_{D(\Sigma)}$ as the domain of h_{li}. The range will be the set $3\mathbb{D}^+_\Sigma$.

Let in the following be $t_v, t_1 \ldots, t_n$, $q_1, \ldots, q_m \in \mathbb{L}_{D(\Sigma)}$ for all $n, m \geq 0$, and $t_v, t_1 \ldots, t_n \notin \{\pi_i | i \geq 1\}$. The function $g_{ze} : \bigcup_{n \geq 0} \Sigma'_n \to \Sigma_0$ takes a label and yields another label consisting of the same symbol, but with rank 0.

$$h_{li}(t) = \begin{cases} (f, (h_{li}(t_1), \ldots, h_{li}(t_n)), 0) & \text{if } t = c(f, t_1, \ldots, t_n) \\ & \text{with } f \in \Sigma'_n \text{ and } n \geq 0 \\ (\Diamond, h_{li}(t_v), (h_{li}(t_1), \ldots, h_{li}(t_n)), 0) & \text{if } t = c(t_v, t_1, \ldots, t_n), \\ & n \geq 1,\ t_v = c(q_1, \ldots, q_m) \\ & \text{and } m \geq 1 \\ (f, (), 1) \text{ with } f = g_{ze}(f_0) & \text{if } t = c(f_0, \pi_1, \ldots, \pi_n) \\ & \text{with } f_0 \in \Sigma'_n \text{ and } n \geq 1. \end{cases}$$

Like our yield function from the previous section, this function is subdivided into three cases. Here we distinguish between nodes that have projection symbols as daughters (future foot nodes) and nodes that do not, and among those between nodes whose leftmost daughter is a symbol in Σ'_n and nodes whose leftmost daughter is the root of another complex term (which is translated into an extension in the third dimension). As the function does not depend on the subscripts of the composition symbols ($c_{n,k}$ for some n and k), they are left out. The function h_{li} yields antitrunks. In order to translate the elements of $\mathbb{L}_{D(\Sigma)}$ into T3 trees we have to apply the function $p_{li} : \mathbb{L}_{D(\Sigma)} \to 3\mathbb{D}_\Sigma$ first with $p_{li}(t) = (\Diamond, h_{li}(t))$ that attaches a three-dimensional root and then recurs to the actual translation function. The symbol $\Diamond$ is a special placeholder for labelling three-dimensional roots, since the lifted trees do not contain the information about the labels they should have, i.e., the ones the nodes have in the original TAG before their expansion. It can have any rank ($\Diamond \in \Sigma_n$ for all $n \geq 0$).

The function $h_{3d} : 3\mathbb{D}^+_\Sigma \times \mathbb{N} \to \mathbb{L}_{D(\Sigma)}$ translates antitrunks into lifted trees. Let $t_v, t_1, \ldots, t_m \in 3\mathbb{D}^+_\Sigma$. The function $g_{rk(\varepsilon)} : \Sigma_0 \times \mathbb{N} \to \bigcup_{n \geq 0} \Sigma'_n$ takes a label of rank 0 and yields another label consisting of the same symbol, but of type $\langle \varepsilon, n \rangle$, where n is fixed by the second argument.

$$h_{3d}(t, n) = \begin{cases} c_m(f, h_{3d}(t_1, n), \ldots, h_{3d}(t_m, n)) & \text{if } m \geq 0,\ f \in \Sigma_m, \\ & t = (f, (t_1, \ldots, t_m), 0) \\ c_m(h_{3d}(t_v, m), & \text{if } m \geq 1,\ f \in \Sigma_m \text{ and} \\ \quad h_{3d}(t_1, n), \ldots, h_{3d}(t_m, n)) & t = (f, t_v, (t_1, \ldots, t_m), 0) \\ c_n(f, \pi_1, \ldots, \pi_n) & \text{if } n \geq 1,\ f_0 \in \Sigma_0 \\ \quad \text{with } f = g_{rk(\varepsilon)}(f_0, n) & \text{and } t = (f_0, (), 1). \end{cases}$$

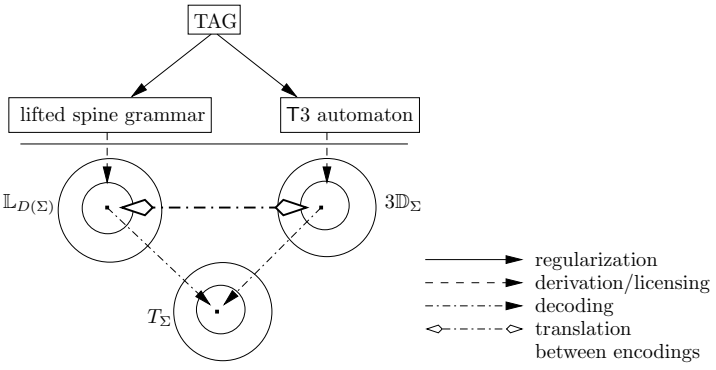


Fig. 5. Equivalence of Lifting and three-dimensional trees as TAG regularizations

The cases for this function are identical to the ones for the yield function. Composition symbols on left branches are translated into extensions in the third dimension, and foot nodes are translated into nodes with the right number of sisters (i.e., the number of daughters of the node that was expanded by the corresponding elementary tree in the original TAG) labeled by projection symbols. Of the composition symbols $c_{n,k}$ only the index n is given (the value of k is not as immediate as the one of n but can be easily inferred from the lifted tree afterwards). The function h_{3d} takes antitrunks as its input. If we want to use h_{3d} to translate a T3 tree $t_a = (f, t)$ with $f \in \Sigma$ and $t \in 3\mathbb{D}^+_\Sigma$, we first have to apply a function $p_{3d} : 3\mathbb{D}_\Sigma \to \mathbb{L}_{D(\Sigma)}$ with $p_{3d}(t_a) = h_{3d}(t, 0)$ that detaches the root of t_a and initializes the second argument of the translation function.

Formally, the equivalence of Lifting and of T3 trees as regularization methods for TAG can be shown by proving that the direct decoding of an "encoded" tree and its translation into the other encoding and the decoding of the result yield exactly the same originally intended tree (see Figure 5 for a diagram of the relevant connections). We therefore state the following theorem – a detailed (very technical) proof can be found in [13].[2]

Theorem 2. *For all closed lifted trees $t_l \in \mathbb{L}_{D(\Sigma)}$ generated by the lifted version of some TAG (over the alphabet Σ) and all trees $t_s \in 3\mathbb{D}_\Sigma$ generated by the* T3 *tree version of the same TAG,*

- $rec(t_s) = yd^{pre}(p_{li}(t_s))$, *and*
- $yd^{pre}(t_s) = rec(p_{3d}(t_s))$.

Figure 6 shows two encoded trees and the intended tree, which can be generated by the TAG from Example 1 (see Figure 2). The lifted tree in the lower right

[2] We also believe an even stronger equivalence in the sense of a bijection to hold: Let A be the set of trees generated by a lifted spine grammar that is the regularized version of a TAG, and let B be the set accepted by a T3 automaton that has been extracted from the same TAG. Then $p_{li}(A) = B$ and $p_{3d}(B) = A$, and even, for some tree $t_1 \in A$, $p_{3d}(p_{li}(t_1)) = t_1$ as well as $p_{li}(p_{3d}(t_2)) = t_2$ for some tree $t_2 \in B$.

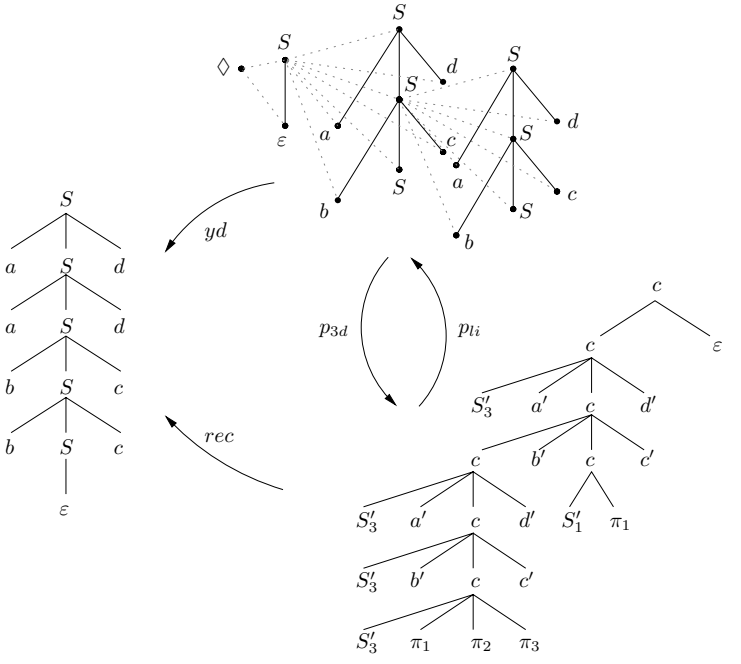


Fig. 6. Two encoded trees and the corresponding intended tree

corner can be generated by the regularized version of that TAG (see Example 2), and in the upper right corner is the matching three-dimensional tree.

6 Conclusion

In this paper, we have presented two regularization methods for TAGs. Regularized TAGs of both kinds define sets of "encoded" trees, which, however, still contain the necessary information to reconstruct the intended trees defined by the original TAG. Both methods operate by transforming the components of a TAG in a way that turns all inner nodes into leaves, thus making it possible to expand these nodes by means of a regular mechanism. Both methods exploit a side effect of the theoretical concepts they are based on – algebraic Lifting and the notion of multi-dimensional trees – since neither was developed with the primary intention of regularizing the grammar formalism TAG.

However, the methods described here have even more in common: The two different kinds of objects generated by regularized TAGs exhibit a number of structural similarities, which we exploited in order to show their direct translatability. Objects of both kinds can be seen as a special sort of derivation tree for the originally intended tree, and it is only natural that this should be somehow related to regularization in that derivation trees, which are composed starting from a root and adding every further step of the derivation somewhere at the leaves, are a special sort of regular trees – recall that regularity in general represents a mode of constructing an object where one element after the other is

added at the frontier of that object, and not somewhere in between. By letting us find different kinds of derivation trees for a formalism, in addition to gaining knowledge about how the properties of derivation as such can be modified, the study of regularization may thus perhaps give further insight about derivation in the formalism in particular as well.

A possible continuation of this work could be to search for more regularization methods for TAGs and determine if the structure of the objects created in the process resembles the structure of the objects treated here. One could even conjecture that every similar effort to reduce the complexity of derivation must result in similar properties, i.e., in objects that are directly translatable into lifted or three-dimensional trees as well.

References

1. Joshi, A.K., Levy, L.S., Takahashi, M.: Tree Adjunct Grammars. Journal of Computer and System Sciences 10(1), 136–163 (1975)
2. Kasprzik, A.: Making Finite-State Methods Applicable to Languages Beyond Context-Freeness via Multi-dimensional Trees. In: Piskorski, J., Watson, B., Yli-Jyrä, A. (eds.) Post-proc. 7th Int. Workshop on Finite-State Methods and NLP. IOS Press, Amsterdam (2009), www.uni-trier.de/index.php?id=18342
3. Kasprzik, A.: A Learning Algorithm for Multi-dimensional Trees, or: Learning beyond Context-Freeness. In: Clark, A., Coste, F., Miclet, L. (eds.) ICGI 2008. LNCS, vol. 5278, pp. 111–124. Springer, Heidelberg (2008), www.uni-trier.de/index.php?id=18342
4. Hopcroft, J.E., Ullman, J.D.: Introduction to automata theory, languages, and computation. Addison-Wesley, Reading (1979)
5. Gécseg, F., Steinby, M.: Tree automata. Akademiai Kiado (1984)
6. Morawietz, F.: Two-Step Approaches to Natural Language Formalisms. Studies in Generative Grammar, vol. 64. Mouton de Gruyter, Berlin (2003)
7. Rogers, J.: wMSO Theories as Grammar Formalisms. TCS 293, 291–320 (2003)
8. Engelfriet, J., Schmidt, E.: IO and OI (I). J. Comp. & Sys. Sci. 15, 328–353 (1977)
9. Engelfriet, J., Schmidt, E.: IO and OI (II). J. Comp. & Sys. Sci. 16, 67–99 (1978)
10. Mönnich, U.: On cloning contextfreeness. In: Kolb, H.P., Mönnich, U. (eds.) Studies in Generative Grammar, vol. 44, pp. 195–229. Mouton de Gruyter, Berlin (1999)
11. Fujiyoshi, A., Kasai, T.: Spinal-formed context-free tree grammars. Theory of Computing Systems 33, 59–83 (2000)
12. Rogers, J.: Syntactic Structures as Multi-dimensional Trees. Research on Language and Computation 1, 265–305 (2003)
13. Kasprzik, A.: Two Equivalent Regularizations for Tree Adjoining Grammar. M.A. thesis (2007), http://www.uni-trier.de/index.php?id=18342

Self-overlapping Occurrences and Knuth-Morris-Pratt Algorithm for Weighted Matching

Aude Liefooghe[1,2], Hélène Touzet[1,2], and Jean-Stéphane Varré[1,2]

[1] LIFL - UMR CNRS 8022 - Université des Sciences et Technologies de Lille, 59655 Villeneuve d'Ascq Cedex, France
[2] INRIA Lille Nord-Europe, 59650 Villeneuve d'Ascq, France
firstname.lastname@lifl.fr

Abstract. Position Weight Matrices are broadly used probabilistic motif models. In this paper, we address the problem of identifying and characterizing potential overlaps between occurrences of such a motif. It has useful applications to the statistics of the number of occurrences, and to weighted pattern matching with an extension of the well-known Knuth-Morris-Pratt algorithm.

1 Introduction

Position Weight Matrices (PWMs for short) are probabilistic approximate patterns. They are popular in computational biology, because they can depict subtle signals in genomic or peptidic sequences. For example, transcription factor binding sites are commonly modeled by such motifs. PWMs are also used to represent splice sites in messenger RNAs [1] or signatures in amino acid sequences [2].

Given a finite alphabet Σ and a positive integer m, a PWM M is a function from Σ^m to $\mathbb{R}$ that associates a score to each word of Σ^m. It has one row for each symbol of the alphabet, and one column for each position in the pattern. The coefficient $M(p, x)$ gives the score at position p for the letter x in Σ. Given a string u in Σ^m, the *score* of M on u is defined as the sum of the scores of each character symbol of u:

$$\texttt{Score}(u, M) = \sum_{p=0}^{m-1} M(p, u_p),$$

where u_p denotes the character symbol at position p in u. Let α be a score threshold. We say that M has an *occurrence* in the text T at position k if $\texttt{Score}(T_k \dots T_{k+m-1}, M) \geq \alpha$.

In this paper, we investigate the self-overlapping properties of PWMs. The propensity of a motif to have overlapping occurrences in a text has many consequences. It can be used for computing the statistics of the number of occurrences in a random text, since occurrences of a self-overlapping motif are not independent [3,4]. It is also crucial in pattern matching. Among ideas that allow to speed

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 481–492, 2009.

up pattern searching, many of them exploits a preprocessing of the pattern that takes advantage of internal overlaps inside the pattern. Famous examples are Knuth-Morris-Pratt [5] and Boyer-Moore [6] algorithms. In the case of multiple exact pattern matching, the same idea gives rise to the Aho-Corasick automaton [7].

The contributions of this paper are the following. In Section 2, we provide a formal characterization of self-overlapping PWMs, as well as an algorithm to efficiently compute the borders of a PWM. In Section 3, we show how to use this result to express the covariance and the variance of the number of occurrences of a PWM in a text. In Section 4, we explain how to adapt the Knuth-Morris-Pratt paradigm to PWMs on the basis of borders for PWMs. This last point gives rise to three pattern matching algorithms, that we have experimentally evaluated on PWMs from the Jaspar database [8].

2 Analysis of Self-overlapping Occurrences of a PWM

We consider throughout a text T of length n on a finite alphabet Σ, and we assume that the character symbols at various positions are independent. Each character symbol x of Σ is assigned a probability μ_x. We also consider a PWM M of length m on Σ, whose indices range from 0 to $m-1$.

2.1 Definitions

The "self-overlapping" property of a word u gives rise to several definitions. The terms may be different in each application field, but the concept is the same. In statistics, this property is usually handled with an indicator function, that is equal to 1 if the first ℓ letters of u are the same, and in the same order as the last ℓ letters of u, and 0 otherwise. In stringology, this property is captured with the notion of *border*. A border of u of length ℓ is a prefix of length ℓ which is also a suffix of u. We extend this definition to motifs modeled by PWMs.

Definition 1 (Matrix border). *Let M be a PWM of length m and let α be a score threshold for M. A* border *for M and α is a word u of Σ^ℓ, with $\ell < m$, such that there exist $v, w \in \Sigma^{m-\ell}$ satisfying:*

1. $\texttt{Score}(M, uv) \geq \alpha$, *and*
2. $\texttt{Score}(M, wu) \geq \alpha$.

It is easy to see that this definition coincides with the definition of border in the case of exact strings. We introduce an equivalent characterization, that relies on the definition of the maximal score of a matrix, and the greatest lower bound of the score of a given position.

Definition 2 (Maximal score). *Let M be a PWM of length m, and let $M[i..j]$ be a slice of M ($0 \leq i \leq j < m$). The* maximal score of $M[i..j]$, *denoted* $\texttt{MaxSc}(M[i..j])$, *is*

$$\texttt{MaxSc}(M[i..j]) = \sum_{k=i}^{j} \max_{x \in \Sigma} M(k, x)$$

Definition 3 (Greatest Lower Bound). *Let M be a PWM of length m, α a score threshold, and p a position of M $(0 \leq p < m)$. The* Greatest Lower Bound *of M, p and α, denoted, $\texttt{GLB}(M, p, \alpha)$, is defined as*

$$\texttt{GLB}(M, p, \alpha) = \alpha - \texttt{MaxSc}(M[p+1..m-1])$$

The greatest lower bound score was originally introduced in [9]. It can be shown that a word u is an occurrence for M and α if, and only if, for each position p of u, $\texttt{Score}(M, u_0..u_p) \geq \texttt{GLB}(M, p, \alpha)$.

Lemma 1. *Let M be a PWM of length m, and let α be a score threshold. The word u of length ℓ is a border of M and α if, and only if, u fulfills the two following properties:*

1. $\texttt{Score}(M[0..\ell-1], u) \geq \texttt{GLB}(M, \ell-1, \alpha)$, *and*
2. $\texttt{Score}(M[m-\ell..m-1], u) \geq \alpha - \texttt{MaxSc}(M[0..m-\ell-1])$.

Proof
($\Rightarrow$) Let u be a border of length ℓ. By Definition 1, there exist two words v and w such that $\texttt{Score}(M, uv) \geq \alpha$ and $\texttt{Score}(M, wu) \geq \alpha$. From the first inequality, it follows that $\texttt{Score}(M[0..\ell-1], u) \geq \alpha - \texttt{Score}(M[\ell..m-1], v)$ and then $\texttt{Score}(M[0..\ell-1], u) \geq \texttt{GLB}(M, \ell-1, \alpha)$. From the second inequality, it follows that $\texttt{Score}(M[m-\ell..m-1], u) \geq \alpha - \texttt{Score}(M[0..m-\ell-1], w)$ and then $\texttt{Score}(M[m-\ell..m-1], u) \geq \alpha - \texttt{MaxSc}(M[0..m-\ell-1])$.

($\Leftarrow$) Assume that u satisfies statements 1. and 2. of the Lemma. By Definition 3, this implies that there exists a word v such that $\texttt{Score}(M, uv) \geq \alpha$, and by Definition 2 that there exists a word w such that $\texttt{Score}(M, wu) \geq \alpha$. Subsequently, u is a border of M and α. □

Definition 4 ($c(M, p, \alpha)$). *Define the predicate $c(M, p, \alpha)$ as true if, and only if, there exists a border of length $m - p$ for M and α.*

2.2 How to Compute *c*

We introduce b, that will serve as an auxiliary function to compute c.

Definition 5. *Let M be a PWM of length m, and let p, i be two positions of M $(0 \leq p \leq i \leq m-1)$. We define the function b as*

$$b(M, p, i, \beta, \delta) = \exists u \in \Sigma^{i-p+1}(\texttt{Score}(M[p..i], u) = \beta \wedge \texttt{Score}(M[0..i-p], u) = \delta)$$

Lemma 2

$$c(M, p, \alpha) = \exists\, \beta \geq \alpha - \texttt{MaxSc}(M[0..p-1])\ \ \exists\, \delta \geq \texttt{GLB}(M, m-1-p, \alpha) \text{ such that } b(M, p, m-1, \beta, \delta)$$

Proof. By Definition 4 and Lemma 1, $c(M, p, \alpha)$ is true if, and only if, there exists a word u of length $m - p$ such that

1. $\texttt{Score}(M[0..m-p-1], u) \geq \texttt{GLB}(M, m-p-1, \alpha)$, and
2. $\texttt{Score}(M[p..m-1], u) \geq \alpha - \texttt{MaxSc}(M[0..p-1]$

Set $\beta = \alpha - \texttt{MaxSc}(M[0..p-1])$ and $\delta = \texttt{GLB}(M, m-1-p, \alpha)$. This concludes the proof. □

Lemma 2 implies that it is sufficient to be able to compute b to deduce c. We now explain how to compute b efficiently.

Lemma 3

$$\begin{cases} b(M,p,i,0,0) = 1, \text{ whenever } i < p \\ b(M,p,i,\beta,\delta) = 0, \text{ whenever } i < p,\ \beta \neq 0 \text{ or } \delta \neq 0 \\ b(M,p,i,\beta,\delta) = \vee_{x \in \Sigma}\ b(M,p,i-1,\beta - M(i,x), \delta - M(i-p,x)) \text{ otherwise} \end{cases}$$

Proof. When $i < p$, the results comes from the definition of b. For the general case, $b(M,p,i,\beta,\delta)$ is true if, and only if, there exists a letter x of Σ satisfying the following property: there exists v in Σ^{i-p} such that $\texttt{Score}(M[p..i-1], v) = \beta - M(i,x)$ and $\texttt{Score}(M[0..i-p-1], v) = \delta - M(i-p,x)$, which is equivalent to $b(M,p,i-1,\beta - M(i,x), \delta - M(i-p,x))$.

So for each p, $b(M,p,i,\beta,\delta)$ can be computed by lazy dynamic programming from smaller values of i, β and δ. A useful remark is that it is not necessary to consider all possible parameters for the final value of c. This is the goal of the next lemma.

Lemma 4. *$c(M,p,\alpha)$ is true if, and only if, for each value i ranging from p to $m-1$, there exist β and δ, such that*

1. *$b(M,p,i,\beta,\delta) = 1$, and*
2. *$\beta \geq \texttt{GLB}(M,i,\alpha) - \texttt{MaxSc}(M[0..p-1])$, and*
3. *$\delta \geq \texttt{GLB}(M, i-p, \alpha)$.*

Proof
($\Rightarrow$) If $c(M,p,\alpha)$ is true, then we know from Definition 5 and Lemma 2 that there exists a word u of Σ^{m-p} such that

$$\begin{aligned} \texttt{Score}(M[p..m-1], u) &\geq \alpha - \texttt{MaxSc}(M[0..p-1]) \\ \texttt{Score}(M[0..m-1-p], u) &\geq \texttt{GLB}(M, m-1, \alpha) \end{aligned}$$

Consider now a position i in $p..m-1$. The values $\beta = \texttt{Score}(M[p..i], u_0..u_{i-p})$ and $\delta = \texttt{Score}(M[0..i-p], u_0..u_{i-p})$ fulfills the conditions of the Lemma, by definition of GLB.

($\Leftarrow$) Straightforward. □

The implementation can use a hashtable to store the set of pairs of scores (β, δ) satisfying the conditions of Lemma 4 for given values of i and p: $c(M,p,\alpha)$ is true if, and only if, $\mathcal{S}^p_{m-1}$ is not empty.

3 Application to Counting Occurrences in a Text

In this section, we are interested in the number of occurrences of a PWM in a text T. We define Y as the random number of occurrences of M and α in T. Our aim is to find the mean and the variance of Y.

3.1 Probability of Observing Two Overlapping Occurrences

We first study the probability to observe two overlapping occurrences for M and α. For that purpose, define the indicator variable Y_k by $Y_k = 1$ if M occurs in position k in the text T with score greater than α, $Y_k = 0$ if it does not. It is routine to verify that

$$E[Y_k] = \mathtt{Pv}(M, \alpha)$$
$$V[Y_k] = \mathtt{Pv}(M, \alpha)(1 - \mathtt{Pv}(M, \alpha))$$

where $\mathtt{Pv}(M, \alpha)$ denotes the P-value of the score α for M, that is the probability of the set of words that are recognized by M and α:

$$\mathtt{Pv}(M, \alpha) = I\!P(\{u \in \Sigma^m;\ \ \mathtt{Score}(M, u) \geq \alpha\})$$

The computation of $\mathtt{Pv}(M, \alpha)$ can be performed efficiently by dynamic programming [10,11]. We are interested in the probability of having $Y_k = 1$ and $Y_{k+p} = 1$, for some $1 \leq p \leq m-1$. This event is true if there exist β and δ such that

1. $\mathtt{Score}(M[0..p-1], T_k..T_{k+p-1}) \geq \alpha - \beta$,
2. $\mathtt{Score}(M[p..m-1], T_{k+p}..T_{k+m-1}) = \beta$ and
   $\mathtt{Score}(M[0..m-p-1], T_{k+p}..T_{k+m-1}) = \delta$,
3. $\mathtt{Score}(M[m-p..m-1], T_{k+m}..T_{k+p+m-1}) \geq \alpha - \delta$.

Since positions in the text are independent, this decomposition gives rise to three mutually independent events, whose probabilities are respectively $\mathtt{Pv}(M[0..p-1], \alpha - \beta)$, $B(M, p, m-1, \beta, \delta)$ and $\mathtt{Pv}(M[m-p..m-1]), \alpha - \delta)$ where:
$B(M, p, i, \beta, \delta) =$
$$I\!P\left(\left\{u \in \Sigma^{i-p+1};\ \mathtt{Score}(M[p..i], u) = \beta\ \wedge\ \mathtt{Score}(M[0..i-p], u) = \delta\right\}\right)$$
It follows that

$$I\!P(Y_k = 1, Y_{k+p} = 1) = \sum_{\beta,\delta} \begin{array}{l} \mathtt{Pv}(M[0..p-1], \alpha - \beta) \\ \times B(M, p, m-1, \beta, \delta) \\ \times \mathtt{Pv}(M[m-p..m-1]), \alpha - \delta) \end{array}$$

The computation of B is similar to the computation of b.

3.2 Mean, Covariance and Variance

Lemma 5. $E[Y] = (n - m + 1)\mathtt{Pv}(M, \alpha)$.

Lemma 6. $\mathrm{Cov}[Y_k, Y_{k+p}] = \sum_{\beta,\delta} \mathtt{Pv}(M[0..p-1], \alpha - \beta) \times B(M, p, m-1, \beta, \delta) \times \mathtt{Pv}(M[m-p..m-1]), \alpha - \delta) - \mathtt{Pv}(M, \alpha)^2$

Proof. By definition of the covariance, we have

$$\mathrm{Cov}[Y_k, Y_{k+p}] = E[Y_k, Y_{k+p}] - E[Y_k]E[Y_{k+p}]$$

Since Y_k and Y_{k+p} are indicator functions, $E[Y_k, Y_{k+p}]$ equals $I\!P(Y_k = 1, Y_{k+p} = 1)$, which has been calculated in the preceding paragraph. $E[Y_k]$ and $E[Y_{k+p}]$ both equal $\mathtt{Pv}(M, \alpha)$. □

Lemma 7. $V[Y] = \mathtt{Pv}(M,\alpha)((n-m+1)(1-\mathtt{Pv}(M,\alpha)) - \mathtt{Pv}(M,\alpha))$
$+2\sum_{\beta,\delta} \mathtt{Pv}(M[0..p-1],\alpha-\beta) \times B(M,p,m-1,\beta,\delta) \times \mathtt{Pv}(M[m-p..m-1]),\alpha-\delta)$

Proof. By property of a variance, we have

$$V[Y] = \sum_{k=0}^{n-m} V[Y_k] + \sum_{k=0}^{n-m} \sum_{j=0}^{n-m} \mathrm{Cov}[Y_k, Y_j]$$

In the sum $\sum_{k=0}^{n-m} \sum_{j=0}^{n-m} \mathrm{Cov}[Y_k, Y_j]$, we only have to consider positions k and j such that $k-m < j < k+m$. Other covariances equal 0. Applying Lemma 6 yields the expected result. □

4 Application to Text Searching and the Knuth-Morris-Pratt Algorithm

4.1 Previous Work Related to PWM Pattern Matching

The problem of efficiently finding occurrences of a PWM in a text has recently attracted a lot of interest. Wu *et al.* were the first ones to propose an improvement of the naive algorithm with the lookahead strategy [9]. They used the fact that one can determine in advance whether a prefix of a word may potentially lead to an occurrence or not, using the greatest lower bound of Definition 3. In [12], Liefooghe *et al.* combined this lookahead strategy with the additivity property of the scoring function to implement a solution based on a multi-table index. This index pre-compute scores for slices of the given matrices and stores them in an optimized way.

Other authors proposed to adapt methods that were initially designed for exact string matching and that are proven to be efficient in this context. The first possibility is to preprocess the text. Beckstette *et al.* proposed in [13] to build a suffix array and took advantage of the lookahead strategy to improve the size of the index. Pizzi *et al.*[14] introduced a filtration technique that is based onto a preprocessing of the text for which the computation of the occurrences of an automaton that contains at least all occurrences of the matrix are computed. Lastly, Salmela *et al.* proposed an expansion of the well-known Shift-Add algorithm to PWMs [15]. The principle of Shift-Add approaches is to use bitwise computation during the searching phase. The complexity of the algorithm depends on the score threshold and the rounding of the scores of the matrix.

Another idea is to preprocess the pattern in order to maximise the length of the shift of the pattern after a mismatch occurs during the scan of the text. This is implemented in the Morris-Pratt and Knuth-Morris-Pratt algorithms for exact string matching. The method computes a table that gives the next position of the text that has to be tested when an attempt fails. The searching phase consists in jumping from a current position to the next position using the value stored in the table. In [14], Pizzi *et al.* studied the construction of the Aho-Corasick automaton [7] for PWMs, which is a generalization of Knuth-Morris-Pratt for multi-pattern matching. It appears that the Aho-Corasick automaton

is very large if the PWM is long or if the score threshold is low. In practice, this method is not suitable for matrices of length greater than fifteen. The advantage of Morris-Pratt and Knuth-Morris-Pratt tables is that their size equals the length of the motif. We recall the way the algorithm works on exact string motifs and we then extend it to PWMs.

4.2 The Original Morris-Pratt and Knuth-Morris-Pratt Algorithms

The Morris-Pratt and Knuth-Morris-Pratt algorithms search for occurrences of a word u within a text T by employing the observation that when a mismatch occurs, the word itself embodies some information to determine where the next match could begin. This information is used to improve the length of the shift of the pattern against the text.

Morris-Pratt Algorithm. The shift of the Morris-Pratt algorithm is based on the longest border. Let $\texttt{mpNext}[i]$ be the length of the longest border of $u_0..u_{i-1}$ for $0 < i \leq m$. When an attempt fails at position i in the pattern and j in the text, then the comparison can resume between characters $i - \texttt{mpNext}[i]$ in u and j in T without missing any occurrence.

Knuth-Morris-Pratt Algorithm. The additional idea of the Knuth-Morris-Pratt algorithm is that if we want to avoid another immediate mismatch, the character following the prefix in the pattern must be different from the current symbol. For that, the shift rule uses *tagged borders*, instead of borders. Given a border v of length ℓ of the prefix $u_0 \dots u_{p-1}$ of length of u, v is a tagged border if the character symbol u_p is different from u_ℓ. $\texttt{kmpNext}[i]$ is the length of the longest tagged border of $u_0..u_{i-1}$.

4.3 Expansion to Position Weight Matrices

In the context of exact string matching, a failure comes from a mismatch between the motif and the text. An attempt at position k in the text stops at position i if $T_k..T_{k+i-1} = u_0..u_{i-1}$ and $T_{k+i} \neq u_i$. When the motif is a PWM, this rule depends on the score of the prefix of the motif. An attempt stops as soon as the running score is too low: $\texttt{Score}(M[0..i], T_k..T_{k+i}) < \texttt{GLB}(M, i, \alpha)$. By Definition 3, we know that there is no occurrence of M and α starting at position k in the text. We now explain how to construct the `mpNext` and `kmpNext` tables.

Morris-Pratt Algorithm. The extension to PWMs is straightforward. We denote $\texttt{Border}(M, \alpha)$ the length of the longest border of M and α, according to Definition 1. `mpNext` is a vector of size $m + 1$ whose elements are defined as follows:

$$\begin{aligned} &\texttt{mpNext}[0] = -1 \\ &\texttt{mpNext}[i] = \texttt{Border}(M[0..i-1], \texttt{GLB}(M, i-1, \alpha)), \quad 1 \leq i \leq m \end{aligned}$$

The computation of each element of `mpNext` is done using Definition 4: $\texttt{mpNext}[i]$ equals $i-p$, where p is the lowest value such that $c(M[0..i-1], p, \texttt{GLB}(M, i-1, \alpha))$ is true. All necessary values of $c(M[0..i-1], p, \texttt{GLB}(M, i-1, \alpha))$ can be efficiently computed using Lemma 4.

Knuth-Morris-Pratt Algorithm. We first need to extend the notion of tagged border to position weight matrices.

Definition 6. *Let M be a PWM, i a position of M and α a score threshold. u is a* tagged border *for M, i and α, if u is a border of $M[0..i]$ and $\texttt{GLB}(M, i, \alpha)$, and there exists a letter x of Σ such that*

1. $\texttt{Score}(M[0..\ell], ux) \geq \texttt{GLB}(M, \ell, \alpha)$, *and*
2. $\texttt{Score}(M[i-\ell..i], ux) < \texttt{GLB}(M, i, \alpha) - \texttt{MaxSc}(M[0..i-\ell-1])$.

where ℓ is the length of u.

We write $\texttt{TaggedBorder}(M, i, \alpha)$ for the length of the longest tagged border of M, i and α. By convention, $\texttt{TaggedBorder}(M, i, \alpha)$ equals -1 if a suitable u does not exist. We now define the table `kmpNext` as follows:

$$\begin{aligned} \texttt{kmpNext}[0] &= -1 \\ \texttt{kmpNext}[i] &= \texttt{TaggedBorder}(M, i-1, \alpha), \quad 1 \leq i \leq m \end{aligned}$$

We introduce a new predicate to compute `kmpNext`.

Definition 7 ($c'(M, p, i, \alpha)$). *Define the predicate $c'(M, p, i, \alpha)$ as true if, and only if, there exists a tagged border of length $i - p$ for M, $i - 1$ and α.*

$\texttt{kmpNext}[i]$ equals $i - p$, where p is the lowest value such that $c'(M, p, i-1, \alpha)$ is true. The effective computation of c' is done with the next lemma, that shows that c' can be deduced from b.

Lemma 8. *$c'(M, j, i, \alpha)$ is true if, and only if, there exist two scores β, δ and a letter x of Σ such that*

1. $b(M, j, i-1, \beta, \delta) = 1$, *and*
2. $\texttt{GLB}(M, i-1, \alpha) - \texttt{MaxSc}(M[0..j-1]) \leq \beta < \texttt{GLB}(M, i, \alpha) - \texttt{MaxSc}(M[0..j-1]) - M(i, x)$, *and*
3. $\delta \geq \texttt{GLB}(M, i-j, \alpha) - M(i-j, x)$.

4.4 The $\texttt{kmpNext}_{\Sigma}$ Table

We propose here a variation of the Knuth-Morris-Pratt algorithm, that implements a shift which depends on the failure letter. Instead of computing the longest tagged border, we compute for each letter x of Σ the longest border which is not followed by x. We will see in Section 4.6 that it gives rise to better shifts and then to a better running time.

Definition 8 (x-tagged border). *Let M be a PWM, i a position of M, α a score threshold and x a letter of Σ. u is a* x-tagged border *for M, i, and α, if u is a border of $M[0..i]$ and $\texttt{GLB}(M, i, \alpha)$ such that*

1. $\texttt{Score}(M[0..\ell], ux) \geq \texttt{GLB}(M, \ell, \alpha)$, *and*
2. $\texttt{Score}(M[i-\ell..i], ux) < \texttt{GLB}(M, i, \alpha) - \texttt{MaxSc}(M[0..i-\ell-1])$,

where ℓ is the length of u.

We write Σ-$\texttt{TaggedBorder}(M, i, \alpha, x)$ the length of the longest x-tagged border of M, i, and α. The shift rule associated to x-tagged borders is specified by a two-entry table: one for the position, and one for the current character symbol. The table $\texttt{kmpNext}_\Sigma$ is defined as follows:

$$\texttt{kmpNext}_\Sigma[0][x] = -1$$
$$\texttt{kmpNext}_\Sigma[i][x] = \Sigma\text{-}\texttt{TaggedBorder}(M, i, \alpha, x), \qquad 1 \leq i \leq m$$

Lemma 9. $\texttt{kmpNext}[i] = \min_{x \in \Sigma} \texttt{kmpNext}_\Sigma[i][x]$

Proof. $\texttt{kmpNext}_\Sigma[i][x]$ gives the length of the longest border u which fulfills the two conditions of Definition 8. Hence, $\ell = \min_{x \in \Sigma} \texttt{kmpNext}_\Sigma[i][x]$ implies that there exists a border u of length ℓ and there exists a letter y in Σ such that the two conditions of Definition 8 are fulfilled. This is exactly the definition of $\texttt{kmpNext}[i]$. □

As we have done previously, we define a predicate that tells whether a x- tagged border exists or not. The difference compared to c' is that the letter x is a parameter of the predicate.

Definition 9 ($c''(M, p, i, \alpha, x)$). *Define the predicate $c''(M, p, i, \alpha, x)$ as true if, and only if, there exists a fixed letter tagged border of length $i - p$ for M, $i - 1$, α and x.*

$\texttt{kmpNext}_\Sigma[i][x]$ equals $i - p$, where p is the lowest value such that $c''(M, p, i - 1, \alpha, x)$ is true.

Lemma 10. *$c''(M, p, i, \alpha, x)$ is true if, and only if, there exist two scores β, δ such that*

1. $b(M, j, i - 1, \beta, \delta) = 1$, *and*
2. $\texttt{GLB}(M, i - 1, \alpha) - \texttt{MaxSc}(M[0..j - 1]) \leq \beta < \texttt{GLB}(M, i, \alpha) - \texttt{MaxSc}(M[0..j - 1]) - M(i, x)$, *and*
3. $\delta \geq \texttt{GLB}(M, i - j, \alpha) - M(i - j, x)$.

4.5 Looking for Occurrences Using mpNext, kmpNext or $\texttt{kmpNext}_\Sigma$

We now give the proof that the shifts induced by the mpNext, kmpNext or $\texttt{kmpNext}_\Sigma$ tables are safe: they do not allow to miss occurrences in the text.

Lemma 11. *Let M a PWM, α be a score threshold, T be a text, k be a position of T and i be a position of M. If the attempt k fails at position i of M, we know there is no occurrence for M and α for any position $k+j$ of T such that $1 \leq j \leq i - \texttt{mpNext}[i]$, or $1 \leq j \leq i - \texttt{kmpNext}[i]$, or $1 \leq j \leq i - \texttt{kmpNext}_\Sigma[i][T[k + i]]$.*

Proof. We give the proof for mpNext. The two other cases are similar. Suppose that there exists a position $k+j$, $1 \leq j \leq i - \texttt{mpNext}[i]$ such that $T_{k+j}..T_{k+j+m-1}$ is an occurrence. Then $\texttt{Score}(M, T_{k+j}..T_{k+j+m-1}) \geq \alpha$ and thus $\texttt{Score}(M[0..i - j], T_{k+j}..T_{k+i-j}) \geq \texttt{GLB}(M, i - j, \alpha)$. But necessarily $\texttt{Score}(M[j - 1..i - 1],$

$T_{k+j}..T_{k+i-j}) < \alpha - \texttt{MaxSc}(M[0..j-2])$. Otherwise $T_{k+j}..T_{k+i-j}$ would be a border of length $i-j+1$ of $M[0..i-1]$, which is impossible because $\texttt{mpNext}[i] < i-j+1$. This second inequality contradicts the fact that an attempt fails at position i of M. □

Contrarily to the original Knuth-Morris-Pratt searching phase, we cannot avoid to re-compute the score of the prefix against the text when shifting. This leads to a time complexity in $O(mn)$ in the worst case, instead of $O(m+n)$ for exact pattern matching. Nevertheless, there is still a strict improvement compared to the lookahead strategy algorithm.

4.6 Experimental Results

As mentioned in the introduction of this paper, PWMs are commonly used to model transcription factor binding sites. We measured the impact of the shifting rules on the Jaspar database [8]. Jaspar contains 123 PWMs that are derived from experimentally validated binding sites. The length of the PWMs ranges from 4 to 30. We computed the occurrences of each matrix from a random DNA sequence of 50 megabases. We used several P-values to define several score thresholds (see Section 3.1). Only matrices whose length allows to have at least one occurrence, $4^m \times p \geq 1$, for a given P-value p are used for each experiment in order to not artficially reduce the average running time.

We compared the running times of the naive algorithm (NA), the lookahead stategy algorithm (LSA), the expansions of the Knuth-Morris-Pratt algorithm using the $\texttt{kmpNext}$ table (KMP) and the $\texttt{kmpNext}_\Sigma$ table (KMB-AB). The algorithms have been implemented in C++ and a computer with a 2.33GHz Intel Core 2 Duo processor with 2 gigabytes of main memory has been used.

Table 1 gives the average searching phase running time for the four algorithms and the average preprocessing running time for KMP and KMP-AB. Figure 1 gives the total running time per matrix for P-value 10^{-4}. Algorithm KMP-AB achieves the best running time, outperforming the algorithm using the lookahead strategy. As we could expect, the running time increases when the P-value increases (the score threshold decreases) because the shifts become smaller. If one removes the longest matrix (of length 30) from the experiments, the preprocessing running time is almost the same for computing the $\texttt{kmpNext}$ table or the

Table 1. Average running times of the searching and preprocessing phases per matrix (in seconds) for several P-values

	p-value	$p = 10^{-7}$	$p = 10^{-5}$	$p = 10^{-3}$
	nb. of matrices	45	86	122
Searching phase	NA	3.80	3.22	2.81
	LSA	1.41	1.86	2.62
	KMP	0.95	1.13	1.35
	KMP-AB	**0.71**	**0.86**	**1.34**
Preprocessing phase	KMP	**0.019**	**0.023**	**0.043**
	KMP-AB	0.956	0.513	0.418

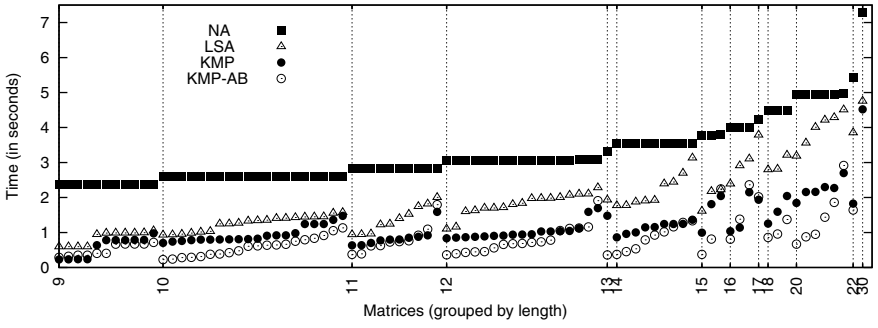


Fig. 1. Running times per matrix (in seconds) for P-value 10^{-5}

Table 2. Average searching plus preprocessing running times per matrix (in seconds) with the filtration technique

	$p = 10^{-7}$	$p = 10^{-5}$	$p = 10^{-3}$
KMP	**0.48**	**0.63**	**0.79**
KMP-AB	0.54	0.65	0.82

$\texttt{kmpNext}_\Sigma$ table. Without this matrix, the average preprocessing running time for computing $\texttt{kmpNext}_\Sigma$tables becomes respectively 0.009s, 0.014s and 0.062s for P-values 10^{-7}, 10^{-5} and 10^{-3}. The preprocessing running time increases with the P-value and the length of the matrices. The sum of the preprocessing and searching phase running times is almost always less than the LSA running time. Moreover, as the patterns used for finding transcription factor binding sites are permanent objects stored into databases, the computation need to be done only once.

When the shift computed thanks to the next tables equals 1, KMP and KMP-AB do not increase the efficiency of the searching phase. Let r be the rightmost position of a matrix such that all shifts between 0 and r equal 1. Instead of computing iteratively the score from 0 to r, we can directly compute the score of the prefix of length $r + 1$. This can be done thanks to a preprocessing phase which computes the scores for all words of length $r + 1$ and stores them into a table [12]. If r is not too large, the running time and the amount of memory used are small. Table 2 gives the average running time when we applied this refinement with r limited to 7 (preprocessing running time is less than 0.01s). KMP is the fastest algorithm, with a gain of almost 50% compared to KMP without filtration. KMP gives better results compared to KMP-AB because the value of r is lower with KMP-AB.

5 Conclusion

We extended the definition of a border for PWMs. This allowed to devise two algorithms directly derived from the Morris-Pratt and Knuth-Morris-Pratt algorithms and a new algorithm (KMP-AB). We then proposed to use a refinement

which, at the cost of a little amount of memory and preprocessing time, allows to get the best of the shifts and leads to a speed-up of almost three in practice compared to the LSA algorithm.

References

1. Mount, S.: A catalogue of splice junction sequences. Nucleic Acids Research 10, 459–472 (1982)
2. Hulo, N., Sigrist, C., Saux, V.L., Langendijk-Genevaux, P., Bordoli1, L., Gattiker, A., Castro, E.D., Bucher, P., Bairoch, A.: Recent improvements to the PROSITE database. Nucleic Acids Research 32, D134–D137 (2004)
3. Ewens, W., Grant, G.: Statistical Methods in Bioinformatics. Springer, Heidelberg (2005)
4. Pape, U., Rahmann, S., Sun, F., Vingron, M.: Compound poisson approximation of the number of occurrences of a position frequency matrix (PFM) on both strands. Journal of Computation Biology 15, 547–564 (2008)
5. Knuth, D., Morris Jr., J., Pratt, V.: Fast pattern matching in strings. SIAM Journal on Computing (1977)
6. Boyer, R.S., Moore, J.S.: A fast string searching algorithm. Commun. ACM 20, 762–772 (1977)
7. Aho, A., Corasick, M.: Efficient string matching: an aid to bibliographic search. Communications of the ACM (1975)
8. Sandelin, A., Alkema, W., Engström, P., Wasserman, W.: Jaspar: an open-access database for eukaryotic transcription factor binding profiles. Nucleic Acids Research (2004)
9. Wu, T.D., Nevill-Manning, C.G., Brutlag, D.L.: Fast probabilistic analysis of sequence function using scoring matrices. Bioinformatics 16, 233–244 (2000)
10. Staden, R.: Methods for calculating the probabilities of finding patterns in sequences. Comput. Appl. Biosci. 5, 89–96 (1989)
11. Touzet, H., Varré, J.S.: Efficient and accurate p-value computation for position weight matrices. Algorithms for Molecular Biology 2 (2007)
12. Liefooghe, A., Touzet, H., Varré, J.S.: Large scale matching for position weight matrices. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 401–412. Springer, Heidelberg (2006)
13. Beckstette, M., Homann, R., Giegerich, R., Kurtz, S.: Fast index based algorithms and software for matching position specific scoring matrices. BMC Bioinformatics (2006)
14. Pizzi, C., Rastas, P., Ukkonen, E.: Fast search algorithms for position specific scoring matrices. In: Hochreiter, S., Wagner, R. (eds.) BIRD 2007. LNCS (LNBI), vol. 4414, pp. 239–250. Springer, Heidelberg (2007)
15. Salmela, L., Tarhio, J.: Algorithms for weighted matching. In: Ziviani, N., Baeza-Yates, R. (eds.) SPIRE 2007. LNCS, vol. 4726, pp. 276–286. Springer, Heidelberg (2007)

Membership Testing: Removing Extra Stacks from Multi-stack Pushdown Automata

Nutan Limaye and Meena Mahajan

The Institute of Mathematical Sciences, Chennai 600 113, India
{nutan,meena}@imsc.res.in

Abstract. We show that fixed membership testing for many interesting subclasses of multi-pushdown machines is no harder than for pushdowns with single stack. The models we consider are MVPA, OVPA and MPDA, which have all been defined and studied in the past.

Multi-stack pushdown automata, MPDA, have ordered stacks with pop access restricted to the stack-top of the first non-empty stack. The membership for MPDAs is known to be in NSPACE(n) and in P. We show that the P-time algorithm can be implemented in the complexity class LogCFL; thus membership for MPDAs is LogCFL-complete.

It follows that membership testing for ordered visibly pushdown automata OVPA is also in LogCFL.

The membership problem for multi-stack visibly pushdown automata, MVPA, is known to be NP-complete. However, many applications focus on MVPA with $O(1)$ phases. We show that for MVPA with $O(1)$ phases, membership reduces to that in MPDAs, and so is in LogCFL.

1 Introduction

Pushdown machines are the machines having a finite control and access to a stack. The languages accepted by such machines are called context-free languages, CFLs. For a fixed machine M, given a string w, the membership problem asks whether $w \in L(M)$. It is of interest due to its implications for parsing and model checking problems. The first polynomial time algorithm for the membership problem for CFLs was given by Cocke, Kasami, and Younger (see for instance [1]).

Let us denote the membership problem for the class of languages $\mathcal{L}$ by MEM($\mathcal{L}$). If $\mathcal{A}$ is a class of automata accepting the language class $\mathcal{L}$, then we use MEM($\mathcal{L}$) and MEM($\mathcal{A}$) interchangeably.

The problems log-space many-one reducible to MEM(CFL) define a complexity class called LogCFL [2]. LogCFL is a subclass of P and is also known to be contained in NC, *i.e.* efficiently parallelizable.

The membership problem for many subclasses of CFLs has been studied rigorously. The set of languages log-space many-one reducible to MEM(DCFL), where DCFL denotes deterministic context-free languages, define the complexity class LogDCFL which is a subclass of LogCFL [2]. It is also known that the membership problems for linear and deterministic linear context-free languages, MEM(LIN)

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 493–504, 2009.

and MEM(DLIN), are complete for the complexity classes nondeterministic and deterministic log-space, NL and Log respectively [3,4]. Another interesting subclass of CFLs is visibly pushdown languages, VPLs ([5,6]). These are the languages accepted by visibly pushdown automata (VPA) which are ϵ-moves-free pushdown automata whose stack behaviour is dictated solely by the input letter under consideration. They are also referred to as input-driven PDA. MEM(VPL) is known to be complete for the class NC^1 [7] of languages accepted by families of polynomial-size log-depth bounded fan-in circuits.

A natural generalisation of pushdown machines is pushdown machines with more than one stack. This generalisation, unfortunately, is not smooth in terms of the power of these machines: A pushdown automaton with two or more stacks is known to recognise all recursively enumerable languages. The model in its full generality is thus intractable. However, for certain model checking applications, pushdown automata with two or more stacks are useful. Hence, some restrictions of multi-stack machines have been considered in the literature.

One possible restriction is a 2-stack VPA. One can consider various models depending on whether to allow simultaneous stack changes or depending on the order of accessing the stacks. Such models indeed have been considered recently (see *e.g.* [8,9]). A language-theoretic study, as well as the membership problem complexity for these models, are important.

Here we focus on the membership problem for three different models which have been defined and studied in the literature.

The first model is one recently considered by La Torre *et al.* [9]: a pushdown machine equipped with two stacks where the access to both the stacks is completely dictated by the input alphabet. This is a natural generalisation of VPLs and a proper restriction of general pushdown automaton having more than one stack. They call such machines multi-stack visibly pushdown machines, MVPA. In their definition, these machines cannot simultaneously access both stacks. On reading any input letter, the MVPA either pushes on one of the stacks or pops from one of the stacks. A *phase* of the input string is a substring such that while reading it, all the pop moves of the machine are on the same stack. In [9], it is shown that MEM(MVPL), where MVPL denotes the class of languages accepted by MVPA, is NP-complete. The proof of NP hardness is a reduction from an instance of SAT. For a fixed MVPA M, a string w is constructed from an n-variable formula such that it has n phases. That the number of phases depends on the input formula is important for the proof of hardness.

In this paper, we consider a restriction of the above problem, where the number of phases is a constant. We define another version of the membership problem, $\mathsf{MEM}(\mathsf{MVPL}_k)$. For a fixed MVPA M and fixed positive integer k, the problem $\mathsf{MEM}(\mathsf{MVPL}_k)$ is to decide whether a given $w \in \Sigma^*$ is in $L_k(M)$, where $L_k(M)$ denotes the language $\{w \in \Sigma^* \mid w \text{ is accepted by } M \text{ with } \leq k \text{ phases }\}$.

This restriction of MVPA, where the number of phases is bounded, is also useful for many applications and has been defined and considered in [9]. The class is known to generalise VPLs and is properly contained in context-sensitive languages. In this paper, we show that the problem $\mathsf{MEM}(\mathsf{MVPL}_k)$ is in LogCFL.

In order to show this, we need another model of multi-pushdown machines defined by Cherubini *et al.* [10]. They define a restriction of multi-pushdown machines wherein there is an order given to the stacks of the machine. The machine is allowed to push on any stack. However, pop moves are allowed only on the first non-empty stack. We denote such machines by PD_n, where n is the number of stacks in the machine. We denote the class of languages accepted by these as L_{PD_n}. A restriction of PD_n, namely PD_2, was studied in [11], where it was shown that $\mathsf{MEM}(\mathsf{PD}_2)$ is in P. Later, in [12], a P-time upper bound for $\mathsf{MEM}(\mathsf{PD}_n)$ was established. We give a reduction from $\mathsf{MEM}(\mathsf{MVPL}_k)$ to $\mathsf{MEM}(\mathsf{PD}_k)$.

We then prove a LogCFL upper bound for $\mathsf{MEM}(\mathsf{PD}_k)$. This improves the P-time upper bound of [12]. Also, combined with our previous reduction, this gives a LogCFL upper bound for the problem $\mathsf{MEM}(\mathsf{MVPL}_k)$. The languages accepted by MVPA within two phases are a proper subclass of context sensitive languages, a proper generalisation of VPLs, and are incomparable with CFLs. Hence, this implies the same upper bound as for CFLs for an incomparable class. However, we do not know if $\mathsf{MEM}(\mathsf{MVPA}_k)$ for any fixed k is hard for LogCFL.

Recently, Carotenuto *et al.* [8] defined another class of two-stack pushdown machines, 2-OVPA. Like MVPAs, these machines have a visible access to their stacks, *i.e.* the stack movement is completely dictated by the input alphabet. There is also an order among the stacks and the second stack is popped only if the first is empty. This model is interesting because emptiness and inclusion are decidable, and languages accepted by such machines form a Boolean algebra [8]. The generalisation where the number of stacks is k, k-OVPA, is also considered in [8]. The language class accepted is contained in L_{PD_k}. Thus, the LogCFL upper bound we prove also applies to this language class.

The main results of our paper can be summarised as follows:

Theorem 1. *For every fixed* $k \geq 1$, $\mathsf{MEM}(\mathsf{MVPL}_k) \leq \mathsf{MEM}(\mathsf{PD}_k)$.

Theorem 2. *For every fixed* $k \geq 1$, $\mathsf{MEM}(\mathsf{PD}_k)$ *is in* LogCFL.

Corollary 1. $\forall k \geq 1$, $\mathsf{MEM}(\mathsf{MVPL}_k)$ *and* $\mathsf{MEM}(k\text{-}\mathsf{OVPA})$ *are in* LogCFL.

2 Preliminaries

Circuits and Complexity. A *Boolean circuit* C_n on n inputs is a directed acyclic graph, with a designated sink (out-degree zero vertex) called the output gate. All the vertices except sources (in-degree zero vertices) are labelled by $\vee$ and $\wedge$. Sources are labelled by $\{0, 1\}$ or by predicates of the form $[x_i, a, 1, 0]$ where $i \in [n]$. Such a predicate takes the value 1 if $x_i = a$ and 0 otherwise[1].

A Boolean circuit C can be unwound into a tree T_C (by duplicating nodes). A *proof tree* T' of C on input w is a subtree of T_C with the following properties: (1) The output gate is in T'. (2) For every $\vee$-gate in T', one of its children is in

[1] This convention of labelling leaves with predicates is used, for *e.g.* , in [13], to deal with languages over non-binary alphabets.

T'. (3) For every $\wedge$-gate in T', all its children are in T'. (4) All the nodes in T' evaluate to 1 on input w.

A proof tree exists if and only if C_n accepts w. In general the proof tree could be of size exponential in n. C is said to have poly-proof-tree-size if whenever a string w is accepted by C, there is a proof tree on w of size poly($|w|$).

The complexity class LogCFL is the class of languages log-space many-one reducible to some CFL, and is known to be equivalent to the class of languages accepted by circuits having polynomial sized proof trees. See *e.g.* [2,14,15].

Visible two stack machines ([9]). An MVPA M is a pushdown machine having two stacks, where the access to the stacks is restricted in the following way: The input alphabet Σ is partitioned into 5 sets. A letter from Σ_c^j causes a push move on stack j, that from Σ_r^j forces a pop move on stack j, and both the stacks are left unchanged on letters from Σ_i. (The subscripts c, r, i denote call, return and internal respectively.) Formally, an MVPA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is a two-stack nondeterministic pushdown machine where Q is a set of finite states, Σ is the finite alphabet which is a union of 5 disjoint sets $\Sigma_c^0, \Sigma_r^0, \Sigma_c^1, \Sigma_r^1, \Sigma_i$, q_0 is the initial state, $F \subseteq Q$ is a set of final states, Γ is the finite stack alphabet containing a special bottom-of-stack symbol $\perp$ that is never pushed or popped, and δ has the following structure: $\delta_i \subseteq Q \times \Sigma_i \times Q$, and for $j \in \{0, 1\}$, $\delta_c^j \subseteq Q \times \Sigma_c^j \times Q \times \Gamma \setminus \{\perp\}$ and $\delta_r^j \subseteq Q \times \Sigma_r^j \times \Gamma \times Q$.

The machine is allowed to pop on an empty stack; that is, on reading a letter from Σ_r^j and seeing $\perp$ on the jth stack top, the machine can proceed with a state change leaving the $\perp$ untouched.

A phase is a substring of the input string $w \in \Sigma^*$ during which pop moves happen only on one of the stacks. Define the set

$$\mathsf{PHASE}_k = \{w \mid w \in \Sigma^*, \text{ number of phases in } w \leq k\}$$

Clearly, for any fixed partition of Σ, PHASE_k is a regular set.

Let M be a fixed MVPA M and k a fixed positive integer. Its k-phase language $L_k(M)$ is defined as $L_k(M) = L(M) \cap \mathsf{PHASE}_k$. By taking a direct product of a finite state automaton accepting PHASE_k with MVPA M, we can obtain an MVPA $M' = \langle M, k\rangle$ such that $L(M') = L_k(M') = L_k(M)$. In Section 3, we assume that the given MVPA M satisfies $L(M) = L_k(M)$.

Ordered multi-stack machines and grammars ([10,11,12]). A PD_k $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a k-stack pushdown machine where Q, Σ, q_0, Z_0, F are as usual, and the transition function δ is of the form $\delta \subseteq Q \times (\Sigma \cup \epsilon) \times \Gamma \times Q \times (\Gamma^*)^k$.

A *configuration* is a $(k+2)$-tuple, $\langle q, w, \gamma_1, \ldots, \gamma_k\rangle$ where $q \in Q$, $w \in \Sigma^*$, and $\gamma_i \in \Gamma^*$ for each i represents the contents of the ith stack. The *initial* configuration on word x is $\langle q_0, x, Z_0, \epsilon, \ldots, \epsilon\rangle$. A configuration is called a *final* configuration if $q \in F$.

If there is a transition $(q', \alpha_1, \ldots, \alpha_k) \in \delta(q, a, A)$, the machine in state q can read a letter a from the input tape, pop A from the first non-empty stack, push α_i on stack i for each $i \in [k]$, and move to state q'. Formally, $\langle q, aw, \epsilon, \ldots, \epsilon, A\gamma_i, \ldots, \gamma_k\rangle \vdash \langle q', w, \alpha_1, \ldots, \alpha_{i-1}, \alpha_i\gamma_i, \ldots, \alpha_k\gamma_k\rangle$.

If $(q', \alpha_1, \ldots, \alpha_k) \in \delta(q, \epsilon, A)$ then $\langle q, w, \epsilon, \ldots, \epsilon, A\gamma_i, \ldots, \gamma_k\rangle \vdash \langle q', w, \alpha_1, \ldots, \alpha_{i-1}, \alpha_i\gamma_i, \ldots, \alpha_k\gamma_k\rangle$.

The PD_k M accepts a string w if it can move from $\langle q_0, w, Z_0, \epsilon, \ldots, \epsilon\rangle$ to some $\langle q, \epsilon, \gamma_1, \ldots, \gamma_k\rangle$ where $q \in F$. The set of all the strings accepted by M is the language accepted by M, denoted $L(M)$.

Theorem 3. *([12]) For a fixed PD_k, given an input string $w \in \Sigma^*$, checking if $w \in L(M)$ is in P. i.e. $\mathsf{MEM}(\mathsf{PD}_k) \in \mathsf{P}$.*

In [11], PD_k are characterized by grammars. We describe the D^2-grammars that correspond to languages accepted by PD_2. A D^2-grammar G is a 4-tuple $G = (N, \Sigma, P, S)$ where N, Σ, S are as usual, and P has productions of the form: $A \to w(\alpha)(\beta)$ where $A \in N$, $w \in \Sigma^*$ and $\alpha, \beta \in N^*$.

Sentential forms in a derivation are of the form $x(\alpha)(\beta)$ where $x \in \Sigma^*$, $\alpha, \beta \in N^*$. The initial sentential form is $(S)(\epsilon)$. If $A \to w(\alpha)(\beta)$ is a production rule, then $w'(A\alpha')(\beta') \Rightarrow w'w(\alpha\alpha')(\beta\beta')$ and $w'(\epsilon)(A\beta') \Rightarrow w'w(\alpha)(\beta\beta')$ are the only valid derivations using this rule. Note that only *leftmost* derivations are allowed. We say that $A \Rightarrow^* w(\alpha)(\beta)$ if $(A)(\epsilon) \Rightarrow^* w(\alpha)(\beta)$ and that $A \Rightarrow^* w$ if $(A)(\epsilon) \Rightarrow^* w(\epsilon)(\epsilon)$. The language generated is the set $L(G) = \{w \mid S \Rightarrow^* w\}$.

Theorem 4. *([11]) Every D^2-grammar G has an equivalent normal form D^2-grammar G' where each production is of one of the following types:*

- $A \to (BC)(\epsilon); A, B, C \in N$ *(branching production)*
- $A \to (\epsilon)(B); A, B \in N$ *(chain production)*
- $A \to a; A \in N, a \in \Sigma$. *(terminal production).*

A derivation in such a grammar is said to be a *normal form derivation* if whenever a non-terminal A is rewritten by a chain production, say $A \to (\epsilon)(B)$, then that occurrence of B is eventually rewritten by either a branch production or a terminal production. That is, no variable participates in two chain rules. For every derivation, there is an equivalent normal form derivation [11].

A typical derivation in this grammar arising from the use of a branching production produces non-contiguous substrings. Say $A \to (BC)(\epsilon) \in P$. Also say $B \Rightarrow^* \beta_1(\epsilon)(\beta) \Rightarrow^* \beta_1\beta_2(\epsilon)(\epsilon)$ and $C \Rightarrow^* \gamma_1(\epsilon)(\gamma) \Rightarrow^* \gamma_1\gamma_2(\epsilon)(\epsilon)$. Then $A \Rightarrow (BC)(\epsilon) \Rightarrow^* \beta_1(C)(\beta) \Rightarrow^* \beta_1\gamma_1(\epsilon)(\gamma\beta) \Rightarrow^* \beta_1\gamma_1\gamma_2(\epsilon)(\beta) \Rightarrow^* \beta_1\gamma_1\gamma_2\beta_2(\epsilon)(\epsilon)$. Thus, we say that in the string $\beta_1\gamma_1\gamma_2\beta_2$, the substring $\beta_1\beta_2$ is produced by B with a *gap*, and the gap is filled by C with the substring $\gamma_1\gamma_2$.

A chain production does not explicitly give rise to a gap in the string. However, the application of a chain production swaps the order of substrings being produced by the non-terminals in the first list. Say $A \to (\epsilon)(B)$ and $B \Rightarrow^* \beta$; *i.e.* A produces a string β via a chain rule. Also say $C \Rightarrow^* \gamma$. Consider a sentential form $w(AC)(\delta)$. The string β produced by A appears in the final string *after* the string γ that is produced by C. That is, we get $w(AC)(\delta) \Rightarrow w(C)(B\delta) \Rightarrow^* w\gamma(\epsilon)(B\delta) \Rightarrow^* w\gamma\beta(\epsilon)(\delta)$. Hence when A produces a string via a chain production, we assume that β has a gap (of length 0) at the beginning (*before* β). Thus, a chain rule always results in a gap at the beginning.

Consider a terminal rule $A \to a$. Say A appears in some list in a sentential form. The terminal a produced by A appears before all the strings produced by all the non-terminals that follow A in its list. Consider sentential form $w(AC)(\delta)$. Then we get $w(AC)(\delta) \Rightarrow wa(C)(\delta) \Rightarrow^* wa\gamma$ where $C \Rightarrow^* \gamma$. Thus, a terminal production produces a gap (of length 0) at the end (*i.e. after* the terminal).

Ordered, visible two stacks machines ([8]). 2-OVPA are pushdown machines with two stacks, access to both of which is dictated by the input alphabet. The input alphabet Σ is a union of 8 disjoint finite sets: except for simultaneous pops on both stacks, all other combinations are allowed. Also the stacks are accessed in an ordered manner *i.e.* a pop is allowed on the second stack only if the first stack is empty. k-stack versions, k-OVPA, are defined similarly [8]. k-OVPA are essentially restrictions of PD_k, with the exception that they can also make moves when their stacks are empty. See [8] for formal definitions.

3 Reduction from $\mathsf{MEM}(\mathsf{MVPL}_k)$ to $\mathsf{MEM}(\mathsf{PD}_k)$

In this section, we consider the problem $\mathsf{MEM}(\mathsf{MVPL}_k)$ and establish Theorem 1. The simplest case is when $k = 1$; for all fixed MVPA M, $L_1(M) \in$ VPL. Since VPLs are known to be in NC^1 [7], for which membership in a fixed regular language is complete [16], $\mathsf{MEM}(\mathsf{MVPL}_1)$ reduces to $\mathsf{MEM}(\mathsf{NFA})$, where NFA are nondeterministic finite-state automata. But a PD_0 is precisely an NFA. Hence $\mathsf{MEM}(\mathsf{MVPL}_1)$ reduces to $\mathsf{MEM}(\mathsf{PD}_0)$.

For $k > 1$, we reduce this problem to $\mathsf{MEM}(\mathsf{PD}_k)$. As described in Section 2, we assume that $L_k(M) = L(M)$. We convert M into a multi-pushdown machine N having k stacks, called Main_i for $1 \le i \le k$, and show that $L(M)$ reduces to $L(N)$ (via logspace many-one reductions).

Consider a phase i in which stack-j ($j \in \{0, 1\}$) of machine M is being popped. The PD works in two stages – *mimic stage* and *buffer stage*. (Exception: phase k has only a mimic stage.)

In the Mimic stage, Main_i and Main_{i+1} contain the contents of stack j and $1 - j$ respectively and mimic the moves of machine M on these two stacks. The rest of the stacks are empty. (In particular for all $l < i$, Main_l are empty.) In the Buffer stage, Main_{i+1} is marked with a special symbol. The contents of Main_i are popped and are pushed onto top of the special symbol (in reversed order), and then popped and pushed into Main_{i+2}. Thus, the contents of Main_i are transferred into Main_{i+2} in the same order. Note that, the contents of Main_k need not be popped at all since there is no subsequent phase, and hence k stacks suffice in N.

To carry out these phases, the input string is padded with some new extra letters by a function f. On reading these letters, N does the necessary transfers. As the next phase expects to pop stack Main_{i+1}, after such a transfer all the stacks are ready for next processing step. More formally,

Lemma 1. *Fix a* MVPA M *and an integer* k. *There exist a* PD_{2k+2} N *and a function* $f \in \mathsf{Log}$, *such that* $\forall w \in \Sigma^*$, $w \in L_k(M) \Leftrightarrow f(w) \in L(N)$.

A small technical difficulty is that MVPAs are allowed pop operations on empty stacks, but PDs cannot make any move if all stacks are empty. If a prefix of an input string has unmatched pop letters (pops on empty stack), then during the mimic phase the simulating machine N may get stuck. To prevent this, we pad the input string with a suffiently long prefix that causes push moves on both the stacks. This boosts the heights of the stacks and ensures that the resulting string has no unmatched pop move. Formally, we show the following:

Lemma 2. *Fix a* MVPA M. *There exists another* MVPA M' *and a function* $g \in \mathsf{Log}$ *such that for every string* $w \in \Sigma^*$, $w \in L(M) \Leftrightarrow g(w) \in L(M')$, M *on* w *and* M' *on* $g(w)$ *have the same number of phases, and* M' *never pops or pushes on empty stack.*

We will call strings obtained by reduction g as *extended* strings and machine M' thus obtained a *good* MVPA. By Lemma 2, we assume that we have a good MVPA M that never uncovers the bottom-of-stack marker (except at the beginning) on either stack on the inputs that it receives.

For an extended string w, let $ht^j(w)$ denote the height of stack-j of a good MVPA M after having processed the string w. Here, $j \in \{0, 1\}$. To compute the function f in Lemma 1, we need the values $ht^j(x)$ for each prefix x of w. These values are easy to compute:

Proposition 1. *For any extended input string* w, *computation of* $ht^j(w)$ *and demarcation of the string into its first* k *phases can be done in* Log.

Suppose we have the extended string $w = w_1 w_2 ... w_k$ (on the extended alphabet Σ) already marked with the phases. That is, w_i is the string processed in the ith phase, and the individual strings $w_1, w_2, \ldots, w_k$ are known. Let k_i denote the height of the stack that was popped in phase i, after having processed the ith phase. We have ensured that $k_i \geq 1$ for all i. Let $U, V, W, Z, \#$ be new letters not in Σ. Then f is defined as below. (No padding is needed after w_k.) $f(w) = Zw_1\#U^{k_1+1}V^{k_1+1}\#Ww_2\#U^{k_2+1}V^{k_2+1}\#W \ldots w_i\#U^{k_i+1}V^{k_i+1}\#W \ldots w_k$.

For the PD_k $N = (Q', \Sigma', \Gamma', \delta', q_0', F')$, Q' consists of $3k$ copies of the states of M, 3 copies for each phase. The first copy is used during the mimic stage and the second and third copies are used for the first and the second steps in the buffer stage respectively. The padding symbol $\#$ is used in order to mark the stack Main_{i+1} with a special marker before the buffer-stage begins and then to pop the marker after the contents on top of it are moved into Main_{i+2}. Also Γ' consists of k copies of Γ, with the i-th copy used as the stack alphabet for Main_i.

Formally, the invariant maintained with respect to M can be stated as follows:

Lemma 3. *Machine* M *on input* w *has a non-deterministic path* ρ *in which for each* $i \in [k]$, *after phase* i *(where phase* i *pops stack* j*)* β_i *is on stack* j, α_i *is on stack* $1-j$ *and* M *is in state* q *if and only if machine* N *has a non-deterministic path* ρ' *along which for each* $i \in [k]$, *after reading the prefix up to and including* w_i *in* $f(w)$, (1) $\beta_i Z_0$ *is on* Main_i, (2) $\alpha_i Z_0$ *is on* Main_{i+1}, (3) *all the other stacks are empty, and* (4) *the state reached is* $[q^{(1)}, i]$.

That is, the runs of machines M *and* N *are in one-to-one correspondence.*

It follows that, M accepts w if and only if N accepts $f(w)$; hence Lemma 1.

4 The LogCFL Upper Bound for MEM(PD_k)

In this section, we show that membership testing for a fixed PD_k is in LogCFL.

The main structure of our LogCFL algorithm closely follows that of the P-time algorithm for membership testing for PD_2 as given in [11]. So we first describe it in some detail (Section 4.1), following the presentation from [17]. We then give (Section 4.2) a different implementation of the same algorithm and improve the upper bound to LogCFL, thus establishing Theorem 2 for $k = 2$. A P-time algorithm for MEM(PD_k) is given in [12]. It is very similar to the algorithm from [11]. In Section 4.3, we discuss the changes needed to be made in our implementation for the LogCFL bound to hold for all fixed k, thereby establishing Theorem 2.

4.1 Outline of the P-Time Algorithm of [11,17]

The P-time algorithm uses the characterization of PD_2 via D^2 grammars in normal form, and normal-form derivations, as described in Section 2. Given an input $w \in \Sigma^*$, the algorithm needs to keep track of substrings of w being produced with gaps. This is done as follows: A table T in constructed such that any entry in the table is indexed by four indices, $T(i, j, r, s)$. The algorithm fills entries in the table with subsets of N. A non-terminal A is in $T(i, j, r, s)$ if and only if A generates the string $w_{i+1} \dots w_j$ with a gap of length s at position $i + 1 + r$. Here r is the offset from $i + 1$ where the gap begins. The table entry $T(i, j, r, s)$ deals with the interval $inv = [i + 1, j]$ modulo the gap interval $gap = [i + r + 1, i + r + s]$. Let $l = j - i$ denote the total length of the interval and $l' = j - i - s$ denote the actual length of the interval under consideration *i.e.* length of the interval without the gap. The table is filled starting from smaller values of l. Further, the table entries with intervals of the same length l are filled starting from $l' = 1$ going up to $l' = l$. All entries are first initialized to contain the empty set.

For fixed values of l and l', we call a tuple $\langle i, j, r, s\rangle$ *valid for* l *and* l' if and only if $j = i + l$, $s = l - l'$ and $i + r + s \leq j$ (*i.e.* $r \leq l'$).

For $l = 1$ all the entries are filled by the following two rules, using information from the input and the fixed grammar.

1. $T(i, i + 1, 1, 0) = \{A \mid A \to w_{i+1}\}$
2. $T(i, i + 1, 0, 0) = \{A \mid A \to (\epsilon)(B), B \to w_{i+1}\}$

In the first (second) rule, the table entries correspond to intervals of size 1, where the zero-length gap is at the end (beginning, respectively). It contains the non-terminals that produce the terminal w_{i+1} using a terminal (chain, respectively) production.

As the value of l increases, depending on the position and size of the gap, various rules are used to fill up the table. For $l > 1$, the following rules are applied to fill the table entries corresponding to valid tuples:

Rule 1: This rule is applied provided the interval size is at least 2, and values of r', s' satisfy $r' < r, s < s' < j - i = l$.

$$T(i,j,r,s) = T(i,j,r,s) \cup \left\{ A \;\middle|\; \begin{array}{l} A \to (BC)(\epsilon), \\ B \in T(i,j,r',s'), \\ C \in T(i+r', i+r'+s', r-r', s) \end{array} \right\}$$

For this update, the algorithm uses values from $T(i,j,r',s')$ and $T(i+r', i+r'+s', r-r', s)$. These values are already available. To see this, note that for $T(i,j,r',s')$, the actual interval length is $j-i-s'$ which is strictly less than l' as $s' > s$, and for $T(i+r', i+r'+s', r-r', s)$, the interval length is s' where $s' < l$.

Rule 2: $T(i,j,0,s) = T(i,j,0,s) \cup \{A \mid A \in T(i+s,j,0,0)\}$. This rule is applied when the offset r is zero, *i.e.* when the gap is on the left. Note that this rule makes no update when the length s of the gap is zero.

For this update, the algorithm uses values from $T(i+s,j,0,0)$ (for which length of the interval $j-i-s < l$). This value is already available.

Rule 3: $T(i,j,r,s) = T(i,j,r,s) \cup \{A \mid A \in T(i,j-s,r,0)\}$. This rule is applied when the gap of length s is on the right. This happens when the gap stretches all the way till j, *i.e.* $i+r = j-s$. Note that this rule makes no update when the length s of the gap is zero.

For this update, the algorithm uses values from $T(i,j-s,r,0)$ (for which length of the interval is $j-s-i < l$). These values are already available.

Rule 4: $T(i,j,0,0) = T(i,j,0,0) \cup \{A \mid A \to (\epsilon)(B), B \in T(i,j,r',0)\}$. This rule is applied when s and r are both zero. And $0 \le r' \le j-i$.

For this update, the algorithm uses values from $T(i,j,r',0)$ checking if $A \to (\epsilon)(B)$ and $B \in T(i,j,r',0)$ for some $0 \le r' \le j-i$. Now for $T(i,j,r',0)$, the l and l' values are the same as that for $T(i,j,0,0)$. So we cannot immediately conclude that the required values are already available. However, for fixed l, l', the P-time algorithm performs steps $1,2,3$ before the step 4. Steps $2,3$ leave entries unchanged if $s = 0$. It is sufficient to argue that step 1 in fact puts B in $T(i,j,r',0)$, which is then used in step 4. Suppose not. *i.e.* suppose B is written in $T(i,j,r',0)$ by rule 4. Let $r' = 0$, as rule 4 cannot have been applied if $r' \neq 0$. Also as B is written in $T(i,j,r',0)$ by rule 4, there exists a $C \in N$ and a rule $B \to (\epsilon)(C)$ such that $B \Rightarrow (\epsilon)(C) \Rightarrow^* w_{i+1} \ldots w_j$. But then the complete derivation is $A \Rightarrow (\epsilon)(B) \Rightarrow (\epsilon)(C) \Rightarrow^* w_{i+1} \ldots w_j$. This contradicts the assumption that we have a normal form derivation. Hence, the required values are already available even for this step.

After a systematic looping through these indices, finally the entry of interest $T(0,n,0,0)$ is filled. If $S \in T(0,n,0,0)$, then the algorithm returns 'yes', else it returns 'no'. The time complexity of the algorithm is $O(n^6)$.

4.2 The LogCFL Algorithm ($k = 2$)

We now give a top-down algorithm to fill up the table T. We will see that it can be implemented by a poly sized circuit having $\wedge$ and $\vee$ gates and having poly sized proof trees. From [14,2] it follows that this algorithm is in LogCFL.

The polynomial time algorithm that fills up the table can be viewed as a polynomial sized circuit. However, this circuit need not have polynomial size

proof trees. In particular, the index computations may blow up the proof-tree size. We note that these index computations are independent of the input, and give a way to build a circuit with small proof-trees.

For each $l' \leq l \leq n$, for all valid tuples corresponding to these values of l, l', and for each $A \in N$, we introduce 5 gates: an OR gate $\langle A, i, j, r, s\rangle$ called a *main* gate, and 4 intermediate gates $X^1_{A,i,j,r,s}$, $X^2_{A,i,j,r,s}$, $X^3_{A,i,j,r,s}$, $X^4_{A,i,j,r,s}$ called *auxiliary* gates. We design the circuit in such a way that $\langle A, i, j, r, s\rangle = 1$ if and only if $A \in T(i, j, r, s)$. The root of the circuit is labelled $\langle S, 0, n, 0, 0\rangle$. The circuit connections are as follows:

$$\langle A, i, j, r, s\rangle = \bigvee_{k\in[4]} X^k_{A,i,j,r,s}$$

$$X^1_{A,i,j,r,s} = \bigvee_{\substack{r' < r \\ s < s' < j-i-r+1 \\ \{B,C|\ A\to(BC)(\epsilon)\}}} \begin{pmatrix} \langle B, i, j, r', s'\rangle \bigwedge \\ \langle C, i+r', i+r'+s', r-r', s\rangle \end{pmatrix}$$

$$X^2_{A,i,j,r,s} = \begin{cases} \langle A, i+s, j, 0, 0\rangle & \text{if } r = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$X^3_{A,i,j,r,s} = \begin{cases} \langle A, i, j-s, r, 0\rangle & \text{if } i+r = j-s \\ 0 & \text{otherwise} \end{cases}$$

$$X^4_{A,i,j,r,s} = \begin{cases} \bigvee_{0\leq r'\leq j-i,\{B|\ A\to(\epsilon)(B)\}} X^1_{B,i,j,r',0} & \text{if } r, s = 0 \\ 0 & \text{otherwise} \end{cases}$$

This finishes the description of all the non-leaf gates. The input gates are predicates and their values are propagated via the following depth-1 circuit.

$$\langle A, i, i+1, 1, 0\rangle = \bigvee_{\{a|\ (A\to a\in P)\}} [i, a, 1, 0]$$

$$\langle A, i, i+1, 0, 0\rangle = \bigvee_{\{a|\ \exists B(B\to a\in P)\wedge(A\to(\epsilon)(B)\in P)\}} [i, a, 1, 0]$$

Note that the above connections give an acyclic digraph of depth $O(n^2)$.

It is now easy to see the following claim, and hence the correctness of the above circuit follows from the correctness of P-time algorithm.

Lemma 4. *$\langle A, i, j, r, s\rangle = 1$ if and only if $A \in T(i, j, r, s)$.*

The LogCFL bound for MEM(PD_2) now follows from the following claim:

Claim. The circuit constructed above has polynomial-size proof-trees.

4.3 The LogCFL Algorithm for MEM(PD_k)

The grammars [10] corresponding to PD_k have rules with a single non-terminal belonging to one of the k lists on the left hand side and at most k lists of non-terminals on the right hand side. The normal form of the grammar is as follows:

– $(A)_h \to (BC)_1$; $k \geq h \geq 1$ (branch production; always expands into list 1)
– $(A)_h \to (B)_g$; $k \geq g > h \geq 1$ (chain production; from list h to a later list g)
– $(A)_h \to a$; $a \in T$; $k \geq h \geq 1$ (terminal production)

Now, any typical string derived by a non-terminal can have as many as 2^{k-1} gaps; see [12]. If $(A)_h \to (BC)_1$ is a branch rule, and B, C derive strings γ and δ respectively, then the string derived from A is a systematic merge of γ and δ. In the case when $k = 2$, only one gap was possible, whereas here we need to keep track of 2^{k-1} gaps to interleave γ and δ properly. Arrays $\tilde{r}$ and $\tilde{s}$, of length 2^{k-1} each, keep track of the off-sets and the lengths of the gaps.

Each table entry is indexed by $i, j, \tilde{r}, \tilde{s}$, as in the case $k = 2$. But now the tables are $2^k + 2$ dimensional (as each $\tilde{r}$ and $\tilde{s}$ are 2^{k-1} length arrays). The table entries contain non-terminals and they are filled in such a way that a non-terminal A belongs to a certain entry $T_{i,j,\tilde{r},\tilde{s}}$ if and only if the string $w_i \ldots w_j$ with gap off-sets as in $\tilde{r}$ and gap sizes as in $\tilde{s}$ can be obtained from A. The rules for filling up the table are slightly more complicated. However, they simply involve some index manipulations. These can be implemented as we did for $k = 2$. Once these rules are established, the order of filling up the entries and hence the rest of the algorithm is exactly the same. Thus, we obtain Theorem 2.

4.4 Bounds for MVPA and OVPA

From Theorems 1 and 2, it follows that $\mathsf{MEM}(\mathsf{MVPA}_k)$ is in LogCFL.

To see this bound for $\mathsf{MEM}(k\text{-OVPA})$, Theorem 2 should suffice, since as claimed in [8], k-OVPA are a special case of PD_k. However, there is a slight subtlety here. k-OVPA are allowed to "pop" on an empty stack: if all stacks are empty (they contain only the special letter $\perp$), then the k-OVPA can still proceed with its computation even on a return letter. However, a PD_k in a similar configuration is stuck and cannot make any move. So it is technically not completely correct to say that k-OVPA are PD_k. However, we can handle this exactly as we did for MVPA in Lemma 2, reducing $\mathsf{MEM}(k\text{-OVPA})$ to $\mathsf{MEM}(\mathsf{PD}_k)$.

5 Discussion

Our results show that adding more stacks to a PDA does not make the fixed membership problem harder than that for ordinary pushdown automata if stack access is restricted to visible behaviour with $O(1)$ phases, or if the the stacks are ordered and the stack pop access is restricted to the first non-empty stack.

Some interesting questions remain unanswered: What complexity classes are characterized by $\mathsf{MEM}(\mathsf{MVPA}_k)$ and $\mathsf{MEM}(\mathsf{OVPA})$? These problems lie somewhere between NC^1 and LogCFL. And what is the complexity of the general membership problem for these models, where the machine and the word are both part of the input?

Acknowledgments. The authors are grateful to the referees for presentation-related comments, and for noting that Theorem 1 needs only k stacks, not $2k-2$.

References

1. Hopcroft, A., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (2001)
2. Sudborough, I.H.: On the tape complexity of deterministic context-free language. Journal of Association of Computing Machinery 25(3), 405–414 (1978)
3. Sudborough, I.H.: A note on tape-bounded complexity classes and linear context-free languages. Journal of Association of Computing Machinery 22, 499–500 (1975)
4. Holzer, M., Lange, K.J.: On the complexities of linear LL(1) and LR(1) grammars. In: 9th International Symposium on Fundamentals of Computation Theory FCT, London, UK, pp. 299–308. Springer, Heidelberg (1993)
5. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: 36th ACM Symposium on Theory of Computing (STOC 2004), pp. 202–211 (2004)
6. Mehlhorn, K.: Pebbling mountain ranges and its application to DCFL recognition. In: 7th International Colloquium on Automata, Languages and Programming, pp. 422–432 (1980)
7. Dymond, P.W.: Input-driven languages are in $\log n$ depth. Information Processing Letters 26, 247–250 (1988)
8. Carotenuto, D., Murano, A., Peron, A.: 2-visibly pushdown automata. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 132–144. Springer, Heidelberg (2007)
9. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: 22nd Symposium on Logic in Computer Science, pp. 161–170 (2007)
10. Cherubini, A., Breveglieri, L., Citrini, C., Crespi Reghizzi, S.: Multipushdown languages and grammars. International Journal of Foundations of Computer Science 7(3), 253–292 (1996)
11. Cherubini, A., Pietro, P.S.: A polynomial-time parsing algorithm for a class of non-deterministic two-stack automata. In: 4th Italian Conference on Theoretical Computer Science, pp. 150–164 (1992)
12. Cherubini, A., Pietro, P.S.: A polynomial-time parsing algorithm for k-depth languages. Journal of Computer and System Sciences 52(1), 61–79 (1996)
13. Allender, E., Jiao, J., Mahajan, M., Vinay, V.: Non-commutative arithmetic circuits: depth reduction and size lower bounds. Theoretical Computer Science 209, 47–86 (1998)
14. Ruzzo, W.: Tree-size bounded alternation. Journal of Computer and System Sciences 21, 218–235 (1980)
15. Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer, New York (1999)
16. Barrington, D.: Bounded-width polynomial size branching programs recognize exactly those languages in NC^1. Journal of Computer and System Sciences 38, 150–164 (1989)
17. Pietro, P.S.: Two-stack automata. Rapporto Interno n. 92-073, Dipartimento Di Elettronica e Informazione, Politecnico di Milano, Milano (October 1992), http://home.dei.polimi.it/sanpietr/pubs/twostack92.ZIP

Automata on Gauss Words

Alexei Lisitsa, Igor Potapov, and Rafiq Saleh

Department of Computer Science,
University of Liverpool, Ashton Building,
Ashton St, Liverpool L69 3BX, UK

Abstract. In this paper we investigate the computational complexity of knot theoretic problems and show upper and lower bounds for planarity problem of signed and unsigned knot diagrams represented by Gauss words. Due to the fact the number of crossing in knots is unbounded, the Gauss words of knot diagrams are strings over infinite (unbounded) alphabet. For establishing the lower and upper bounds on recognition of knot properties we study these problems in a context of automata models over infinite alphabet.

1 Introduction

Algorithmic and computational topology is a new growing branch of modern topology. Much of the recent effort has focused on classifying the inherent complexity of topological problems. In this paper we investigate the computational complexity of knot theoretic problems and show upper and lower bounds for planarity problem of signed and unsigned knot diagrams. The main goal of proposed approach is to give a new insight on knot problems and characterise knot problems according to their computational complexity. The results presented in this paper were achieved by a combination of methods from knot theory, automata theory and computational complexity.

Knot theory is the area of topology 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 an ambient space. It is well established and exciting area of mathematical research with strong connections with topology, algebra and combinatorics. Examples of interactons between knot theory and computer science include works on formal language theory [1], quantum computing [2,3,4] and computational complexity [5].

Knots can be described in various ways, including various discrete representations. For example, a common method of describing a knot is a planar diagram called a knot diagram. A knot diagram is a projection of the knot onto a plane, where at each crossing we must indicate which section is "over" and which is "under", so as to be able to recreate the original knot. As an additional information we can also add a label ("'+" or "-") to each of the crossing for representing orientation of its strands.

A knot diagram can be encoded as a string of symbols O_i's (over) and U_i's (under) also known as *Gauss word*. The procedure of writing a Gauss word can be described as follows: Starting from a base point on the circle, write down the

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 505–517, 2009.

labels of the crossings in the counterclockwise direction, e.g. the trefoil K can be defined by a Gauss word ${U_1}^+{O_2}^+{U_3}^+{O_1}^+{U_2}^+{O_3}^+$, where indices indicate an (arbitrary) order of the crossings in the knot diagram and signs stand for orientation of each crossing. Likewise, links can be represented by *several* Gauss words - one for each component of the link.

As you can see the construction of the Gauss word is quite straightforward by reading visited crossings travelling along a circle. The inverse problem of constructing a knot from some strings of symbols from the set of O_i's and U_i's is harder and it is not always possible. It may happen that some Gauss words would not correspond to any classical (planar) diagrams. In such case we say that a Gauss word corresponds to a non-planar knot where any of its diagram should contain virtual crossings (i.e. which are not listed in the Gauss word). Most of the problems of recognising knots properties (such as virtuality, unknottedness, equivalence) are known to be decidable, with different time complexity. However their complexity in terms of computational power of devices needed to recognise the knot properties was not studied yet. In this paper we address this problem and provide first known bounds for some knot problems in this context.

The central problem which we are studying in this paper is to determine whether a given Gauss word corresponds to planar or non-planar knot. Due to the fact the number of crossing in knots is unbounded, the Gauss words of knot diagrams are strings over infinite (unbounded) alphabet. In this context we cannot estimate computational complexity in terms of classical models over finite alphabets and need to consider a new hierarchy of languages and models over infinite alphabet. Such models were recently introduced in [6,7].

In Section 2 we describe and extend the models of automata over infinite alphabet that we used for establishing the lower and upper bounds on recognition of knot properties. Then in Section 3 we show that the language of planar (non-planar) signed Gauss words can be recognised by deterministic two-way register automata by simulation of recently discovered linear time algorithm proposed in [8]. Due to the fact that the algorithm presented in [8] allows to check planarity property not only for knots but also for links we think that the proposed idea of recognising planarity by register automata can be extended for links after some minor modification. The result is final in a sense that the power of non-deterministic one-way register automata is not even enough to recognise whether an input is a Gauss word. We also conjecture that planarity problem for unsigned Gauss words is harder than the the same problem for signed Gauss words and cannot be solved by register or k-pebble automata over infinite alphabet. For the case of unsigned Gauss words we provide the upper bound by showing that planarity can be checked by deterministic linearly bounded memory automata.

2 Automata over Infinite Alphabets

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 $d_1, \dots d_n$ where $d_i \in D$, $i = 1, \dots, n$. 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. A language L over an infinite alphabet D is called *n-bounded* if and only if there is a constant $n \in N$ such that for any $w \in L$ and for any $d \in D \mid w \mid_d \leq n$. All languages we consider are bounded languages.

2.1 Register Automata

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 introduced in [6] and studied further in [7].

Definition 1 ([7]). *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,* $\tau_0 : \{1,\ldots,k\} \rightarrow D \cup \{\triangleright, \triangleleft\}$ *is the initial register assignment and* P *is a finite set of transitions of the forms:*

1) $(i, q) \rightarrow (q', d)$ (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 specified direction d;)
2) $q \rightarrow (q', i, d)$ (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 $d \in \{stay, left, right\}$.)

Given a D-word d delimited by symbols $\triangleright, \triangleleft$ on the input tape an automaton starts in a state q_0 and in the position of the first letter of d and applies non-deterministically any applicable rules. As usual, if automaton is able ever to reach a state $q \in F$ it accepts the word, otherwise the word is rejected. The set of all accepted words forms a language recognisable by an automaton. An automaton is *deterministic* if in each configuration at most one transition applies.

For the purpose of this paper we modify the definition of register automata from [7] by allowing more general transition rules that allows replication of the same value in different registers. This does not affect the computational power of the model (see Lemma 1 below), but makes the design of such automata for various recognition problems much more natural and easier. Similar modifications (in more general setting) have appeared in [9,10].

We define *modified* two-way k-register automata by adding to the definition above two extra types of transitions rules:

3) $(i, q) \rightarrow (q', j, d)$ 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 specified direction d;
4) $q \rightarrow (q', d)$ 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 specified direction d, where $i, j \in \{1, .., k\}, q, q' \in Q$ and $d \in \{stay, left, right\}$.

Lemma 1. *The models of original register automaton and modified register automaton over an infinite alphabet are equivalent.*

Proof. We show that two extra types of rules of the modified model can be simulated by the original automata. The rule of type 4 is simulated by adding one extra dummy register $k+1$ and replacement of rules of modified automata of the form $q \to (q', d)$ by a pair of rules $(k+1, q) \to (q'd)$ and $q \to (q', k+1, d)$. If a value of the register $k+1$ is equal to the observed symbol then the rule $(k+1, q) \to (q'd)$ is applicable otherwise the rule $q \to (q', k+1, d)$ is applicable. Also we replace the rule $q \to (q', i, d)$ of type 2 in the modified automata model by a pair of rules; original type 2 rule $q \to (q', i, d)$ and type 3 rule $(k+1, q) \to (q'd)$.

The rule of type 3 $(i, q) \to (q', j, d)$ of the modified model that allows storing the same value in different registers, can be simulated in original model by using the following construction.

The state of the registers of the modified automaton, that is a sequence of not necessarily different values $R = [r_1, r_2, \ldots, r_{k+1}]$ is represented in the simulating automaton as a pair:

- the set of unique values $U = \{r_1, r_2, \ldots, r_{k+1}\}$, and
- the surjective mapping $\phi : \{1, \ldots, k+1\} \to U$

The content of U is kept in the registers and since the mapping ϕ is finite, it can be kept in the finite state control. Now it is straightforward to simulate the effects of all possible types of rules, including the type 3, in terms of pairs U, ϕ. We omit obvious details. □

Pebble Automata. As an alternative model of automata over infinite alphabet, *pebble automata*(PA) was introduced in [11] and further studied in [7]. We follow the definitions in [7]. In this model, instead of registers, finite state machines are equipped with the finite set of pebbles which can be placed on the input string and later lifted following the *stack* discipline. That means pebbles are numbered from 1 to k and pebble $i+1$ can only be placed when pebble i has already been placed on the string and vice-versa, pebble i can only be lifted if $i+1$ is not on the string. Further assumption is that the pebble with the highest number acts as a head, so an automaton has an access to the symbol of the string under such a pebble and to the information on which other pebbles are located at the same position. The transition of pebble automata depends on the following: the current state, the pebbles placed on the current position of the head, the pebbles that see the same symbol as the top pebble. The effect of the transition is the change of a state, movement of the head and, possibly, removal of the head pebble, or placement of the new pebble. As usual acceptance of a word is defined as reachability of one of the final states.

As expressive power concerned, in general pebble automata are incomparable with register automata [7]. We will show, however, in the Section 3 that over a class of *bounded languages*, including all languages of our interest, PA can be effectively simulated by RA.

Linearly Bounded Memory Automata. In all models above the input can be thought of as given on the input tape which can only be read, but not written on. Linearly bounded automata (LBMA) is an extension of register automata with the input tape. The automaton can read and write in the tape cells the symbols of an infinite alphabet. The input is given on the initial part of the tape and for the input size n, the size of the tape is assumed to be $O(n)$, i.e. linearly bounded. Types of rules of LBMA include all types of rules of (modified) RA and additional rules allowing to write on the tape. For every form $L \to (\ldots)$ of rules of the (modified) RA model the following is a form of rule for LBA: $L \to (\ldots, i)$, where $i \in \{1, \ldots k\}$. The effect of application of the latter is the same as of the former, plus the automaton *writes the content of the register* i in the current position on the tape before possible head movement.

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* [12]. 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. For example, the rule $(c, i, q) \to (q', d)$ is read as "if an automaton is in a state q and observes the symbol (c, a) and a is the content of the register i then the automaton can change the state to q' and move the head along the direction d." It should be clear now how to modify the definitions of all above models to work over data words.

3 Recognisability of Knot Properties

3.1 The Language of Gauss Words

Knots which are defined as embeddings of a circle in 3-dimensional Euclidean space can be faithfully represented by finite structures, such as graphs or words. One of such discrete representations is a Gauss code, which is a word of crossing labels O ("over") and U ("under") with appropriate indices, which can be read of the projection of the knot on the plane. Given orientation of the plane one can distinguish between left and right-handed crossings (see Figure 1) which are labelled by signs $-$ and $+$, respectively.

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 Gauss word w is a word over alphabet $\mathbb{N}$ (i.e. finite sequence of natural numbers) such that for every $n \in \mathbb{N}$ either $|w|_n = 0$ or $|w|_n = 2$*

Shadow Gauss words can be seen as the projections of (un)signed Gauss words on the second components of their symbols. A (unsigned, signed, shadow) Gauss language is an arbitrary set of (unsigned, signed, shadow) Gauss words.

We denote the data language of all unsigned Gauss words by L_{GW} and the language of all signed Gauss words by L_{SGW}.

Proposition 1. *The languages L_{GW} and L_{SGW} are recognisable by deterministic 2-way register automata.*

Proof. We explain only the construction of a 2-way deterministic register automaton A which recognises L_{GW}. With obvious modifications the automaton can be adapted to the case of L_{SGW}. Let w be a data word $(a_1, b_1)...(a_n, b_n)$ such that $a \in \Sigma = \{U, O\}$ and $b \in N = \{1, ..., n\}$. The automaton A reads the first symbol (a_i, b_i) and stores the value of b_i in some register, then it moves right then left along the word to compare the current symbol (a_j, b_j) with the value of b_i held in some register. If the symbol (a_j, b_j) where $\mathrm{b}_i = b_j$ and $\mathrm{a}_i \neq a_j$ is found and there are no further occurrences of b_i then automaton A moves right along the word and checks next symbol. If next symbol is equal to end symbol then A moves to an accepting state.

Proposition 2. *The languages L_{GW} and L_{SGW} are not recognisable by nondeterministic one-way register automata.*

Proof. We show the argument only for the case of L_{GW}. With obvious modifications it works for L_{SGW} as well. The argument is not new and was used e.g. in [12] to show non-recognizability of some data languages by one-way register automata. Assume that language L is recognisable by some one-way register automaton A with n registers. Consider the word $w = (U, 1)(U, 2)\dots(U, n + 1)(O, 1)(O, 2)\dots(O, n + 1) \in L$. The automaton A accepts this word. After

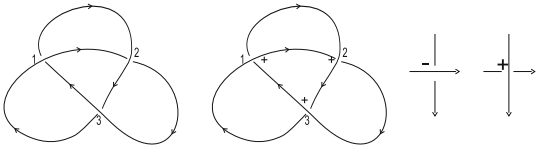


Fig. 1. Example of a knot diagram with its corresponding Gauss words(signed and unsigned)

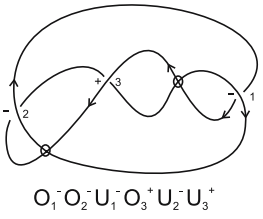


Fig. 2. Example of a virtual knot

reading first $n+1$ positions there is at least one index value $i \in \{1,\ldots,n+1\}$ which does not appear in any register of R. That means that automaton A also accepts a word $w' \notin L$ which obtained from w by replacing (U,i) with $(U,i+1)$. That is in contradiction with an assumption on A.

3.2 Planar and Non-planar Gauss Words

Every knot can be represented by a Gauss word but not every Gauss word represents a knot. For example, any attempt to reconstruct a knot diagram from the Gauss word $O_1^- O_2^- U_1^- O_3^+ U_2^- U_3^+$ will lead to new (*virtual*) crossings which are not present in the Gauss word (see Figure 2). Such an observation was one of the motivations for introducing *virtual knot theory* [13]. The 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 problem of recognition of planar Gauss words have been formulated by Gauss himself and recently several algorithmic solutions for both signed and unsigned case have been proposed, e.g. in [14,15,13,8].

In this section we address the question of recognisability of planarity of Gauss words by automata models and consider unsigned and signed case separately.

Signed Gauss Words. In this subsection we will show that a language of planar signed Gauss words can be recognised by two-way deterministic register automata. The design of automata will be based on a linear time algorithm, presented by V. Kurlin in [8]. The main idea of the algorithm is computing the least genus of the surface to which a knot diagram is embeddable without virtual crossings. For this purpose the Euler characteristics χ of the *combinatorial cell complex (Carter surface)* [16] associated with the knot diagram is computed. Computation of the Euler characteristics χ requires counting the number of faces (cycles), edges and vertices of a combinatorial cell complex (Carter surface) associated with a knot diagram. In the context of Gauss word, the number of edges corresponds to the length of the word, the number of vertices corresponds to the number of crossings (that is half the length of the word) and the number of faces can be determined by implementing a set of traversal rules.

Lemma 2. *Two-way deterministic register automaton can traverse all faces of (a complex associated with) a knot diagram which are adjacent to a crossing i.*

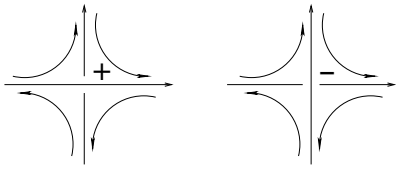


Fig. 3. Automaton moves

Proof. The traversal of adjacent face to a crossing i in the knot diagram can be done by choosing initial direction and turning left on each consecutive visited crossing staring from i. This global property of "turning left" can be defined by deterministic set of traversal rules which will take into account only local property of the current crossing and a finite information about previously visited one. In general we have 8 cases since there are two types of crossings (with a sign "+" and with a sign "-") and four directions from which we can approach each crossing, see Figure 3. We follow interpretation of the local rules for selecting cycles defined in [8], but present them here in slightly different notation, which is more appropriate for the design of a register automaton. A register automaton that is observing a current symbol S, needs to choose a correct symbol which correspond to the next crossing after turning (geometrically) to the left on a knot diagram. In fact on the Gauss code it will correspond to finding S' which is the counterpart of S and then choosing a symbol which is either left or a right neighbour of S'. For example, if S is O_i (U_i) then we need to choose a neighbour of U_i (O_i) in the Gauss word. In order to define whether we need a symbol from the left or from the right side we need to know the current type of the crossing which is S and the information about the previous choice of direction, i.e. whether S was chosen as a left or a right symbol. Now we define eight rules in the form $(D, S) \rightarrow (S', D')$ where $D, D' \in \{Right, Left\}$ and $S, S' \in \{U, O\}\times\mathbb{N}\times\{+, -\}$ where $\mathbb{N} \in$\{1,2...,n\}. Each rule can be read as follows: if the current symbol S is reached via direction D then find S' (counterpart of S) and move one step to the specified direction.

(Right, O_i^+)$\rightarrow$ (U_i^+, Right)	(Right, U_i^+)$\rightarrow$ (O_i^+, Left)
(Left, O_i^+)$\rightarrow$ (U_i^+, Left)	(Left, U_i^+)$\rightarrow$ (O_i^+, Right)
(Right, U_i^-)$\rightarrow$ (O_i^-, Right)	(Right, O_i^-)$\rightarrow$ (U_i^-, Left)
(Left, U_i^-)$\rightarrow$ (O_i^-, Left)	(Left, O_i^-)$\rightarrow$ (U_i^-, Right)

Following above rules we can design a register automata that will keep the finite information about its previous choice of direction (Right or Left) in its state space and which will choose the Right or Left symbol of S' observing the symbol S. It can also keep records on which rule was applied to the starting symbol S and will terminate the traversal of a face if the same rule will be applied for S again. The fact of the repetition corresponds to the completion of a cyclic path. In order to traverse all faces which are adjacent to a crossing i we need to start from two different initial conditions associated with labels (O_i or U_i) and two different initial direction (Left or Right). □

Lemma 3. *Two-way register automata with k registers on the input with t distinct symbols can simulate a counter machine with k counters bounded by t.*

Proof. Let us assume that a word on an input tape has at least t distinct symbols. The value of a counter stored in register i corresponds to the number of distinct symbols from the beginning of the word till the position of the first appearance of symbol stored in the register. Then we can increase (decrease) the value by looking for the next (previous) symbol on the string that will appear for the first time. Counter i is equal to zero if the value stored in the register i is the first symbol on the input tape. If we use k registers then we can store k counters bounded by t, where t is a number of distinct symbols on the input tape. □

Lemma 4. *Two-way deterministic register automata can compute the Euler characteristics of a signed Gauss word.*

Proof. In order to compute Euler characteristics we need to count the number of faces in a degree 4 graph G represented by a signed Gauss word, the number of edges and the number of vertices in G. The number of vertices in G is a number of distinct symbols, and the number of edges is the number of symbols in the Gauss word. Both values can be counted in a straightforward way. The number of faces can be counted by traversal of G in the following way. The automaton goes sequentially through the list of vertices. For each vertex i it traverses (as described in Lemma 2) all adjacent faces and increases a counter by one for every face F in which there are no vertices of smaller than i labels. Also the automaton counts how many times a crossing i is met during the traversal of faces adjacent to i. As soon as the value reaches 4 the automaton starts the traversal for the next crossing. The computation of the Euler characteristics χ is done by counting the values for edges, vertices and faces in individual counters and then by subtracting number of edges from the sum of the numbers of vertices and faces. Since the number of each value in counters is bounded by the number of distinct symbols the computation can be done by the two-way deterministic register automaton. □

Theorem 1. *A language of planar signed Gauss words can be recognised by two-way deterministic register automata.*

Proof. Compute the Euler characteristics by the two-way deterministic register automaton. If the Euler characteristics χ is equal to 2 then a signed Gauss word is planar [16,8]. □

Unsigned Gauss Words. As for signed words, an unsigned Gauss word w is called *planar* if there is a knot diagram such that w is a Gauss word read off this diagram. Before discussing the recognisability of this property by automata we briefly present an algorithm for recognition of planar unsigned Gauss word proposed by L. Kauffman in [13]. In fact, planarity here does not depend on first components of Gauss data words, i.e. on information whether particular crossing is under- or over-crossing. Because of that the input for the algorithm is

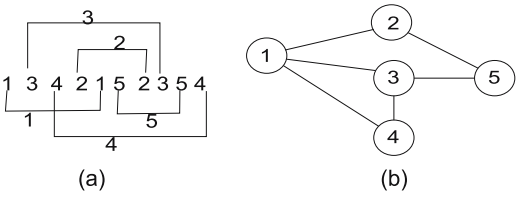


Fig. 4. Not dually paired word and the corresponding graph

a shadow Gauss word, i.e a sequence of labels (=natural numbers), where each label in the sequence occurs twice. We assume the labels in the input word w are $1, 2, \ldots, n$ and they first occur in w in that order. The algorithm proceeds in three stages:

1. The input word is checked on whether there is *even* number of labels in between two appearances of any label. If "yes" the algorithm proceeds to the second stage, if "no" the algorithm stops with the result "no, the input word is non-planar".
2. Starting with $i = 1$, the order of labels occurrence between two occurrences of i is reversed. The process is repeated successively using $i = 1, 2, \ldots, n$. Let w^* is a resulting word.
3. w^* is checked on whether it is a *dually paired* word. If "yes" the algorithm stops with the result "yes, the input word is planar". If "no" the algorithm stops with the result "no, the input word is non-planar".

To complete the algorithm description we explain what is a dually paired word [13].

Definition 5. *For a shadow Gauss word w two labels i and j are called* interlaced *in w iff $w = w_1\ i\ w_2\ j\ w_3\ i\ w_3\ j\ w_4$ or $w = w_1\ j\ w_2\ i\ w_3\ j\ w_3\ i\ w_4$.*

Definition 6. *A shadow Gauss word w is called dually paired iff the set of all labels of w can be partitioned into two subsets such that no two labels in the same subset are interlaced.*

An example of the shadow word which is *not* dually paired is shown in Figure 4(a).

Returning to the question on lower and upper bounds for the planarity of unsigned Gauss words problem one may notice that the first stage of the Kauffman algorithm is obviously implementable on the deterministic register automata. The second stage looks problematic, for to implement reversing the finite memory appears to be insufficient. We do not have a proof though, neither we know a better method to check planarity, so we propose the following

Conjecture 1. Planarity of unsigned Gauss words is not recognisable by (non)-deterministic register automata.

The next proposition will be used later to show upper bound for planarity of unsigned Gauss words problem.

Proposition 3. *The shadow Gauss language of not dually paired words is recognisable by a non-deterministic two-ways register automaton.*

Proof. Given a shadow Gauss word w define an undirected graph g_w as follows. The vertices of q_w are labels in w and there is an edge (i, j) in g_w iff i and j are interlaced. Figure 4(b) demonstrates the graph g_w for the word w shown in Figure 4(a). Now it is straightforward to verify that w is dually paired iff g_w is a bipartite graph. The graph is bipartite iff it does not contain the cycle of the *odd* size. Required register automaton given a word w simulates non-deterministic traversal of the graph g_w. At the beginning it picks up non-deterministically a vertex i of g_w by moving its head to the first occurrence of the label i in w, stores the label i in the register and starts the traversal of the graph moving along the edges of g_w. The *parity* of the length of the path is stored in the finite state control of the automaton. If during traversal the automaton arrives at the vertex stored in the register it checks the parity of the path covered so far and if it is odd the word is accepted. □

We conjectured earlier that planarity of unsigned Gauss words is not recognisable by register automata. In order to get an upper bound one may try to extend the register automata with pebbles. However as the following proposition shows over Gauss words register automata are capable to model effects of adding any finite number of pebbles.

Proposition 4. *If an n-bounded language over infinite alphabet A can be recognised by a k-pebble automaton, then it is also recognised by a k-register automaton.*

Proof. Assume that we have to place a pebble on the symbol $x \in A$ from the word $u \cdot x \cdot v$, $u, v \in A^*$. The position of this pebble can be uniquely represented by a pair (x, l_x), where x is a symbol with a pebble and $l_x = |u|_x$, i.e. the number of occurrences of a symbol x in a word u. Thus, in order to simulate a pebble we need one register (for storing x) and a finite number of states (for keeping l_x in a state space) since l_x is bounded by a constant n. To simulate LIFO stack discipline by the register automaton one may use resister i to store a symbol which has a pebble i placed on it and the finite state structure of the automaton to store a reference to the register which corresponds to the current top of the stack. □

Theorem 2. *Planarity of unsigned Gauss words is recognisable by deterministic linearly bounded memory automata*

Proof. The proof consists in showing that the Kauffman algorithm is implementable on LBMA.

Indeed, the first stage of the algorithm is implementable with the finite memory. Also, it is straightforward to implement the second stage (reversing) on the linear memory. Proposition 3 states that the third stage of the algorithm, that is the search for the cycles of odd size on the graph associated with the Gauss word, can be done non-deterministically using only the finite memory. Notice,

that if the cycle of odd size exists in the graph, then necessarily the odd cycle of the size no more than n also exists, where n is the length of the input word. Deterministic automaton may iterate then over all paths of the length up to n and check the odd cycle condition. The linear order on the input Gauss word induces the linear order on the vertices of the associated graph. It is clear that this order as well as the relation "next" with respect to the order are computable using only finitely many registers. The linear order on vertices is extended lexicoraphically on paths in the graph. Deterministic automaton iterates over paths along this order. No more than $O(n)$ memory is needed. □

4 Conclusion

We have applied automata over infinite alphabets for studying complexity of problems related to knots. We have shown that the language of the signed Gauss words can be recognized by deterministic two-way register automata, while for the same problem for unsigned words we demonstrated an upper bound in terms of automata with linearly bounded memory. The obvious next step is to try to establish a lower and better upper bound for the latter problem. More generally, automata based approach opens new perspectives for studying more complex knot problems, like unknotedness or equivalence. Non-trivial lower bounds for such problems are unknown and weak automata models are plausible candidates here to try. In opposite direction, knot theory provides a reach supply of natural problems formulated in terms of languages over infinite alphabets, and that, one may expect, will influence the development of the theory of such languages and related computational models.

References

1. Kari, J., Niemi, V.: Morphic Images of Gauss Codes. In: Developments in Language Theory, pp. 144–156 (1993)
2. Abramsky, S.: Temperley-Lieb Algebra: From Knot Theory to Logic and Computation via Quantum Mechanics. Mathematics of Quantum Computation and Quantum Technology (2007)
3. 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)
4. Lomonaco Jr., S., Kauffman, L.: Topological Quantum Computing and the Jones Polynomial. Arxiv Preprint Quant-ph/ 0605004 (2006)
5. Hass, J., Lagarias, J., Pippenger, N.: The computational complexity of knot and link problems. Journal of the ACM (JACM) 46(2), 185–211 (1999)
6. Kaminski, M., Francez, N.: Finite-memory automata. In: Proceedings of 31st Annual Symposium on Foundations of Computer Science, pp. 683–688 (1990)
7. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Transactions on Computational Logic (TOCL) 5(3), 403–435 (2004)
8. Kurlin, V.: Gauss phrases realizable by classical links. Arxiv Preprint Math. GT/ 0610929 (2006)

9. David, C.: Mots et données infinies. Master's thesis, LIAFA (2004)
10. Demri, S., Lazic, R.: LTL with the freeze quantifier and register automata. In: LICS, vol. 6, pp. 17–26 (2006)
11. Milo, T., Suciu, D., Vianu, V.: Typechecking for XML transformers. Journal of Computer and System Sciences 66(1), 66–97 (2003)
12. Bjorklund, 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)
13. Kauffman, L.: Virtual knot theory. Arxiv Preprint Math. GT/ 9811028 (1998)
14. Cairns, G., Elton, D.: The planarity problem for signed Gauss words. Journal of Knot Theory and its Ramifications 2(4), 359–367 (1993)
15. Cairns, G., Elton, D.: The Planarity Problem II. Journal of Knot Theory and its Ramifications 5, 137–144 (1996)
16. Carter, J.: Classifying immersed curves. Proc. Amer. Math. Soc. 111, 281–287 (1991)

Analysing Complexity in Classes of Unary Automatic Structures

Jiamou Liu[1] and Mia Minnes[2]

[1] Department of Computer Science, University of Auckland, New Zealand
jliu036@aucklanduni.ac.nz
[2] Department of Mathematics, MIT, USA
minnes@math.mit.edu

Abstract. This paper addresses the time complexity of several queries (including membership and isomorphism) in classes of unary automatic structures and the state complexity of describing these classes. We focus on unary automatic equivalence relations, linear orders, trees, and graphs with finite degree. We prove that in various senses, the complexity of these classes is low: (1) For the isomorphism problem, we either greatly improve known algorithms (reducing highly exponential bounds to small polynomials) or answer open questions about the existence of a decision procedure; (2) for state complexity, we provide polynomial bounds with respect to natural measures of the sizes of the structures.

1 Introduction

A (relational) structure is *automatic* if its elements can be coded in a way such that the domain and all the relations of the structure are recognized by finite automata (precise definitions are in Section 2). Automatic structures form a large class of infinite structures with finite representations and effective semantics. In particular, for any automatic structure $\mathcal{A}$ and first-order query φ, one can effectively construct an automaton that recognizes all elements of $\mathcal{A}$ that satisfy φ. Such useful algorithmic and model-theoretical properties of automatic structures have led to extensive work in the area in recent years.

The field of automatic structures can be viewed as an extension of finite model theory in which one studies the interaction between logical definability and computational complexity. As finite model theory has found applications in databases [1] automatic structures have been used in relational databases and computer-aided verifications [2,3]. However, this approach has limitations. In particular, since the configuration space of a Turing machine can be coded by a finite automaton [4], reachability is undecidable for automatic structures in general. On the other hand, *unary automatic structures*, those recognized by automata over a unary alphabet, have decidable monadic second-order theories. Any automatic structure has an isomorphic copy over the binary alphabet [5]; the intermediate class of structures whose domain elements are encoded as finite strings over 1^*2^* reachability is no longer decidable because the grid can be coded.

Much is known about the complexity of automatic structures. *Algebraic characterizations* have been given for automatic Boolean algebras [4] and finitely

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 518–529, 2009.

generated automatic groups [6]. Some of these results give information about computational content: the isomorphism problem of automatic Boolean algebras is decidable. However, the full class of automatic structures has significant underlying complexity. From a *computability theoretic* point of view, the isomorphism problem and the embedding problem for automatic structures are both Σ^1_1-complete [4,7]. *Model theoretically*, we also find richness: [8] shows that the Scott ranks of automatic structures can be as high as possible, up to the successor of the first non-computable ordinal. Finally, the *resource-bounded complexity* of automatic structures has been studied. For a fixed automaton, the complexity of model-checking for quantifier-free formulas is LOGSPACE-complete and for existential formulas it is NPTIME-complete [9]. On the other hand, there are automatic structures whose first-order theories are nonelementary [10].

In this paper we consider classes of unary automatic structures and show they are simple with respect to two notions of complexity. First, we study the **time complexity** of natural decision problems in a fixed class $\mathcal{K}$ of structures. The *membership problem* asks, given a unary automatic presentation of $\mathcal{A}$, decide if $\mathcal{A} \in \mathcal{K}$. The *isomorphism problem* asks, given unary automatic presentations of $\mathcal{A}$ and $\mathcal{B}$ from $\mathcal{K}$, decide if $\mathcal{A} \cong \mathcal{B}$. For each class of unary automatic structures we consider, we give low polynomial time solutions to its membership problem. For unary automatic equivalence relations, linear orders, and trees we solve the isomorphism problem in low polynomial time; the isomorphism problem for graphs with finite degree is decidable with elementary time bound.

State complexity measures the descriptive complexity of regular languages, context-free grammars, and other classes of languages with finite representations. The state complexity of a regular language L is defined as the size of the smallest automaton recognizing L. It has been studied since the 1950s [11,12,13] and motivated, in part, by the relationship between the runtime of realtime computations running on automata and the size of their state space. We generalize state complexity to structures (rather than sets): the *state complexity* of an automatic structure $\mathcal{A}$ is defined to be the number of states in an optimal automaton presentation of a structure isomorphic to $\mathcal{A}$. For unary automatic structures, we require that the presentation witnessing the state complexity also be a unary automatic structure. We prove that the state complexity of unary automatic equivalence relations, linear orders, and trees are each polynomial with respect to a natural representation of the structures. The study of state complexity of automatic structures which we hope will continue to be fruitful.

Paper organization. Section 2 introduces the terminology and notation used throughout the paper. In particular, it recalls the definitions of automatic structures and unary automatic structures and makes precise the definition of state complexity for unary automatic structures. Sections 3, 4, 5 and 6 discuss linear orders, equivalence relations, trees and graphs of finite degree (respectively). Due to space considerations, this paper is an extended abstract and so we omit many proofs.

2 Preliminaries

We assume the basic terminology and notation from automata theory (see [14] for example). For a fixed alphabet Σ, a *finite automaton* is a tuple $\mathcal{A} = (S, \Delta, \iota, F)$ where S, Δ, ι, F are (respectively) the state space, transition function, initial state, and accepting states. In particular, if $\mathcal{A}$ is a finite automaton over the unary alphabet $\{1\}$ it is called a *unary automaton*. We use *synchronous n-tape automata* to recognize n-ary relations. Such automata have n input tapes, each of which contains one of the input words. Bits of the n input words are read in parallel until all input strings have been completely processed. Formally, let $\Sigma_\diamond = \Sigma \cup \{\diamond\}$ where $\diamond$ is a symbol not in Σ. Given an n-tuple of words $w_1, w_2, \ldots, w_n \in \Sigma^\star$, we convert them to a word over the alphabet $(\Sigma_\diamond)^n$ with length $\max\{|w_1|, \ldots, |w_n|\}$ whose k^{th} symbol $(\sigma_1, \ldots, \sigma_n)$ where σ_i is the k^{th} symbol of w_i if $k \leq |w_i|$, and is $\diamond$ otherwise. An n-ary relation R is FA recognizable if the set of words obtained in this way is a regular subset of $(\Sigma_\diamond^n)^\star$.

A relational *structure* $\mathcal{S}$ consists of a countable domain D and atomic relations on D. A structure is called *automatic* over Σ if it is isomorphic to a structure whose domain is a regular subset of $\Sigma^\star$ and each of whose atomic relations is FA recognizable. A structure is called *unary automatic* if it is automatic over the alphabet $\{1\}$. The structures $(\mathbb{N}; s)$ and $(\mathbb{N}; \leq)$ are both unary automatic structures. On the other hand, $(\mathbb{Q}; \leq)$ and $(\mathbb{N}; +)$ are automatic over $\{0, 1\}$ but are not unary automatic. The structure $(\mathbb{N}; \times)$ is not automatic over any finite alphabet. For proofs of these facts, see the survey papers [15,16].

Consider $FO + \exists^\infty + \exists^{k,l}$, the first-order logic extended by quantifiers for there exist infinitely many and there exist k many mod l $(k, l \in \mathbb{N})$. The following theorem from [9,17,18,5] connects this extended logic with automata.

Theorem 1. *For an automatic structure, $\mathcal{A}$, there is an algorithm that, given a formula $\varphi(\bar{x})$ in $FO + \exists^\infty + \exists^{k,l}$, produces an automaton whose language is those tuples $\bar{a}$ from $\mathcal{A}$ that make φ true.*

We study automatic structures $(D; R)$ where R is a binary relation over D. Theorem 1 can be used to give decision procedures for some properties of binary relations. Table 1 lists the complexity of the associated algorithms if $\mathcal{A}_D$ (m states) and $\mathcal{A}_R$ (n states) are deterministic FA recognizing D and R, respectively. Note that if $(D; R)$ is automatic over Σ and $D = \Sigma^\star$, then $m = 1$.

We use x to denote the string 1^x and $\mathbb{N}$ for the set of all such strings $\{1\}^*$. In [19], Blumensath shows that a structure is unary automatic if and only if it is

Table 1. Deciding properties of binary relations in automatic structures

Property	First-order definition	Time complexity
Reflexivity	$\forall x\,(R(x,x))$	$O(mn)$
Symmetry	$\forall x, y\,(R(x,y) \implies R(y,x))$	$O(n^2)$
Antisymmetry	$\forall x, y\,(R(x,y) \wedge R(y,x) \implies x = y)$	$O(n^2)$
Totality	$\forall x, y\,(R(x,y) \vee R(y,x))$	$O(m^2n^2)$
Transitivity	$\forall x, y, z\,(R(x,y) \wedge R(y,z) \implies R(x,z))$	$O(n^3)$

first-order interpretable in $\mathcal{U} = (\mathbb{N}; 0, <, s, \{\mathrm{mod}_j\}_{j>1})$, where s is the successor relation and $\mathrm{mod}_j(x, y)$ holds if and only if $x \equiv y \mod j$, and therefore proves the following.

Theorem 2. *The monadic second order (MSO) theory of a unary automatic structure is decidable.*

We need to understand the structure of regular subsets and relations of $\{1\}^\star$.

Lemma 1 ([19]). *A set $L \subseteq \mathbb{N}$ is unary automatic if and only if there are $t_L, \ell_L \in \mathbb{N}$ such that $L = L_1 \cup L_2$ with $L_1 \subseteq \{0, 1, \ldots, t_L - 1\}$ and L_2 is a finite union of sets in the form $\{j + i\ell_L\}_{i \in \mathbb{N}}$ where $t_L \leq j < t_L + \ell_L$.*

A 2-tape unary automaton may be described as follows.States reachable from the initial state by reading inputs of type $(1, 1)$ are called $(1, 1)$-*states* and form a disjoint union of a *tail* and a *loop*. We label the $(1, 1)$-tail states as $q_0, \ldots, q_{t-1}$; the $(1, 1)$-loop states as $q_t, \ldots, q_{t+\ell-1}$. States reachable from a $(1, 1)$-state by reading inputs of type $(1, \diamond)$ are called $(1, \diamond)$-*states*. The set of $(1, \diamond)$-states reachable from any given q_i consist of a tail and a loop, called the $(1, \diamond)$-*tail* and *loop* from q_i; the $(\diamond, 1)$-*tails* and *loops* are defined similarly.

Khoussainov and Rubin [20] and Blumensath [19] gave a characterization of all unary automatic binary relations on $\mathbb{N}$. Let $\mathcal{B} = (B, E_B)$ and $\mathcal{D} = (D, E_D)$ be finite graphs. Let R_1, R_2 be subsets of $D \times B$, and R_3, R_4 be subsets of $B \times B$. Consider the graph $\mathcal{D}$ followed by countably infinitely many copies of $\mathcal{B}$, ordered as $\mathcal{B}^0, \mathcal{B}^1, \mathcal{B}^2, \ldots$. We define the infinite graph $\mathit{unwind}(\mathcal{B}, \mathcal{D}, \bar{R})$ as follows. Its vertex set is $D \cup B^0 \cup B^1 \cup B^2 \cup \ldots$ and its edge set contains $E_D \cup E^0 \cup E^1 \cup \ldots$ as well as the following edges, for all $a, b \in B$, $d \in D$, and $i, j \in \omega$:

- (d, b^0) when $(d, b) \in R_1$, and (d, b^{i+1}) when $(d, b) \in R_2$,
- (a^i, b^{i+1}) when $(a, b) \in R_3$, and (a^i, b^{i+2+j}) when $(a, b) \in R_4$.

Theorem 3 ([19,20]). *A graph is unary automatic if and only if it is isomorphic to $\mathit{unwind}(\mathcal{B}, \mathcal{D}, \bar{R})$ for some parameters $\mathcal{B}, \mathcal{D}, \bar{R}$.*

The *state complexity* of a regular language L is the number of states of the minimal deterministic finite automaton that recognizes L [12].

Definition 1. *The* state complexity *of an (unary) automatic structure $\mathcal{A}$ is the size of the smallest (unary) automaton $\mathcal{M}$ (*optimal automaton*) such that $\mathcal{M}$ recognizes a structure $\mathcal{B} \cong \mathcal{A}$. If $\mathcal{K}$ is a class of automatic structures with associated finite isomorphism invariant $R_{\mathcal{A}}$, the* state complexity *of $\mathcal{K}$ is f such that for $A \in \mathcal{K}$, if $|R_{\mathcal{A}}| \leq n$, the state complexity of $\mathcal{A}$ is less than $f(n)$.*

Note that this is a worst-case complexity measure: if a class of structures has state complexity f, there may still be a particular member of the class whose state complexity is much smaller. We look at the state complexity of three classes of unary automatic structures: equivalence relations, linear orders, and trees. We define natural isomorphism invariants for each class and show that the state complexity for each class is polynomial with respect to these invariants.

In the sequel, we make the following assumptions without loss of generality. All structures are infinite with domain $\mathbb{N}$. All automata are deterministic. Algorithms on unary automatic structures $(\mathbb{N}; R)$ have as input a synchronous 2-tape automaton recognizing R. The size of the input is the size of this input automaton. The sets of (1,1)-, $(\diamond, 1)$-, and $(1, \diamond)$- states are pairwise disjoint.

3 Linear Orders

A *linear order* is $\mathcal{L} = (\mathbb{N}; \leq_L)$ where $\leq_L$ is total, reflexive, antisymmetric, and transitive. Table 1 immediately gives that the membership problem for automatic linear orders is decidable in time $O(n^3)$. Blumensath [19] and Khoussainov and Rubin [20] characterized unary automatic linear orders. We use ω, ω^*, ζ, and $\mathbf{n}$ to denote the order types of the positive integers, the negative integers, the integers, and the finite linear order of length n (respectively).

Theorem 4 ([19,20]). *A linear order is unary automatic if and only if it is isomorphic to a finite sum of linear orders of type ω, ω^* or $\mathbf{n}$.*

Corollary 1. *The isomorphism problem for unary automatic linear orders is decidable.*

This corollary is proved by defining, for each unary automatic linear order $\mathcal{L}$, a sentence $\varphi_{\mathcal{L}}$ such that for any $\mathcal{L}'$, $\mathcal{L} \cong \mathcal{L}'$ if and only if $\mathcal{L}' \vDash \varphi_{\mathcal{L}}$; this yields a triply exponential decision procedure. We improve this bound, giving a quadratic time algorithm for the isomorphism problem for unary automatic linear orders.

3.1 Efficient Solution to the Isomorphism Problem

Theorem 5. *The isomorphism problem for unary automatic linear orders is decidable in quadratic time in the sizes of the input automata.*

We use the notation from Section 2: given a unary automaton $\mathcal{A}$ it has parameters t, ℓ which are the lengths of its $(1,1)$-tail and -loop.

Lemma 2. *Suppose $\mathcal{A}$ is a unary automaton that represents a linear order $\mathcal{L} = (\mathbb{N}; \leq_L)$. For any $t \leq j < \ell$, the sequence $(j + i\ell)_{i \in \mathbb{N}}$ is either an increasing chain in a copy of ω in $\mathcal{L}$ or a decreasing chain in a copy of ω^* in $\mathcal{L}$.*

Proof (of Theorem 5). Suppose $\mathcal{L} = (\mathbb{N}; <_L)$ is a unary automatic linear order represented by a unary automaton $\mathcal{A}$ with parameters t, ℓ. We will extract its canonical word $\alpha_{\mathcal{L}} \in \{\omega, \omega^*, \mathbf{n}\}^\star$ by determining the relative ordering of the at most ℓ many copies of ω and ω^* given by Lemma 2 and the elements $0, \ldots, t-1$.

For $t \leq j < t + \ell$, we decide in linear time whether the sequence $(j + i\ell)_{i \in \mathbb{N}}$ is in a copy of ω or ω^*. Two sequences $(j + i\ell)_{i \in \mathbb{N}}$ and $(k + i\ell)_{i \in \mathbb{N}}$ $(j < k)$ *interleave* if they belong to the same copy of ω or ω^* in $\mathcal{L}$. We check this by reading the $(\diamond, 1)$-loops at most ℓ many times. These loops also specify the relative order of $(j + i\ell)_{i \in \mathbb{N}}$ and $(k + i\ell)_{i \in \mathbb{N}}$ if they do not interleave. Thus, in $O(n^2)$ time,

we compute descriptions of the predecessors of each $(j + i\ell)_{i\in\mathbb{N}}$ in $\mathcal{L}$ and an equivalence relation on $\{t, \ldots, t+\ell-1\}$ based on interleaving. Similarly, we can compute the predecessors of each j, $0 \leq j < t$ in $O(n)$ time. To extract $\alpha_\mathcal{L}$ from $\mathcal{A}$, it remains to iterate through states according to their $\mathcal{L}$-predecessors, and decided if they belong to a finite block, a copy of ω, or a copy of ω^2. This can be done in $O(n^2)$. □

3.2 State Complexity

The order type of a unary automatic linear order $\mathcal{L} = (\mathbb{N}; \leq_L)$ is specified by $\alpha_\mathcal{L} \in \{\omega, \omega^*, \{\mathbf{n}\}_{n\in\mathbb{N}}\}^\star$. Let $m_\mathcal{L}$ be the number of copies of ω or ω^* in $\alpha_\mathcal{L}$ and let $k_\mathcal{L}$ be the sum of all n such that $\mathbf{n}$ appears in $w_\mathcal{L}$. Define $|\alpha_\mathcal{L}| = \max\{m_\mathcal{L}, k_\mathcal{L}\}$.

Theorem 6. *The state complexity of the class of unary automatic linear orders*

- *with* $m_\mathcal{L} = 0, k_\mathcal{L} > 0$ *is* $\frac{1}{2}(k_\mathcal{L}^2 + k_\mathcal{L})$*; with* $m_\mathcal{L} > 0, k_\mathcal{L} = 0$ *is* $2m_\mathcal{L}^2 + m_L$*; and*
- *with* $m_\mathcal{L} > 0, k_\mathcal{L} > 0$ *is* $k_\mathcal{L}^2 + 2m_\mathcal{L}^2 + 2m_\mathcal{L}k_\mathcal{L} + m_\mathcal{L}$*.*

Proof. Lemma 2 implies the optimal automaton has $m_\mathcal{L} + k_\mathcal{L}$ $(1,1)$-states. □

Corollary 2. *The (unary) state complexity for the class of unary automatic linear orders is quadratic in the size of the associated parameter,* $|\alpha_\mathcal{L}|$*.*

4 Equivalence Relations

A structure $\mathcal{E} = (\mathbb{N}; E)$ is an *equivalence structure* if E is an equivalence relation (reflexive, symmetric, and transitive). Table 1 immediately gives that the membership problem for automatic equivalence structures is decidable in time $O(n^3)$. Blumensath [19] and Khoussainov and Rubin [20] classified unary automatic equivalence structures.

Theorem 7 ([19,20]). *An equivalence structure has a unary automatic presentation if and only if it has finitely many infinite equivalence classes and there is a finite bound on the sizes of the finite equivalence classes.*

The *height* of an equivalence structure $\mathcal{E}$ is a function $h_\mathcal{E} : \mathbb{N} \cup \{\infty\} \to \mathbb{N} \cup \{\infty\}$ where $h_\mathcal{E}(x)$ is the number of E-equivalence classes of size x. Two equivalence structures $\mathcal{E}_1$ and $\mathcal{E}_2$ are isomorphic if and only if $h_{\mathcal{E}_1} = h_{\mathcal{E}_2}$. By Theorem 7, if $\mathcal{E}$ is unary automatic then $h_\mathcal{E}$ is finitely nonzero.

Corollary 3. *The isomorphism problem for unary automatic equivalence structures is decidable.*

We may define an extended first order sentence which describes the height function $h_\mathcal{E}$ and leads to doubly exponential runtime for deciding whether two

equivalence structures are isomorphic. We significantly improve this bound by giving a quadratic time algorithm for the isomorphism problem.

4.1 Efficient Solution to the Isomorphism Problem

Theorem 8. *The isomorphism problem for unary automatic equivalence structures is decidable in quadratic time in the sizes of the input automata.*

Proof. Suppose $\mathcal{E}$ is recognized by a unary automaton $\mathcal{A}$ with n states. Recall the definitions of t, ℓ, and q_j from Section 2. Observe that each $j < t+\ell$ belongs to an infinite equivalence class if and only if there is an accepting state on the $(\diamond, 1)$ loop from q_j. Let $t \leq j < t + \ell$. If j belongs to an infinite equivalence class then the set $\{j + i\ell\}_{i\in\mathbb{N}}$ is partitioned into c infinite equivalence classes for some $c > 0$. It takes $O(n^2)$ time to compute the total number of infinite equivalence classes. Each $0 \leq j < t + \ell$ such that q_j has no accepting state on its $(\diamond, 1)$-loop may be responsible for infinitely many finite equivalence classes of the same size and finitely many other equivalence classes. We can find all k such that $h_{\mathcal{E}}(k) = \infty$ in $O(n)$ time. It remains to compute the sizes of equivalence classes for elements represented on the $(1, 1)$-loop but such that $h_{\mathcal{E}}(k) < \infty$. This can be done by reading through the $(\diamond, 1)$-tails off the $(1, 1)$-tail and has runtime $O(n)$. This algorithm takes $O(n^2)$. □

4.2 State Complexity

The height function $h_{\mathcal{E}}$ is a finite isomorphism invariant for unary automatic equivalence structures. We will express the state complexity in terms of the height function $h_{\mathcal{E}}$; let $h_{\inf} = h_{\mathcal{E}}(\infty)$ and $n_{\inf} = \sum_{n:h_{\mathcal{E}}(n)=\infty} n$. Define

$$|h_{\mathcal{E}}| = \left(\sum_{n<\infty : h_{\mathcal{E}}(n)<\infty} n h_{\mathcal{E}} \right) + n_{inf} + h_{inf}.$$

Theorem 9. *The state complexity of unary automatic equivalence structure* $\mathcal{E} = (\mathbb{N}; E)$ *is at least* $\left(\sum_{n<\infty : h_{\mathcal{E}}(n)<\infty} n^2 h_{\mathcal{E}}(n) \right) + 2h_{\inf}(n_{\inf} + 1) + n_{\inf} + 1$ *and at most* $\left(\sum_{n<\infty : h_{\mathcal{E}}(n)<\infty} n^2 h_{\mathcal{E}}(n) \right) + \left(\sum_{n<\infty : h_{\mathcal{E}}(n)=\infty} n^2 \right) + 2h_{inf}(n_{inf} + 1) + 1.$

Proof. The upper bound is obtained by analysing the relationship between infinite equivalence classes, finite equivalence classes that occur infinitely often, and the $(1, 1)$-loop of the optimal automaton. The lower bound is computed by overlapping $(1, 1)$-states so they correspond to multiple equivalence classes. □

Corollary 4. *The (unary) state complexity for the class of unary automatic equivalence structure is quadratic in the height function.*

5 Trees

5.1 Characterizing Unary Automatic Trees

A *tree* is $\mathcal{T} = (\mathbb{N}; \leq_T)$ where $\leq_T$ is a partial order (reflexive, antisymmetric, and transitive) with a root (the least element) and the set of *ancestors* of any node x, $\{y : y \leq_T x\}$, is a finite linear order. Two nodes x, y are *incomparable*, $x|_T y$, if $x \not\leq_T y$ and $y \not\leq_T x$; an *anti-chain* of $\mathcal{T}$ is a set of nodes which are pairwise incomparable. Table 1 gives an $O(n^3)$ algorithm to check if a given automatic relation is a partial order. Checking if $\leq_T$ is total on every set of predecessors takes time $O(n^4)$. There is an $O(n)$ time algorithm checking if a unary automatic partial order $(\mathbb{N}; \leq_T)$ has a least element. Thus, the membership problem for unary automatic trees is decidable in time $O(n^4)$.

As we saw in previous sections, a good characterization of a class of unary automatic structures may lead to a better understanding of its complexity bounds. We present such a characterization of unary automatic trees. A *parameter set* Γ is a tuple $(\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma, X)$ where $\mathcal{T}_0, \ldots, \mathcal{T}_m$ are finite trees (with disjoint domains T_i), $\sigma : \{1, \ldots, m\} \to T_0$ and $X : \{1, \ldots, m\} \to \{\emptyset\} \cup \bigcup_i T_i$ such that $X(i) \in T_i \cup \{\emptyset\}$.

Definition 2. *A* tree-unfolding *of a parameter set Γ is the tree $UF(\Gamma)$ defined as follows:*

- *$UF(\Gamma)$ contains one copy of $\mathcal{T}_0$ and infinitely many copies of each $\mathcal{T}_i$ ($1 \leq i \leq m$), $(\mathcal{T}_i^j)_{j \in \omega}$. If $x \in T_i$, its copy in $\mathcal{T}_i^j$ is denoted by (x, j)*
- *For $1 \leq i \leq m$, if $X(i) \neq \emptyset$, the root of $\mathcal{T}_i^0$ is a child (immediate descendent) of $\sigma(i)$, and the root of $\mathcal{T}_i^{j+1}$ is a child of $(X(i), j)$ for all j.*
- *For $1 \leq i \leq m$, if $X(i) = \emptyset$, the root of $\mathcal{T}_i^j$ is a child of $\sigma(i)$ for all j.*

Theorem 10. *A tree $\mathcal{T}$ is unary automatic if and only if there is a parameter set $\Gamma = (\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma, X)$ such that $\mathcal{T} \cong UF(\Gamma)$.*

Suppose $\mathcal{T} = (\mathbb{N}; \leq_T)$ is recognized by a unary automaton $\mathcal{A}$ with n states and parameters t, ℓ. We say that two disjoint sets X and Y of nodes in $\mathcal{T}$ are *incomparable* if $(\forall x \in X)(\forall y \in Y)(x|_T y)$. We sketch the proof of Theorem 10.

Lemma 3. *For $t \leq j < t + l$, the set $(j + i\ell)_{i \in \mathbb{N}}$ forms either an anti-chain or finitely many pairwise incomparable infinite chains in T.*

Let $A = \{j : (j + i\ell)_{i \in \mathbb{N}}$ is an anti-chain, $t \leq j < t + \ell\}$ and $C = \{t, \ldots, t + \ell - 1\} - A$. For each $j \in C$, let n_j be the number of infinite chains in $(j + i\ell)_{i \in \mathbb{N}}$. For $0 \leq m < n_j$, we denote the infinite chain formed by $(j + (m + in_j)\ell)_{i \in \mathbb{N}}$ by $W_{j,m}$. $W_{j,m}$ and $W_{k,m'}$ may belong to the same infinite path in $\mathcal{T}$ (*interleave* in the sense of Section 3); if they do, then $n_j = n_k$. Any infinite path through $\mathcal{T}$ must be given by element(s) in C. Hence, $\mathcal{T}$ contains only finitely many infinite paths. We define a *component* of $\mathcal{T}$ to be a connected subgraph of $\mathcal{T}$ which contains exactly one infinite path and such that all the elements in the subgraph are greater than or equal to t. For $j \in C$ and $k \in A$, we can decide if any

element in $(k+i\ell)_{i\in\mathbb{N}}$ belongs to a component of $\mathcal{T}$ intersecting with $(j+i\ell)_{i\in\mathbb{N}}$; if some element does, then $(k+i\ell)_{i\in\mathbb{N}}$ belongs to the same components as the class $(j+i\ell)_{i\in\mathbb{N}}$. If $j,k\in A$ and neither $(j+i\ell)_{i\in\mathbb{N}}$ nor $(k+i\ell)_{i\in\mathbb{N}}$ intersects with any component of $\mathcal{T}$, we check if the union $(j+i\ell)_{i\in\mathbb{N}}\cup(k+i\ell)_{i\in\mathbb{N}}$ is a subset of infinitely many disjoint finite subtrees in $\mathcal{T}$, each of which contains the nodes $j+i\ell$ and $k+(i+m)\ell$ for some i. We call these disjoint finite trees *independent*.

The above argument facilitates the definition of an equivalence relation $\sim$ on $\{t,\ldots,t+\ell-1\}$ as $j\sim k$ if and only if $j\in C$ (or $k\in C$) and $\{j+i\ell\}_{i\in\mathbb{N}}$ and $\{k+i\ell\}_{i\in\mathbb{N}}$ belong to the same n_j (or n_k) components in $\mathcal{T}$; or, $j,k\in A$ and there is $h\in C$ such that $j\sim h$ and $k\sim h$; or $j,k\in A$ and $\{j+i\ell\}_{i\in\mathbb{N}}$ and $\{k+i\ell\}_{i\in\mathbb{N}}$ belong to the same collection of independent trees in $\mathcal{T}$. We use $[j]$ to denote the $\sim$-equivalence class of j.

Proof (of Theorem 10). We show that any unary automatic tree is isomorphic to the tree-unfolding $\mathrm{UF}(\Gamma)$ of some parameter set $\Gamma=(\mathcal{T}_0,\mathcal{T}_1,\ldots,\mathcal{T}_m,\sigma,X)$. Each $\sim$-equivalence class $[j]$ either represents infinitely many independent trees or finitely many components of $\mathcal{T}$. Components of $\mathcal{T}$ represented by $[j]$ are pairwise isomorphic and can be described by "unfolding" a finite graph of size $|[j]|$. In either case, the set of ancestors of $[j]$ in $\mathcal{T}$ is finite. This description may be translated to a parameter set for Γ. Conversely, suppose that $\Gamma=(\mathcal{T}_0,\mathcal{T}_1,\ldots,\mathcal{T}_m,\sigma,X)$ is a parameter set. Let $t=|T_0|$, $\ell=\Sigma_{r=1}^{m}|T_r|$ and $\alpha_r=\Sigma_{i=1}^{r-1}|T_i|$ for $r=1,\ldots,m$. We consider the isomorphic copy $(\mathbb{N};\leq_T)\cong\mathrm{UF}(\Gamma)$ where $T_0\mapsto\{0,\ldots,|T_0|\}$ and the j^{th} copy of $\mathcal{T}_r$ maps to $\{t+(j-1)\ell+\alpha_r,\ldots,t+(j-1)\ell+\alpha_{r+1}-1\}$. The unary automaton recognizing $\mathrm{UF}(\Gamma)$ has parameters t,ℓ; the other states and F can be deduced from σ and X. □

5.2 Efficient Solution to the Isomorphism Problem

Two tree-unfoldings may be isomorphic even if the associated parameter sets are not isomorphic term-by-term. We define an isomorphic invariant: a restricted parameter set. Fix a computable linear order $\preceq$ on the set of finite trees.

Definition 3. *The* canonical parameter set *of a unary automatic tree $\mathcal{T}=(\mathbb{N};\leq_T)$ is the parameter set $\Gamma=(\mathcal{T}_0,\mathcal{T}_1,\ldots,\mathcal{T}_m,\sigma,X)$ such that $UF(\Gamma)\cong\mathcal{T}$ and which is minimal in the following sense:*

1. *As finite trees, $T_1\preceq\ldots\preceq T_m$.*
2. *If $T_i\cong T_j$, $\sigma(i)=\sigma(j)$, and $X(i)=X(j)=\emptyset$ then $i=j$.*
3. *Each T_i $(1\leq i\leq m)$ is minimal: If $X(i)\neq\emptyset$ then if $y_1\leq_T y_2\leq_T X(i)$ the subtree with domain $\{z:y_1\leq_T z\wedge y_2\not\leq_T z\}$ is not isomorphic to the subtree with domain $\{z:y_2\leq_T z\wedge X(i)\not\leq_T z\}$.*
4. *T_0 is minimal: T_0 has the fewest possible nodes and for all $1\leq i\leq m$ where $X(i)\neq\emptyset$, there is no $y\in T_0$ such that $y\leq_T\sigma(i)$ and the subtree with domain $\{z:y\leq_T z\wedge\sigma(i)\not\leq_T z\}$ is isomorphic to T_i.*

Lemma 4. *Suppose $\mathcal{T},\mathcal{T}'$ are unary automatic trees with canonical parameter sets Γ,Γ'. Then, $\mathcal{T}\cong\mathcal{T}'$ if and only if Γ,Γ' have the same number (m) of finite trees, $(\mathcal{T}_0,\sigma)\cong(\mathcal{T}_0',\sigma')$, and for $1\leq i\leq m$, $(\mathcal{T}_i,X(i))\cong(\mathcal{T}_i',X'(i))$.* □

The canonical parameter set can be used to define an extended first-order formula $\varphi_{\mathcal{T}}$ which specifies the isomorphism type of $\mathcal{T}$; hence, the isomorphism problem for unary automatic trees is decidable. We now improve this result by reducing the time complexity of the associated decision procedure.

Theorem 11. *The isomorphism problem for unary automatic trees is decidable in time $O(n^4)$ in the sizes of the input automata.*

It is sufficient to compute the canonical parameter set; then, Lemma 4 and a decision procedure for isomorphism on finite trees solves the isomorphism problem on unary automatic trees. We omit the (long) proof of Lemma 5.

Lemma 5. *If $\leq_T$ is recognized by unary automaton with n states, there is an $O(n^4)$ time algorithm that computes the canonical parameter set of $\mathcal{T}$*

Proof (Theorem 11). Suppose $\mathcal{T}_1, \mathcal{T}_2$ are presented by unary automata $\mathcal{A}_1, \mathcal{A}_2$ with n_1, n_2 states (respectively). Let $n = \max\{n_1, n_2\}$ By Theorem 10 and Lemma 5, deciding if $\mathcal{T}_1 \cong \mathcal{T}_2$ reduces to checking finitely many isomorphisms of finite trees. The appropriate canonical parameter sets are built in $O(n^4)$ time and each have $O(n^2)$ finite trees, each of size $O(n)$. □

5.3 State Complexity

Suppose $\mathcal{T} = \mathrm{UF}(\Gamma)$ and $\Gamma = (\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma, X)$ is the canonical parameter set of T. Let $t = |T_0|$ and $\ell = \sum_{i=1}^{m} |T_i|$.

Theorem 12. *The state complexity of unary automatic tree T is less than $(t + \ell)^2 - t\ell + t + \ell$ and greater than ℓ^2.*

Proof. Theorem 10 gives the upper bound. The lower bound is obtained when $T_1, \ldots, T_\ell$ are pairwise nonisomorphic. □

Corollary 5. *The (unary) state complexity of a unary automatic tree $\mathcal{T}$ is quadratic in the parameters t, ℓ of its canonical parameter set.*

6 Graphs of Finite Degree

A graph $\mathcal{G} = (\mathbb{N}; R)$ is of *finite degree* if $(\forall x)(\neg\exists^{\infty} y)(R(x, y) \vee R(y, x))$. If R is recognized by a unary automaton, $\mathcal{G}$ is of finite degree if and only if there is no accepting state on any $(\diamond, 1)$- or $(1, \diamond)$-loops. Therefore, the membership problem is decidable in linear time. In [21], Khoussainov, Liu, and Minnes investigated a range of algorithmic properties of unary automatic graphs of finite degree. For example, they showed that the reachability problem for unary automatic graphs of finite degree can be decided in polynomial time in the sizes of the input vertices and the automaton. We will make use of the following.

Theorem 13 ([21]). *For a unary automatic graph $\mathcal{G}$ of finite degree, we can construct a unary automaton that recognizes the reachability relation on $\mathcal{G}$ in polynomial time; connectedness of $\mathcal{G}$ is decidable in $O(n^3)$ time.*

However, [21] left open the decidability of the isomorphism problem. We now settle this question and provide an algorithm deciding the isomorphism problem for unary automatic graphs of finite degree.

Theorem 14. *The isomorphism problem for unary automatic graphs of finite degree is decidable in elementary time.*

For brevity, we assume $\mathcal{G}$ is undirected and sketch the proof We use the following characterization from [21]. Given a finite graph $\mathcal{F} = (V_\mathcal{F}; E_F)$ and a map $\sigma : V_\mathcal{F} \to \mathcal{P}(V_\mathcal{F})$, $\mathcal{F}_{\sigma^\omega}$ is the disjoint union of infinitely many copies of $\mathcal{F}$ with an edge between $x \in \mathcal{F}_i$ and $y \in \mathcal{F}_{i+1}$ if and only if $y \in \sigma(x)$.

Theorem 15 ([21]). *A graph of finite degree $\mathcal{G} = (\mathbb{N}; R)$ is unary automatic if and only if there are finite graphs $\mathcal{D}, \mathcal{F}$ and a map $\sigma : V_\mathcal{F} \to P(V_\mathcal{F})$ such that $\mathcal{G} \cong \mathcal{G}'$ and $\mathcal{G}'$ is a disjoint union of $\mathcal{D}$ and $\mathcal{F}_{\sigma^\omega}$ with possible additional edges between $\mathcal{D}$ and $\mathcal{F}_0$. Furthermore, the parameters $\mathcal{D}, \mathcal{F}, \sigma$ can be extracted in time $O(n^2)$ from a unary automaton recognizing R.*

A *component of* $\mathcal{G}$ is the transitive closure of a vertex under the edge relation. By Theorem 15, any infinite component in $\mathcal{G}$ has nonempty intersection with almost all $\mathcal{F}_i$. Therefore, $\mathcal{G}$ has at most $|V_\mathcal{F}|$ many infinite components. Also, any finite component of $\mathcal{G}$ has size at most $|V_\mathcal{D} + V_\mathcal{F}|$. Let $\mathcal{G}_{\text{Fin}}$ be the subset of $\mathcal{G}$ containing only its finite components. By Theorem 15, if C is any finite component of $\mathcal{G}$ then either $C \cap \mathcal{F}_j \neq \emptyset$ for some $j < \ell$ or C has infinitely many isomorphic copies in $\mathcal{G}$. Moreover, there are finitely many isomorphism classes of finite components of $\mathcal{G}$, and we can tell which of these correspond to infinitely many components in $\mathcal{G}$. Hence, given two graphs $\mathcal{G}, \mathcal{G}'$ we can decide if $\mathcal{G}_{\text{Fin}} \cong \mathcal{G}'_{\text{Fin}}$.

It remains to prove that it is decidable whether two infinite components of unary automatic graphs are isomorphic. It suffices to prove that we can decide whether two infinite connected unary automatic graphs are isomorphic. We will give sufficient MSO conditions which guarantee that an infinite connected graph $\mathcal{H}$ is isomorphic to $\mathcal{G}$. For a partition of $\mathcal{H}$ into $3k$ sets P_i, a subgraph $\mathcal{M}$ of $3k$ vertices in $\mathcal{H}$ is an $\mathcal{F}_{\times 3}$-*type* if $\mathcal{M}$ intersects with each P_i at exactly one vertex, and if we let $v_i = \mathcal{M} \cap P_i$, then the three sets of vertices $\{v_1, \ldots, v_k\}$, $\{v_{k+1}, \ldots, v_{2k}\}$, $\{v_{2k+1}, \ldots, v_{3k}\}$ respectively form three copies of $\mathcal{F}$, with v_i, v_{k+i}, v_{2k+i} corresponding to the same vertex in $\mathcal{F}$. Also, the edge relation between these three copies of $\mathcal{F}$ must respect the mapping σ. Then an isomorphism between $\mathcal{H}$ and $\mathcal{G}$ is guaranteed by the following (MSO-expressible) conditions. Each vertex v in $\mathcal{H}$ belongs to a unique subgraph that is an $\mathcal{F}_{\times 3}$-type; and, for each $\mathcal{F}_{\times 3}$-type $\mathcal{M}$, there is a unique $\mathcal{F}_{\times 3}$-type $\mathcal{N}$ that is a *successor* of $\mathcal{M}$ (all edges between $\mathcal{M}$ and $\mathcal{N}$ are from the last copy of $\mathcal{F}$ in $\mathcal{M}$ to the first copy of $\mathcal{F}$ in $\mathcal{N}$ and respect the mapping σ). Also, there is a unique $\mathcal{F}_{\times 3}$-type $\mathcal{M}_0$ which is not the successor of any other $\mathcal{F}_{\times 3}$-types and any other $\mathcal{F}_{\times 3}$-type is the successor of a unique $\mathcal{F}_{\times 3}$-type. The isomorphism maps $\mathcal{M}_0$ to the first 3 copies of $\mathcal{F}$ in $\mathcal{G}$, and then map the other vertices according to the successor relation and σ. By Theorem 2, satisfiability of an MSO sentence is decidable for unary automatic graphs. Therefore the isomorphism problem for unary automatic graphs of finite degree is decidable. Note that formalizing the conditions on $\mathcal{H}$ requires only

finitely many alternations of quantifiers (regardless of the size of the automaton presenting it) and so the decision procedure is elementary in terms of the size of the input automaton.

References

1. Libkin, L.: Elements of Finite Model Theory. EATCS. Springer, Heidelberg (2004)
2. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS 1986), pp. 332–344. IEEE Computer Society Press, Los Alamitos (1986)
3. Vardi, M.: Model checking for database theoreticians. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 1–16. Springer, Heidelberg (2004)
4. Khoussainov, B., Nies, A., Rubin, S., Stephan, F.: Automatic structures: Richness and limitations. In: Proc. 19th LICS, pp. 44–53. IEEE Computer Society, Los Alamitos (2004)
5. Rubin, S.: Automatic Structures. PhD thesis, University of Auckland (2004)
6. Olver, G., Thomas, R.: Automatic presentation for finitely generated groups. In: Proc. 5th DLT. LNCS, pp. 130–144. Springer, Heidelberg (2002)
7. Vinokurov, N.: Complexity of some natural problems in automatic structures. Siberian Mathematical Journal 46, 56–61 (2005)
8. Khoussainov, B., Minnes, M.: Model theoretic complexity of automatic structures (Extended abstract). In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 520–531. Springer, Heidelberg (2008)
9. Blumensath, A., Grädel, E.: Automatic structures. In: Proc. 15th LICS, pp. 51–62. IEEE Computer Society, Los Alamitos (2000)
10. Blumensath, A., Grädel, E.: Finite presentations of infinite structures: Automata and interpretations. Theory of Computing Systems 37, 641–674 (2004)
11. Campeanu, C., Culik II, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages, automata implementation. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 60–70. Springer, Heidelberg (2001)
12. Yu, S.: Regular languages. Handbook of Formal Languages, ch. 2 (1997)
13. Yu, S.: State complexity: recent results and open problems. Fundamenta Informaticae 64(1-4), 471–480 (2005)
14. Hopcroft, J.E., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (2001)
15. Khoussainov, B., Minnes, M.: Three lectures on automatic structures. In: Proc. LC 2007. Cambridge University Press, Cambridge (2008)
16. Rubin, S.: Automata presenting structures: A survey of the finite string case. Bulletin of Symbolic Logic 14(2), 169–209 (2008)
17. Hodgson, B.: On direct products of automaton decidable theories. Theoretical Computer Science 19, 331–335 (1982)
18. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995)
19. Blumensath, A.: Automatic Structures. Diploma thesis, RWTH Aachen (1999)
20. Khoussainov, B., Rubin, S.: Graphs with automatic presentations over a unary alphabet. J. of Automata, Languages and Combinatorics 6(4), 467–480 (2001)
21. Khoussainov, B., Liu, J., Minnes, M.: Unary automatic graphs: An algorithmic perspective. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 548–559. Springer, Heidelberg (2008)

An Application of Generalized Complexity Spaces to Denotational Semantics via the Domain of Words

Jordi Llull-Chavarría and Oscar Valero*

Departamento de Ciencias Matemáticas e Informática, Universidad de las Islas Baleares, 07122, Baleares, Spain
o.valero@uib.es

Abstract. In 1995 M. Schellekens introduced the theory of complexity spaces as a part of the development of a mathematical (topological) foundation for the complexity analysis of programs and algorithms [Electronic Notes in Theoret. Comput. Sci. 1 (1995), 211-232]. This theory is based on the structure of quasi-metric spaces which allow to measure relative progress made in lowering the complexity when a program is replaced by another one. In his paper, Schellekens showed the applicability of the theory of complexity spaces to the analysis of Divide & Conquer algorithms. Later on, S. Romaguera and Schellekens introduced the so-called dual (quasi-metric) complexity space in order to obtain a more robust mathematical structure for the complexity analysis of programs and algorithms [Topology Appl. 98 (1999), 311-322]. They studied some properties of the original complexity space, which are interesting from a computational point of view, via the analysis of the dual ones and they also gave an application of the dual approach to the complexity analysis of Divide and Conquer algorithms. Most recently, Romaguera and Schellekens introduced and studied a general complexity framework which unifies the original complexity space and the dual one under the same formalism [Quaestiones Mathematicae 23 (2000), 359-374]. Motivated by the former work we present an extension of the generalized complexity spaces of Romaguera and Schellekens and we show, by means of the so-called domain of words, that the new complexity approach is suitable to provide quantitative computational models in Theoretical Computer Science. In particular our new complexity framework is shown to be an appropriate tool to model the meaning of while-loops in formal analysis of high-level programming languages.

2000 AMS Classification: 54E50, 54F05, 54C35, 68Q55, 68Q45, 68Q25.

* Corresponding author. The first author has been supported by a grant from the Technical College of the Balearic Islands University. The second author thanks the support of the Spanish Ministry of Education and Science, and FEDER, grant MTM2006-14925-C02-01.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 530–541, 2009.

1 Introduction and Preliminaries

Throughout this paper the letters $\mathbb{R}^+$, $\mathbb{N}$, $\mathbb{Z}$ and ω will denote the set of nonnegative real numbers, the set of natural numbers, the set of integer numbers and the set of nonnegative integer numbers, respectively.

Next we give some pertinent concepts on quasi-metric spaces. Our basic reference is [4].

Following the modern terminology, by a quasi-metric on a (nonempty) set X we mean a function $d : X \times X \to \mathbb{R}^+$ such that for all $x, y, z \in X$: (i) $d(x, y) = d(y, x) = 0 \Leftrightarrow x = y$; (ii)$d(x, z) \leq d(x, y) + d(y, z)$.

Each quasi-metric d on a set X induces a T_0 topology $\mathcal{T}(d)$ on X which has as a base the family of open d-balls $\{B_d(x, r) : x \in X,\ r > 0\}$, where $B_d(x, r) = \{y \in X : d(x, y) < r\}$ for all $x \in X$ and $r > 0$.

A quasi-metric space is a pair (X, d) such that X is a (nonempty) set and d is a quasi-metric on X.

If d is a quasi-metric on a set X, then the function d^s defined on $X \times X$ by $d^s(x, y) = \max\{d(x, y), d(y, x)\}$ is a metric on X.

A quasi-metric is called bicomplete if d^s is a complete metric.

According to [5] a quasi-metric on X is said to be weightable if there is a function $w : X \to \mathbb{R}^+$ such that $d(x, y) + w(x) = d(y, x) + w(y)$ for all $x, y \in X$. The function w is said to be a weight function for the quasi-metric space (X, d).

An interesting example of a bicomplete weightable quasi-metric space is given by the pair $(\mathbb{R}^+, u)$ where the function $u : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ is defined by $u(x, y) = (y - x) \vee 0$ for all $x, y \in \mathbb{R}^+$. The pair $((0, +\infty], u_{-1})$, where the function $u_{-1} : (0, +\infty] \times (0, +\infty] \to \mathbb{R}^+$ is given by $u_{-1}(x, y) = (\frac{1}{y} - \frac{1}{x}) \vee 0$ for all $x, y \in (0, +\infty]$ (we adopt the convention that $\frac{1}{+\infty} = 0$), is another interesting example of bicomplete weightable quasi-metric space . Both quasi-metric spaces play a central role in the theory of complexity spaces such as we will state later on (see also [12,10,9] for more details).

As usual a partial order (or simply an order) on a (nonempty) set X is a reflexive, transitive and antisymmetric binary relation $\leq$ on X. A set X equipped with an order is said to be an ordered set. Our basic references for Order Theory are [2] and [1].

A paradigmatic and well-known example of ordered set is the set of partial mappings. Let us recall that, given a nonempty set X, a mapping from X to X is a partial (total) mapping if $\mathrm{dom}(f) \subset X$ ($\mathrm{dom}(f) = X$). Where we denote by $\mathrm{dom}(f)$ the domain of f. The set of partial mappings from $\mathbb{Z}$ to $\mathbb{Z}$, denoted by $[\mathbb{Z} \to \mathbb{Z}]$, becomes an ordered set endowed with the extension order $\sqsubseteq$ given by $f \sqsubseteq g \Leftrightarrow \mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and $f = g$ on $\mathrm{dom}(f)$.

The least upper bound of a subset S of $(X, \leq)$ is denoted by $\mathrm{lub}(S)$ if it exists.

We say that a sequence $(x_n)_{n\in\mathbb{N}}$ in an ordered set $(X, \leq)$ is ascending if $x_n \leq x_{n+1}$ for all $n \in \mathbb{N}$.

It is well-known that a quasi-metric space (X, d) can be endowed with an order $\leq_d$ which is defined by $x \leq_d y \Leftrightarrow d(x, y) = 0$.

In [12], M. Schellekens introduced the (quasi-metric) complexity space as a part of the development of a topological foundation for the complexity analysis

of programs and algorithms. In particular, he presented some applications of this theory to the complexity analysis of Divide & Conquer algorithms. More concretely he gave a novel proof, based on fixed point arguments, of the well-known fact that the mergesort algorithm has optimal asymptotic average running time.

The complexity space is the pair $(\mathcal{C}, d_{\mathcal{C}})$, where

$$\mathcal{C} = \{f : \omega \to (0, +\infty] : \sum_{n=0}^{\infty} 2^{-n} \frac{1}{f(n)} < +\infty\},$$

and $d_{\mathcal{C}}$ is the quasi-metric on $\mathcal{C}$ defined by $d_{\mathcal{C}}(f, g) = \sum_{n=0}^{\infty} 2^{-n}[(\frac{1}{g(n)} - \frac{1}{f(n)}) \vee 0]$.

According to Section 4 of [12], the intuition behind the complexity distance between two functions $f, g \in \mathcal{C}$ is that $d_{\mathcal{C}}(f, g)$ measures relative progress made in lowering the complexity by replacing any program P with complexity function f by any program Q with complexity function g. Therefore, if $f \neq g$, the condition $d_{\mathcal{C}}(f, g) = 0$ can be interpreted as f is "more efficient" than g. In fact $f \leq_{d_{\mathcal{C}}} g \Leftrightarrow f(n) \leq g(n)$ for all $n \in \omega$. Moreover, the quasi-metric space $(\mathcal{C}, d_{\mathcal{C}})$ is bicomplete and weightable with weight function $W_{d_{\mathcal{C}}}$ defined by $W_{d_{\mathcal{C}}}(f) = \sum_{n=0}^{\infty} 2^{-n} \frac{1}{f(n)}$ for all $f \in \mathcal{C}$.

Later on, S. Romaguera and Schellekens ([10]) introduced the so-called dual complexity space and obtained several quasi-metric properties of the complexity space that are interesting from a computational point of view, such as the Smyth completeness, via the analysis of its dual.

Recall that the dual complexity space is the pair $(\mathcal{C}^*, d_{\mathcal{C}^*})$, where

$$\mathcal{C}^* = \{f : \omega \to \mathbb{R}^+ : \sum_{n=0}^{\infty} 2^{-n} f(n) < +\infty\},$$

and $d_{\mathcal{C}^*}$ is the quasi-metric on $\mathcal{C}^*$ given by $d_{\mathcal{C}^*}(f, g) = \sum_{n=0}^{\infty} 2^{-n}[(g(n) - f(n)) \vee 0]$.

A motivation for the use of the dual complexity space is given by the fact that Romaguera and Schellekens proved that the complexity analysis of algorithms can be carried out by means of techniques based on the dual complexity space when the considered complexity measure is the running time of computing. In this case the computational interpretation of the condition $d_{\mathcal{C}^*}(f, g) = 0$ is provided by the fact $f \leq_{d_{\mathcal{C}^*}} g \Leftrightarrow g(n) \leq f(n)$ for all $n \in \omega$. Thus $d_{\mathcal{C}^*}(f, g) = 0$ provides that g is more efficient than f. Moreover, the dual complexity space has an advantage with respect to the original one. In the dual context, and contrary to the case of the complexity space, there is a minimum which corresponds to the minimum of semantic domains. Let us recall that the minimum plays a central role in domain theory in order to model in an appropriate way the mathematical meaning of recursive definitions of procedures. Furthermore, the dual complexity space admits a more robust mathematical structure than the original complexity space. In particular, the dual complexity space has a cone structure while that of the complexity space is only a semigroup without neutral element (for a detailed discussion we refer the reader to [7]). As in the case of the complexity space, the

quasi-metric space $(\mathcal{C}^*, d_{\mathcal{C}^*})$ is bicomplete and weightable with weight function $W_{d_{\mathcal{C}^*}}$ defined by $W_{d_{\mathcal{C}^*}}(f) = \sum_{n=0}^{\infty} 2^{-n} f(n)$ for all $f \in \mathcal{C}^*$.

Most recently, Romaguera and Schellekens presented a general theory of complexity spaces which unifies in a same framework the theory of the original complexity space and the dual one (see [9]).

Following [9], given a quasi-metric space (X, d) and a fixed $x_0 \in X$, the (generalized) complexity space of (X, d, x_0) is the quasi-metric space $(\mathcal{C}^{\omega}_{X,x_0}, d_{\mathcal{C}^{\omega}_{X,x_0}})$, where

$$\mathcal{C}^{\omega}_{X,x_0} = \{f : \omega \to X : \sum_{n=0}^{\infty} 2^{-n} d^s(x_0, f(n)) < +\infty\}$$

and the quasi-metric $d_{\mathcal{C}^*_{X,x_0}}$ on $\mathcal{C}^{\omega}_{X,x_0}$ is defined by

$$d_{\mathcal{C}^{\omega}_{X,x_0}}(f, g) = \sum_{n=0}^{\infty} 2^{-n} d\,(f(n), g(n))$$

for all $f, g \in \mathcal{C}^{\omega}_{X,x_0}$.

Notice that if we take in the preceding definition the base quasi-metric space (X, d) with $X = (0, +\infty]$, $x_0 = +\infty$ and $d = u_{-1}$, then the complexity space $(\mathcal{C}^*_{X,x_0}, d_{\mathcal{C}^*_{X,x_0}})$ is exactly $(\mathcal{C}, d_{\mathcal{C}})$. Moreover, when $X = \mathbb{R}^+$, $x_0 = 0$ and $d = u$ we retrieve the dual complexity space $(\mathcal{C}^*, d_{\mathcal{C}^*})$ as a particular case of the above construction.

For the purpose of our work in this paper we need to consider a more general structure. We will replace the set of nonnegative integer numbers ω by the set of integer numbers $\mathbb{Z}$ in the definition of $\mathcal{C}^{\omega}_{X,x_0}$ above. In particular given a quasi-metric space (X, d) and $x_0 \in X$ we define the complexity space of (X, d, x_0) as the pair $(\mathcal{C}_{X,x_0}, d_{\mathcal{C}_{X,x_0}})$, where

$$\mathcal{C}_{X,x_0} = \{f : \mathbb{Z} \to X : \sum_{n=0}^{\infty} 2^{-n} [d^s(x_0, f(n)) + d^s(x_0, f(-n))] < +\infty\}$$

and $d_{\mathcal{C}_{X,x_0}}$ is the nonnegative real valued function defined on $\mathcal{C}_{X,x_0} \times \mathcal{C}_{X,x_0}$ by

$$d_{\mathcal{C}_{X,x_0}}(f, g) = \sum_{n=0}^{\infty} 2^{-n} [d\,(f(n), g(n)) + d\,(f(-n), g(-n))]$$

A trivial verification shows that the pair $(\mathcal{C}_{X,x_0}, d_{\mathcal{C}_{X,x_0}})$ is a quasi-metric space.

Of course $\mathcal{C}^{\omega}_{X,x_0} \subset \mathcal{C}_{X,x_0}$, since each mapping $g \in \mathcal{C}^{\omega}_{X,x_0}$ can be identified with a mapping $f_g \in \mathcal{C}_{X,x_0}$ given by $f_g(n) = g(n)$ and $f_g(-n) = x_0$ for all $n \in \omega$. Furthermore, $d_{\mathcal{C}^{\omega}_{X,x_0}} | \mathcal{C}_{X,x_0}(f, g) = d_{\mathcal{C}^{\omega}_{X,x_0}}(f, g)$ for all $f, g \in \mathcal{C}^{\omega}_{X,x_0}$.

In [9] the authors studied from a theoretical point of view several properties of the mathematical structure of the generalized complexity spaces $(\mathcal{C}^{\omega}_{X,x_0}, d_{\mathcal{C}^{\omega}_{X},x_0})$ as, for instance and among others, the bicompleteness and the Smyth completeness. Although these new structures are inspired by computational topics they did not apply them to Computer Science. Motivated by this fact, our aim in

this paper is to show that the generalized complexity structures are a suitable framework to model several processes that arise in a natural way in Computer Science. In particular we show in Section 3 that the new complexity spaces $(\mathcal{C}_{X,x_0}, d_{\mathcal{C}_{X,x_0}})$ are an appropriate theoretical tool to formal analysis and specification of the semantical aspects of high-level programming languages in the spirit of semantics theory developed by D.S. Scott. Some advantages of the use of our new approach with respect to the use of the classical one are shown. Section 2 is devoted to present the mathematical results which play a fundamental role in order to develop the mentioned application.

2 The Mathematical Results

The next result was proven by Romaguera and Schellekens in [9].

Theorem 1. *Let (X, d) be a bicomplete quasi-metric space and let $x_0 \in X$. Then the complexity space $(\mathcal{C}^{\omega}_{X,x_0}, d_{\mathcal{C}^{\omega}_{X,x_0}})$ is bicomplete.*

Next we extend the preceding result to the context of our new complexity spaces. We omit the proof because of it can be obtained following similar arguments to those given in [9].

Theorem 2. *Let (X, d) be a bicomplete quasi-metric space and let $x_0 \in X$. Then the complexity space $(\mathcal{C}_{X,x_0}, d_{\mathcal{C}_{X,x_0}})$ is bicomplete.*

The next result will play a fundamental role in our later work.

Proposition 1. *Let (X, d) be a weightable quasi-metric space with weight function w and let $x_0 \in X$. Then the complexity space $(\mathcal{C}_{X,x_0}, d_{\mathcal{C}_{X,x_0}})$ is weightable with weight function $W_{\mathcal{C}_{X,x_0}}$ given for all $f \in \mathcal{C}_{X,x_0}$ by* $W_{\mathcal{C}_{X,x_0}}(f) = \sum_{n=0}^{\infty} 2^{-n}[w(f(n)) + w(f(-n))]$.

Proof. Since (X, d) is a weightable quasi-metric space there exists a weight function $w : X \to \mathbb{R}^+$ such that $d(x, y) + w(x) = d(y, x) + w(y)$ for all $x, y \in X$. So $d(x_0, x) + w(x_0) = d(x, x_0) + w(x)$ for all $x \in X$. Consequently $w(x) = d(x_0, x)+w(x_0)-d(x, x_0) \leq d(x_0, x)+w(x_0)$ for all $x \in X$. Thus, given $f \in \mathcal{C}_{X,x_0}$, we have that

$$\sum_{n=0}^{\infty} 2^{-n}[w(f(n)) + w(f(-n)] \leq$$
$$\sum_{n=0}^{\infty} 2^{-n}[d(x_0, f(n)) + d(x_0, f(-n))] + 4w(x_0) < +\infty.$$

Define $W_{\mathcal{C}_{X,x_0}} : \mathcal{C}_{X,x_0} \to \mathbb{R}^+$ by $W_{\mathcal{C}_{X,x_0}}(f) = \sum_{n=0}^{\infty} 2^{-n}[w(f(n)) + w(f(-n))]$. Obviously, from the preceding inequality $W_{\mathcal{C}_{X,x_0}}$ is well-defined.

A straightforward computation shows that

$$d_{\mathcal{C}_{X,x_0}}(f, g) + W_{\mathcal{C}_{X,x_0}}(f) = d_{\mathcal{C}_{X,x_0}}(g, f) + W_{\mathcal{C}_{X,x_0}}(g).$$

The proof is complete. □

Corollary 1. *Let (X,d) be a weightable quasi-metric space and let $x_0 \in X$. Then the complexity space $(\mathcal{C}^{\omega}_{X,x_0}, d_{\mathcal{C}^{\omega}_{X,x_0}})$ is weightable with weight function $W_{\mathcal{C}^{\omega}_{X,x_0}}$ given by $W_{\mathcal{C}^{*}_{X,x_0}}(f) = \sum_{n=0}^{\infty} 2^{-n} w(f(n))$ for all $f \in \mathcal{C}^{\omega}_{X,x_0}$.*

In Computer Science when a program uses a recursion to find the solution of a problem in each step of the computation we obtain an approximation of the mentioned solution which is better than the approximations obtained in the preceding steps and, in addition, the computing process must always obtain as "limit" the final approximation of the problem solution. A mathematical theory of computation which provides a satisfactory model to this sort of situations was developed by D.S. Scott which is based on ideas from order theory and topology (see, for instance, [13] and [14]). In particular the order represents some notion of information in such a way that each step of the computation is identified with an element of the mathematical model which is greater than (or equal to) the other ones associated to the preceding steps, since each approximation gives more information about the final solution than the those computed before. The final output of the computational process is seen as the limit of the successive approximations. Thus the recursion processes are modeled as increasing sequences of elements of the ordered set which converge to its least upper bound with respect the given topology. From an information theory point of view the least upper bound captures the amount of information defined by the increasing sequence, i.e. the least upper bound gives the total information provided by the elements of the increasing sequence, and it does not contain more information than can be obtained from the elements of the increasing sequence.

In [5], S.G. Matthews introduced the notion of Scott-like topology as a mathematical framework to model increasing information content sequences in Computer Science.

According to [5], a weakly order consistent topology over an ordered set $(X,\leq)$ is a topology $\mathcal{T}$ over X such that $x \leq y \Leftrightarrow x \in \mathrm{cl}(y)$ for all $x,y \in X$, where by $\mathrm{cl}(y)$ we denote the closure of y with respect to $\mathcal{T}$. Moreover, a Scott-like topology over an ordered set $(X,\leq)$ is a weakly consistent topology $\mathcal{T}$ over X satisfying the following properties:

(a) every increasing sequence $(x_n)_{n\in\mathbb{N}}$ in $(X,\leq)$ has least upper bound $\mathrm{lub}((x_n)_{n\in\mathbb{N}})$.
(b) for every $O \in \mathcal{T}$ with $\mathrm{lub}((x_n)_{n\in\mathbb{N}}) \in O$ there exists $n_0 \in \mathbb{N}$ such that $x_n \in O$ for all $n > n_0$.

The next result was given by Matthews in [5].

Proposition 2. *Let (X,d) be a bicomplete weightable quasi-metric space with weight function w. Then the topology $\mathcal{T}(d)$ is a Scott-like toplogy over $(X \leq_d)$, and thus every ascending sequence $(x_n)_{n\in\mathbb{N}}$ in X has a least upper bound $x \in X$ with $lim_{n\to+\infty} x_n = x$ in (X,d) and $lim_{n\to+\infty} w(x_n) = w(x)$.*

Remark 1. Note that the fact that the sequence $(x_n)_{n\in\mathbb{N}}$ is ascending with least upper bound x provides that $d(x_n,x) = 0$. So in the preceding proposition we actually have that $\lim_{n\to+\infty} x_n = x$ in (X,d^s).

Proposition 3. *Let (X, d) be a bicomplete weightable quasi-metric space with weight function w. If $(x_n)_{n\in\mathbb{N}}$ is an ascending sequence in $(X, \leq_d)$ and $x \in X$ is its least upper bound, then $w(x) = inf_{n\in\mathbb{N}}(w(x_n))$.*

Proof. Since $d(x_n, x_{n+1}) = 0$ for all $n \in \mathbb{N}$ we have that $w(x_{n+1}) - w(x_n) = -d(x_{n+1}, x_n) \leq 0$. Whence we deduce that $w(x_{n+1}) \leq w(x_n)$ for all $n \in \mathbb{N}$. So the sequence $(w(x_n))_{n\in\mathbb{N}}$ is a decreasing sequence in $(\mathbb{R}^+, |\cdot|)$ with lower bound. It immediately follows that there exists $a \in \mathbb{R}^+$ such that $a = \inf_{n\in\mathbb{N}} w(x_n)$ and $\lim_{n\to+\infty} w(x_n) = a$. By Proposition 2 we have that $\lim_{n\to+\infty} w(x_n) = w(x)$. Therefore $w(x) = a$. □

From Theorem 2, Proposition 1, Proposition 2, Remark 1 and Proposition 3 we immediately deduce the following result.

Theorem 3. *Let (X, d) be a bicomplete weightable quasi-metric space and let $x_0 \in X$. The every ascending sequence $(f_n)_{n\in\mathbb{N}}$ in $(\mathcal{C}_{X,x_0}, \leq_{d_{\mathcal{C}_{X,x_0}}})$ has least upper bound $f \in \mathcal{C}_{X,x_0}$ satisfying:*

(1) $lim_{n\to+\infty} f_n = f$ in $(\mathcal{C}_{X,x_0}, d^s_{\mathcal{C}_{X,x_0}})$.
(2) $W_{\mathcal{C}_{X,x_0}}(f) = inf_{n\in\mathbb{N}} W_{\mathcal{C}_{X,x_0}}(f_n)$.
(3) $lim_{n\to+\infty} W_{\mathcal{C}_{X,x_0}}(f_n) = W_{\mathcal{C}_{X,x_0}}(f)$.

Corollary 2. *Let (X, d) be a bicomplete weightable quasi-metric space and let $x_0 \in X$. The every ascending sequence $(f_n)_{n\in\mathbb{N}}$ in $(\mathcal{C}^\omega_{X,x_0}, \leq_{d^\omega_{\mathcal{C}_{X,x_0}}})$ has least upper bound $f \in \mathcal{C}^\omega_{X,x_0}$ satisfying:*

(1) $lim_{n\to+\infty} f_n = f$ in $(\mathcal{C}^\omega_{X,x_0}, d^s_{\mathcal{C}^\omega_{X,x_0}})$.
(2) $W_{\mathcal{C}^\omega_{X,x_0}}(f) = inf_{n\in\mathbb{N}} W_{\mathcal{C}^\omega_{X,x_0}}(f_n)$.
(3) $lim_{n\to+\infty} W_{\mathcal{C}^\omega_{X,x_0}}(f_n) = W_{\mathcal{C}^\omega_{X,x_0}}(f)$.

Remark 2. Notice that the preceding theorem guarantees that the topologies $\mathcal{T}(d_{\mathcal{C}_{X,x_0}})$ and $\mathcal{T}(d_{\mathcal{C}^\omega_{X,x_0}})$ are Scott-like over $(\mathcal{C}_{X,x_0}, \leq_{d_{\mathcal{C}_{X,x_0}}})$ and $(\mathcal{C}^\omega_{X,x_0}, \leq_{d^\omega_{\mathcal{C}_{X,x_0}}})$, respectively.

In [5] it has been pointed out that the weight w of a weightable quasi-metric space (X, d) can be used as a tool to describe the amount of information contained in an element of the quasi-metric space. In particular if $x, y \in X$ with $d(x, y) = 0$ $(x \leq_d y)$ then $w(x) \geq w(y)$, i.e. if y has at least as much information as x then $w(x) \geq w(y)$. So the numerical value $w(x)$ can be considered as the information content in the element $x \in X$.

Consider the dual complexity space $(\mathcal{C}^*, d_{\mathcal{C}^*})$ (see Section 1). It is clear that

$$f \leq_{d_{\mathcal{C}^*}} g \Leftrightarrow d_{\mathcal{C}^*}(f, g) = 0 \Leftrightarrow g(n) \leq f(n) \text{ for all } n \in \omega.$$

So the relation $f \leq_{d_{\mathcal{C}^*}} g$ can be interpreted as an information order where the information content is identified with the "degree" of complexity. In fact, $f \leq_{d_{\mathcal{C}^*}} g$ can be interpreted as "all the information contained in f is contained in

g" in the sense that the complexity associated to g is less than the associated one to f. Furthermore, the value $W_{d_{\mathcal{C}^*}}(f) = \sum_{n=0}^{\infty} 2^{-n} f(n)$ gives a global measure of the complexity of a program with associated complexity function f (actually it is a kind of distance from f to the ideal complexity function 0, with $0(n) = n$ for all $n \in \omega$). In addition if $f \leq_{d_C^*} g$ then $W_{d_{\mathcal{C}^*}}(g) \leq W_{d_{\mathcal{C}^*}}(f)$. Consequently the value $W_{d_{\mathcal{C}^*}}(f)$ can be used to describe the amount of information contained in f when we interpreted it as the mentioned degree of complexity. A similar reasoning can be made for the case of the original complexity space $(\mathcal{C}, d_{\mathcal{C}})$ where $f \leq_{d_{\mathcal{C}}} g \Leftrightarrow f(n) \leq g(n)$ for all $n \in \mathbb{N}$ and $W_{d_{\mathcal{C}}}(f) = \sum_{n=0}^{\infty} 2^{-n} \frac{1}{f(n)}$ for all $f \in \mathcal{C}$. Note that $W_{d_{\mathcal{C}}}(f)$ is the distance from f to the complexity function ∞ with $\infty(n) = +\infty$ for all $n \in \mathbb{N}$, and that $f \leq_{d_C} g$ implies $W_{d_{\mathcal{C}}}(g) \leq W_{d_{\mathcal{C}}}(f)$.

From information viewpoint, our new framework $(\mathcal{C}_{X,x_0}, d_{\mathcal{C}_{X,x_0}})$ preserves the spirit of increasing information processes. More concretely, statement (1) in Theorem 3 (and also in Corollary 2) allows us to see the least upper bound of an ascending sequence as its limit. Note that the mentioned limit is unique. Furthermore, if we take, similarly to the case of $(\mathcal{C}, d_{\mathcal{C}})$ and $(\mathcal{C}^*, d_{\mathcal{C}^*})$, the numerical value $W_{\mathcal{C}_{X,x_0}}(f)$ $(W_{\mathcal{C}^{\omega}_{X,x_0}}(f))$ as a measure of the information content of $f \in \mathcal{C}_{X,x_0}$ $(f \in \mathcal{C}^{\omega}_{X,x_0})$, then statements (2) and (3) in Theorem 3 (and in Corollary 2) ensure that the information content of the least upper bound of an ascending sequence is exactly the information content that can be derived from each element of the sequence but that it does not contain more.

3 Complexity Spaces in Denotational Semantics

In Denotational Semantics one of the aims consists of giving mathematical models of programming languages in such a way that the meaning of a procedure can be obtained as an element of the constructed model. In particular most programming languages allow to construct procedures by means of recursive definitions in such a way that the meaning of such definition is employed in its own definition. In order to analyse if a such recursive definition of a procedure is meaningful it is necessary to make use of a mathematical approach in which the meaning of such denotational specification is obtained as the least upper bound of a sequence of generated approximations in which each one of approximations gives more information about the meaning of the procedure than the preceding ones.

Below we give a typical example of the mentioned situation. Consider a while-loop, i.e. "**while** B **do** C". It is obvious that its denotation $\langle$**while** B **do** $C\rangle$, expressed by means of sequential composition of commands, should satisfy the next equation:

$$\langle \textbf{while } B \textbf{ do } C \rangle = \langle \textbf{if } B \textbf{ then } C; (\textbf{while } B \textbf{ do } C) \textbf{ else skip} \rangle$$

Of course the preceding denotational specification presents a handicap because to define $\langle$**while** B **do** $C\rangle$ we need to use in both sides of the above equation the same "object" whose meaning we are trying to define. In order to avoid

this disadvantage is constructed a sequence of partial mappings $(w_n(B,C))_{n\in\mathbb{N}}$, where $w_n(B,C)$ coincides with the computation of "**while** B **do** C" involving fewer than n iterations of the loop and is undefined otherwise. So we obtain an increasing (with respect the order of partial mappings) information sequence of approximations of the meaning of the while-loop in which each one of them can be specified without recursion. Moreover, the least upper bound of $(w_n(B,C))_{n\in\mathbb{N}}$ matches up with a mapping (partial or total) that can been identified with the meaning of the while-loop.

In order to show that our developed theory is useful to model while-loop denotational specification processes following the successive approximation scheme explained in Section 2, we apply it to a particular case of the preceding general problem.

Let us consider the following while-loop:

$$\textbf{while } X > 0 \textbf{ do } (Y := X * Y; X := X - 1), \tag{1}$$

where the storage variables X and Y take a nonnegative integer value and an integer one, respectively. Notice that the input values of X and Y are different in general. It is obvious that the output of (1) is exactly $(0,(x!)*y)$ for all $x \in \omega$ and for all $y \in \mathbb{Z}$, where x and y stand for the input values of X and Y, respectively.

3.1 The Classical Approach

The classical technique that solves the while-loop problem is based on the so-called "fixed point induction" whose spirit has been explained above. Next we recall what it involves from a mathematical viewpoint.

Consider the usual order $\sqsubseteq$ defined on the set of total and partial mappings $[\mathbb{Z}\times\mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}]$. We assume, as usual, that $[\mathbb{Z}\times\mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}]$ contains the totally undefined partial mapping $\perp$. Obviously $\perp \sqsubseteq w$ for all $w \in [\mathbb{Z}\times\mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}]$.

The denotation of (1) is defined to be an element of $[\mathbb{Z}\times\mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}]$ that we will denote by v_∞ and that is a fixed point of the below equation

$$f(v)(x,y) = \begin{cases} (x,y) & \text{if } x < 1 \\ v(x-1, x*y) & \text{if } x \geq 1 \end{cases}, \tag{2}$$

where $f : [\mathbb{Z}\times\mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}] \to [\mathbb{Z}\times\mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}]$.

Define the sequence $(v_n)_{n\in\mathbb{N}}$ in $[\mathbb{Z}\times\mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}]$ given by

$$v_n(x,y) = \begin{cases} (x,y) & \text{if } x < 1 \\ (0,(x!)*y) & \text{if } 1 \leq x < n \\ \text{undefined} & \text{if } x \geq n \end{cases}$$

for all $n \in \mathbb{N}$. The sequence $(v_n)_{n\in\mathbb{N}}$ is clearly increasing, i.e. $v_n \sqsubseteq v_{n+1}$ for all $n \in \mathbb{N}$. Moreover, the least upper bound of $(v_n)_{n\in\mathbb{N}}$ with respect to $\sqsubseteq$ is exactly

$$v_\infty(x,y) = \begin{cases} (x,y) & \text{if } x < 1 \\ (0,(x!)*y) & \text{if } x \geq 1 \end{cases}.$$

Observe that $f(v_\infty) = v_\infty$.

Note that in this approach the sequence $(v_n)_{n\in\mathbb{N}}$ is built up applying the iterations of the mapping f over the undefined symbol $\perp$, i.e. $v_n = f^n(\perp)$.

3.2 Our Approach

In this subsection we show that the context of complexity spaces is useful to model the meaning of (1) in the spirit of the classical approach of successive approximations.

Let $\Sigma = \mathbb{Z}\times\mathbb{Z}$. Denote by Σ^ω the set of all finite and infinite sequences (words) over Σ. We assume that the empty sequence $\varnothing$ is an element of Σ^ω. Denote by $\sqsubseteq$ the prefix order on Σ^ω, i.e. $x \sqsubseteq y \Leftrightarrow x$ is a prefix of y. The empty word $\varnothing$ is considered as a prefix of the rest of words in Σ^ω (see [2], page 16). The pair $(\Sigma^\omega, \sqsubseteq)$ is known as the domain of words over the alphabet $\mathbb{Z}\times\mathbb{Z}$.

On the other hand, given $x, y \in \Sigma^\omega$, we denote by $x \sqcap y$ the longest common prefix of x and y. Moreover, for each $x \in \Sigma^\omega$ the length of x will be denoted by $\ell(x)$. Thus, $\ell(x) \in [1, +\infty]$ whenever $x \in \Sigma^\omega$ with $x \neq \varnothing$, and $\ell(x) = 0$ if $x = \varnothing$.

Following [5] and [4] we define on Σ^ω the function $d_{\Sigma^\omega} : \Sigma^\omega \times \Sigma^\omega \to \mathbb{R}^+$ by $d_{\Sigma^\omega}(x, y) = 2^{-\ell(x\sqcap y)} - 2^{-\ell(x)}$. It is well-known that $(\Sigma^\omega, d_{\Sigma^\omega})$ is a bicomplete weightable quasi-metric space with weight function $w : \Sigma^\omega \to \mathbb{R}^+$ given by $w(x) = 2^{-\ell(x)}$ for all $x \in \Sigma^\omega$, where we adopt the convention that $2^{-\ell(+\infty)} = 0$. Furthermore, $x \sqsubseteq y \Leftrightarrow d_{\Sigma^\omega}(x, y) = 0$. So the order $\leq_{d_{\Sigma^\omega}}$ coincides with the prefix order on Σ^ω. Note that $\varnothing \leq_{d_{\Sigma^\omega}} x$ for all $x \in \Sigma^\omega$. Recent works, with applications to Symbolic Computation and Complexity Theory, related to the former framework can be found in ([3], [8], [11], [6]).

Now for each $n \in \mathbb{N}$ define $f_n : \mathbb{Z} \to \Sigma^\omega$ by

$$f_n(k) = \begin{cases} (k, y) & \text{if } k < 1 \\ (k-1, k*y)(k-2, k*(k-1)*y)...(0, (k!)*y) & \text{if } 1 \leq k < n \\ \varnothing & \text{if } k \geq n \end{cases}.$$

Hence the sequence $(f_n)_{n\in\mathbb{N}}$ is in $\mathcal{C}_{\Sigma^\omega,\varnothing}$, since

$$\sum_{k=0}^{\infty} 2^{-k}[d^s_{\Sigma^\omega}(\varnothing, f_n(k)) + d^s_{\Sigma^\omega}(\varnothing, f_n(-k))] = \sum_{k=1}^{n-1}(1 - 2^{-k})2^{-k} + \frac{3}{2} < +\infty.$$

Moreover $d_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_n, f_{n+1}) = 0$ for all $n \in \mathbb{N}$, since $f_n(k) \sqsubseteq f_{n+1}(k)$ for all $k \in \mathbb{Z}$ and $n \in \mathbb{N}$. It follows that the sequence $(f_n)_{n\in\mathbb{N}}$ is increasing in $(\mathcal{C}_{\Sigma^\omega,\varnothing}, \leq_{d_{\mathcal{C}_{\Sigma^\omega,\varnothing}}})$. Applying statement (1) in Theorem 3 we have that there exists a unique $f \in \mathcal{C}_{\Sigma^\omega,\varnothing}$ such that $\lim_{n\to+\infty} f_n = f$ in $(\mathcal{C}_{\Sigma^\omega,\varnothing}, d^s_{\mathcal{C}_{\Sigma^\omega,\varnothing}})$. Furthermore f is the least upper bound of $(f_n)_{n\in\mathbb{N}}$. So $f_n \leq_{d_{\mathcal{C}_{\Sigma^\omega,\varnothing}}} f$ for all $n \in \mathbb{N}$. Consequently $f_n(k) \sqsubseteq f(k)$ for all $k \in \mathbb{Z}$ and $n \in \mathbb{N}$.

The function f_∞ given by

$$f_\infty(k) = \begin{cases} (k, y) & \text{if } k < 1 \\ (k-1, k*y)(k-2, k*(k-1)*y)...(0, (k!)*y) & \text{if } k \geq 1 \end{cases}$$

is an element of $\mathcal{C}_{\Sigma^\omega,\varnothing}$ because of

$$\sum_{k=0}^{\infty} 2^{-k}[d^s_{\Sigma^\omega}(\varnothing, f_\infty(k)) + d^s_{\Sigma^\omega}(\varnothing, f_\infty(-k))] \leq \frac{5}{2}.$$

On the other hand, it is clear that $f_n \leq_{d_{\mathcal{C}_{\Sigma^\omega,\varnothing}}} f_\infty$ from $d_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_n, f_\infty) = 0$ for all $n \in \mathbb{N}$. It follows that $f \leq_{d_{\mathcal{C}_{\Sigma^\omega,\varnothing}}} f_\infty$, and hence $d_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f, f_\infty) = 0$.

Next we show that $\lim_{n\to+\infty} f_n = f_\infty$ in $(\mathcal{C}_{\Sigma^\omega,\varnothing}, d^s_{\mathcal{C}_{\Sigma^\omega,\varnothing}})$. Indeed, given $\varepsilon > 0$ there exists $n_0 \in \mathbb{N}$ such that $\sum_{k\geq n_0}^{+\infty} 2^{-k} < \varepsilon$. Thus we have that

$$d_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_\infty, f_n) \leq \sum_{k\geq n}^{+\infty} 2^{-k} \leq \sum_{k\geq n_0}^{+\infty} 2^{-k} < \varepsilon$$

for all $n \geq n_0$. Since $d_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_n, f_\infty) = 0$ for all $n \in \mathbb{N}$ we deduce immediately the desired conclusion. Taking into account that $\lim_{n\to+\infty} f_n = f$ in $(\mathcal{C}_{\Sigma^\omega,\varnothing}, d^s_{\mathcal{C}_{\Sigma^\omega,\varnothing}})$ we conclude that $f = f_\infty$.

Observe that in the classical context each v_n gives only the output values of the storage variables X and Y at the final state of the while-loop (1) after $n-1$ iterations. However, in our new context every complexity function f_n gives, contrarily to the classical context, the information of the all intermediate steps of the computation of the while-loop (1) involving fewer than n iterations of itself. In addition, the complexity function f_∞ is exactly the meaning of the while-loop (1) matching up with the total computation (not only with the final output) of $((x!) * y)$ for all $x \in \omega$ and $y \in \mathbb{Z}$. Furthermore, we point out that in the classical approach the meaning of the (1), v_∞ is taken as the least upper bound of the iterations sequence, which coincides with the least fixed point of equation (2). This is done because in general one can find while-loops whose associated fixed point equation has not a unique solution. Note that this is not the case of (2). However in our context of generalized complexity spaces (incorporating topology) the increasing sequence of iterations has always a unique limit, say f_∞, which will be the natural candidate for the solution of the fixed point equation that will provide the meaning of the while-loop.

Finally we remark that

$$W_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_\infty) = \sum_{k=0}^{\infty} 2^{-k}[2^{-\ell(f_\infty(k))} + 2^{-\ell(f_\infty(-k))}] = \frac{3}{2} + \sum_{k=1}^{\infty} 2^{-2k},$$

and that

$$W_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_n) = \sum_{k=0}^{\infty} 2^{-k}[2^{-\ell(f_n(k))} + 2^{-\ell(f_n(-k))}] = \frac{3}{2} + \sum_{k=1}^{n-1} 2^{-2k} + \sum_{k=n}^{\infty} 2^{-k-1}$$

for all $n \in \mathbb{N}$. Of course note that $W_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_{n+1}) \leq W_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_n)$ for all $n \in \mathbb{N}$, $W_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_\infty) = \inf_{n\in\mathbb{N}} W_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_n)$ and that $W_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_\infty) = \lim_{n\to+\infty} W_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_n)$ such as we have established in Theorem 3. Moreover $W_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_n)$ measures the

information content of the function f_n in the sense that the smaller $W_{\mathcal{C}_{\Sigma^\omega,\varnothing}}(f_n)$ the more information f_n contains about f_∞ (the meaning of the while-loop). Thus the preceding equalities guarantee that f_∞ captures the amount of information of the sequence $(f_n)_{n\in\mathbb{N}}$ but f_∞ does not contain more information that can be derived from each f_n.

References

1. Abramsky, S., Jung, A.: Domain theory. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.) Handbook of Logic in Computer Science, vol. III, Oxford University Press, Oxford (1994)
2. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order. Cambridge University Press, Cambridge (1990)
3. Kahn, G.: The semantics of a simple language for parallel processing. In: Proc. IFIP Congress, vol. 74, pp. 471–475. Elsevier, North-Holland, Amsterdam (1974)
4. Künzi, H.P.A.: Nonsymmetric topology. In: Proc. Colloquium on topology, Szekszárd, Hungary (1993); Colloq. Math. Soc. János Bolyai Math. Studies 4, 303-338 (1995)
5. Matthews, S.G.: Partial metric topology. In: Proc. 8th Summer Conference on General Topology and Applications, Ann. New York Acad. Sci., vol. 728, pp. 183-197 (1994)
6. Rodríguez-López, J., Romaguera, S., Valero, O.: Denotational semantics for programming languages, balanced quasi-metrics and fixed points. Internat. J. Comput. Math. 85, 623–630 (2008)
7. Romaguera, S., Sánchez-Pérez, E.A., Valero, O.: Computing complexity distances between algorithms. Kybernetika 39, 569–582 (2003)
8. Romaguera, S., Sapena, A., Tirado, P.: The Banach fixed point theorem in fuzzy quasi-metric spaces with application to the domain of words. Topology Appl. 154, 2196–2203 (2007)
9. Romaguera, S., Schellekens, M.: The quasi-metric of complexity convergence. Quaestiones Mathematicae 23, 359–374 (2000)
10. Romaguera, S., Schellekens, M.: Quasi-metric properties of complexity spaces. Topology Appl. 98, 311–322 (1999)
11. Romaguera, S., Valero, O.: On the structure of the space of complexity partial functions. Internat. J. Comput. Math. 85, 631–640 (2008)
12. Schellekens, M.: The Smyth completion: a common foundation for the denotational semantics and complexity analysis. In: Proc. MFPS 11, Electronic Notes in Theoret. Comput. Sci., vol. 1, pp. 211-232 (1995)
13. Scott, D.S.: Outline of a mathematical theory of computation. In: Proc. 4th Annual Princeton Conference on Information Sciences and Systems, pp. 169–176 (1970)
14. Scott, D.S.: Domains for denotational semantics. In: Nielsen, M., Schmidt, E.M. (eds.) ICALP 1982. LNCS, vol. 140, pp. 577–613. Springer, Heidelberg (1982)

Segmentation Charts for Czech – Relations among Segments in Complex Sentences*

Markéta Lopatková and Tomáš Holan

Charles University in Prague, Czech Republic
lopatkova@ufal.mff.cuni.cz,
Tomas.Holan@mff.cuni.cz

Abstract. Syntactic analysis of natural languages is the fundamental requirement of many applied tasks. We propose a new module between morphological and syntactic analysis that aims at determining the overall structure of a sentence prior to its complete analysis.

We exploit a concept of segments, easily automatically detectable and linguistically motivated units. The output of the module, so-called 'segmentation chart', describes the relationship among segments, especially relations of coordination and apposition or relation of subordination.

In this text we present a framework that enables us to develop and test rules for automatic identification of segmentation charts. We describe two basic experiments – an experiment with segmentation patterns obtained from the Prague Dependency Treebank and an experiment with the segmentation rules applied to plain text. Further, we discuss the evaluation measures suitable for our task.

1 Motivation

Syntactic analysis of natural languages is the fundamental requirement of many applied tasks. The solution of this complex task is not satisfactory yet, especially for languages with free word order. Long-term efforts of many researchers brought parsers, which are quite reliable for relatively short and simple sentences. However, their reliability is significantly lower for long and complex sentences (see e.g. [1] for more citations).

A new module between morphological and syntactic analysis is a natural step capable to reduce the complexity of this task. Let us mention at least the idea of chunking [2] and cascaded parsing [3–5]. Roughly speaking, these approaches group individual tokens into more complex structures (as e.g. nominal or prepositional phrases). We propose another approach that aims at determining the overall structure of a sentence, i.e. a hierarchy of sentence segments, prior to its complete analysis. The advantage of having the estimation of sentence structure (especially for long and complex sentences) is quite obvious – it allows us to

* This paper presents the results of the project supported by the GAČR grant No. 405/08/0681 and partially also by the IS program No. 1ET100300517. The research is carried out within the project of MŠMT No. MSM0021620838.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 542–553, 2009.

exclude inappropriate relations in syntactic trees and thus the complexity of the task is substantially reduced and the parsing process is speeded up.

We exploit a concept of segments, easily automatically detectable and linguistically motivated units. Firstly, individual segments are identified; then their mutual relationship is determined. So-called 'segmentation chart' describes the relationship among segments, especially relations of coordination and apposition or relation of subordination, i.e. the relation between governing and subordinated parts of sentence; parentheses are also identified.

We slightly modify and exploit the concept of segments which has been originally proposed in [6] and modified in [7]. The initial set of rules for segmentation of Czech sentences has also been introduced there. These rules serve for identification of (nondeterministic) segmentation charts showing the relationship of individual segments in sentences.

Let us demonstrate the basic idea of segmentation on an example of Czech sentence from the news (1). At first, the sentence is split into individual segments. We consider the punctuation marks , , the coordinating conjunction *a*, the brackets (,) and the full stop . as boundaries of segments. Then we determine mutual relations of these units – we distinguish coordination, parenthesis and subordination. Thus we obtain **segmentation chart**, which allows us to identify the overall structure of the sentence.

(1) *S tím byly trochu problémy , protože starosta v řeči rád zdůrazňoval své vzdělání (však studoval až v Klatovech a v Roudnici), a Víťa tedy občas nutně trochu tápal .*
[There was a bit problem with it , as the mayor liked to stress his education in his talk (after all he studied in Klatovy and Roudnice), and thus Víťa was occasionally a bit confused .]

The first segment consists of the main clause of the complex sentence (no subordinating expression appears in this segment and there is the finite verb *byly* [were] there). This segment is placed on the basic layer (layer 0) of the segmentation chart. The second segment is introduced by the subordinating conjunction *protože* [because] and it contains the finite verb *zdůrazňoval* [(he) emphasized]. This segment is identified as a segment subordinated to the first one and thus it is placed on the lower layer (layer 1) in the chart. The opening bracket follows, which is interpreted as a beginning of a parenthesis. Thus the third segment belongs to another lower layer (layer 2). The fourth segment is separated by the coordinating conjunction *a* [and], therefore it should be at the same layer as the third segment. The third segment contains a finite verb *studoval* [studied], contrary to the fourth segment – it implies with high probability that we have the case of coordination of sentence members. Embedded parenthesis ends with the closing bracket; we climb up in the segmentation chart. The last segment contains the word *tedy* [therefore] – the triplet *, a tedy* [(comma) and therefore] is considered as a characteristic of coordination. Thus the fifth segment is analyzed as a segment coordinated to either the first or the second segment.

The segmentation chart can be expressed graphically (Fig. 1 shows one of the possible charts for the sentence (1)), or as a vector of layers (e.g., two vectors reflecting two segmentation charts (01220) and (01221) for the sentence (1)).

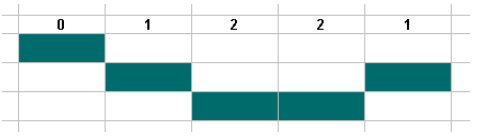


Fig. 1. Segmentation chart (01221)

Note that segmentation and analysis of segmented sentences can be formally modeled, e.g., by Parallel Communicating Grammar Systems (PCGS) and Freely Rewriting Restarting [8].

The capability to determine (reliably enough) the mutual relationship among segments and subsequently the possible structure of clauses in complex sentences prior to their full analysis would simplify the task of syntactic analysis / parsing of natural language sentences. Moreover, it appears that in a number of important application – such as information retrieval, determining the structure of documents and their main and secondary themes – there is no explicit need for full syntactic analysis. It would be of great interest to examine to which extent we can limit ourselves to the 'upper' layers of sentence structure (and ignore deeply nested segments) for such applied tasks. The achievements of similar methods for the analysis of different type of languages, e.g. [9] or [10], encourage further research in this area.

The main goal of this text is to present a framework which enables us to further develop, test and evaluate rules for automatic identification of segmentation charts. After the definition of segments and segmentation charts (Section 2), we describe two basic experiments – the experiment with obtaining segmentation patterns from tree structures stored in the Prague Dependency Treebank (Section 3.1) and the experiment with the segmentation rules applied to plain text (Section 3.2). We conclude with Section 4 where we introduce and discuss the appropriate measures for evaluating the segmentation rules. We compare segmentation charts obtained by these two sets of rules with the manually annotated sample of complex sentences and we show their limits for selected language phenomena.

2 Segment Boundaries, Segments and Segmentation Chart

An (input) sentence is understood here as a sequence of tokens $w_1 w_2 \ldots w_n$, when each token w_i represents either one word (lexical form of a given language) or one punctuation mark (comma, full stop, question mark, exclamation mark, dash, colon, semicolon, quotation marks, brackets, ...).

We do not care about dividing the text into sentences here as we dispose of tools reliable enough for sentence identification for Czech; in our experiments, we adopt it from the Prague Dependency Treebank. We also presuppose full morphological analysis of the text, i.e. we expect that each token bears its full morphological analysis.

Based on their morphological characteristics, all tokens are disjunctively divided into two groups — ordinary words and segment boundaries. After identification of boundaries, the input sentence is partitioned into individual segments.

Segment Boundaries

Boundaries are tokens and their sequences that divide a sentence into individual units referred to as segments.

In the following experiments we consider the following tokens as **elementary boundaries**:

- **punctuation marks:** comma, colon, semicolon, question mark, exclamation mark, dash (all types), opening and closing bracket (all kinds), vertical bar, quotation mark (all types), i.e. symbols *, : ; ? ! - ( )* [ ] | { } ‘ ’ “ ” , ‘ ,, “
- **punctuation ending a sentence**
- **coordinating conjunctions:** morphological tag starting with the pair J^ [11].

Several elementary boundaries may appear in a sentence following immediately one after another (as the sequence *), a* in sentence (1)). We consider a maximum sequence of such elementary boundaries as a **(compound) boundary.**

Segment S is then understood as the maximal non-empty sequence of tokens $w_1 w_2 \dots w_s$ that does not contain any boundary.

When determining the individual segments we presuppose that every sentence begins and ends with a boundary (if there is no boundary at the beginning or at the end of the sentence, we add the empty boundary there).

Let us point out that the boundaries specified on the basis of morphological analysis are not necessarily unambiguous. Punctuation marks are not ambiguous but this is not true for coordinating conjunctions (e.g. the wordform *ale* is either coordinating conjunctions [but] or it is a wordform belonging to three substantive lemmas *ala*). Thus we admit ambiguous segmentation of the sentence in general. However, there are highly reliable taggers for Czech (i.e., automated tools that are able to select exactly one morphological tag per token; the highest published accuracy for the first two positions of morphological tag is 99.36% [12]). Therefore, we disregard possible ambiguity of morphological analysis and we presuppose a unique morphological tag for each token (in the experiments described below, we take over the morphological tags from the Prague Dependency Treebank). It implies that boundaries and individual segments are defined unambiguously.

Segment Flags

Morphological analysis of the text contains a lot of more or less reliable information that can be used for identification of relationship among individual segments. This information is stored in a form of specific **flags** that are assigned to individual segments. In our experiments, we use only subordination flag, other flags as coordination flag or flag for finite verb are foreseen [6].

Subordination flag (SF). A subordination flag is assigned to a particular segment either if this segment contains any wordform with the morphological tag that begins with the following pair (for conjunctions, pronouns, and numerals [11]), or if this segment contains one of the listed pronominal adverbs:

- **subordinating conjunction:** `J,`
- **interrogative / relative pronoun:** `P4`, `PE`, `PJ`, `PK`, `PQ`, `PY`
- **numeral:** `C?`, `Cu`, `Cz`
- **pronominal adverb:** *jak, kam, kde, kdy, proč, kudy*

Segmentation Chart

The segmentation of a particular sentence can be represented by one or more **segmentation charts** that describe the mutual relationship of individual segments with regard to their coordination or subordination. A segmentation chart captures the **layer of embedding** for individual segments. The basic idea of the segmentation chart is very simple:

- Segments forming all main clauses of a complex sentence belong to the basic layer (layer 0);
- Segments forming clauses that depend on the clauses at the k-th layer obtain layer of embedding $k+1$ (i.e., layer of embedding for subordinated segments is higher than layer of segments forming their governing clause);
- Segments forming coordinated segments or segments in apposition have the same layer;
- Segments forming parentheses (e.g., sequence of wordforms within brackets) obtain layer $k+1$ compared to the layer k of their adjacent segments

3 Experiments with Automatic Identification of Segmentation Charts

3.1 How to Obtain Segments from Syntactic Tree?

This chapter explains the possible algorithm producing segmentation charts for individual sentences from their analytical trees in the Prague Dependency Treebank[1] (PDT [13]). Analytical layer of PDT captures the surface syntax. In principle, it contains the same information that may be directly used for the identification of segment layers.

[1] http://ufal.mff.cuni.cz/pdt2.0/

A sentence at the analytical layer is represented as a dependency-based tree, i.e., a connected acyclic directed graph in which no more than one edge leads from a node. The nodes – labeled with complex symbols (sets of attributes) – represent individual tokens (wordforms or punctuation marks); one token of the sentence is represented by exactly one node of the tree. The edges represent syntactic relations in the sentence (the dependency relation and the relation of coordination and apposition being the basic ones). The actual type of the relation is given as a function label of the edge, so-called analytical function. In addition, linear ordering of the nodes corresponds to the sentence word order. In particular, there are no nonterminal nodes in PDT representing more complex sentence units – such units are expressed as (dependency) subtrees.

In order to be able to present a basic set of rules, it is necessary to introduce the concept of a path between the segments and the concept of a group of segments. For the sentence W, there is **an edge from the segment S_i to the segment S_j** ($S_i, S_j \subset W$) iff there exists a pair of words $u \in S_i$ and $v \in S_j$ such that there exists a path from u to v in the dependency tree T of the sentence W.

A **path from the segment S_i to the segment S_j** of the sentence W ($S_i, S_j \subset W$) exists iff there exists a sequence of segments $S_i = S_{p_1}, \ldots, S_{p_m} = S_j$, $S_{p_k} \subset W (k = 1 \ldots m)$ such that for every $k = 1 \ldots m - 1$ there is an edge from the segment S_{p_k} to the segment $S_{p_{k+1}}$.

A set of segments of the sentence W is said to be a **group of segments** G iff for each pair of segments $S_i, S_j \in G$ holds that there is a path from S_i to S_j (symmetrically, also a path from the S_j to the S_i must exist).

We use the following algorithm for obtaining the segmentation chart for individual sentences of PDT.

Determination of segments: The first step for obtaining the segmentation chart consists in the determination of boundaries; based on the boundaries, individual segments are identified.

Groups of segments: Groups of segments are identified.

Zero Layer: The segments which are connected by some path (either direct, i.e. edge, or via nodes representing elementary boundaries only) with the root node of the dependency tree T are identified; these segments as well as all segments belonging to the same groups are assigned layer 0.

Coordination and apposition: If there is a segment S_i with already assigned layer k and its adjacent segment S_j has unknown layer and, moreover, the boundary between these two segments consists of some coordinating expression or expression introducing an apposition (e.g., the node representing the elementary boundary has an analytical function `Coord` or `Apos`), then the segment S_j gets the same layer as the segment S_i has.

Deeper embedded segments: All segments with unknown layer connected by some path (either direct, i.e. edge, or via nodes representing elementary boundaries only) with segments of the layer k are assigned the layer $k + 1$; the same holds for all segments belonging to the same group of segments.

Coordination and apposition: Again all segments adjacent to the segments with already known layers are checked (see above).

This process is repeated until all segments get their layer.

The proposed algorithm assigns exactly one segmentation chart (not necessarily the correct one) to any input sentence represented by the analytical tree. Let us demonstrate it on a sentence (2); the analytical tree is in Fig. 2.

(2) _ *Po rozhovorech s majiteli našich soukromých firem a nakonec i představiteli firem zahraničních mám dojem , že v této republice nejsou schopní lidé .*
[_ After the discussions with the owners of our private companies and after all even with the representatives of foreign companies I have an idea , that there aren't clever people in this republic .]

Sentence (2) consists of four segments (the boundaries are underlined in the sentence whereas they are separated by vertical lines in Fig. 2). The first and the third segment form a group (there is an edge from the node *po* [after] to the node *mám* [(I) have] and at the same time a path leads from the node *představiteli* [representatives] to the node *s* [with], see the arrows). These two segments obtain the zero layer as there is the edge from the node *mám* [(I) have] to the root of the tree. The second segment also gets the zero layer as its boundary with the first segment is the coordination conjunction *a* [and]. The fourth segment obtains layer 1 since there is an edge leading from this segment to the third segment with already known zero layer. Therefore, the segmentation chart assigned to the sentence is (0001) (the correct segmentation chart in this case).

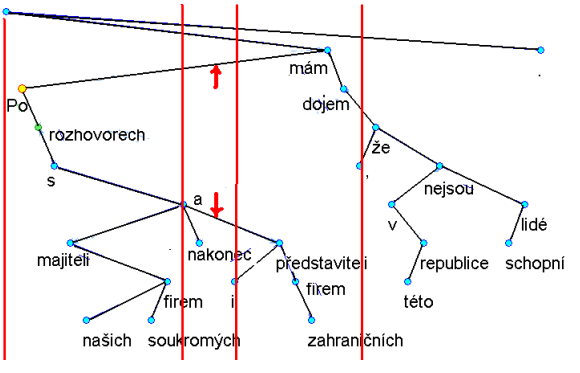


Fig. 2. Analytical tree of the sentence (2) with highlighted segments

3.2 How to Obtain Segments from Plain Text?

The basic set of (heuristic) segmentation rules for plain text was published in [7]. We have specified these rules more precisely and implemented them. That allows

us to compare the results of these rules with the results of segmentation based on the analytical trees from PDT.

When processing an input sentence, we start at its beginning; we move right, identify individual boundaries and segments and determine their appropriate layers of embedding.

The following rules define the layer of embedding that is assigned to the first segment. They also determine how this layer may change when crossing the elementary boundaries. Let us point out that the rules do not always give a single unambiguous answer (e.g., comma may be considered as coordinating expression – then the layer should be preserved – or as the end of embedding segment – then it should raise the layer). Thus each segment is not assigned a single number but an **interval of** possible **layers**.

The adjacent segments may be separated by compound boundaries, i.e. by sequences of elementary boundaries. In such a case, the rules are applied to individual elementary boundaries. The segment is assigned the layer which is obtained after processing the last elementary boundary preceding this segment.

The following list introduces the rules for elementary boundaries.[2]

Beginning of the sentence: If subordination flag (SF, see Section 2) is not assigned to the first segment, then this segment gets the basic zero layer. Otherwise, it gets layer 1.

Comma: If SF is not assigned to the subsequent segment, then the lower limit of the interval of layers does not change, the upper limit is set to 0 (i.e., the case of end of any number of embedded clauses). Otherwise, the layer of embedding is increased by 1 (i.e., the beginning of embedded clause or its part).

Opening bracket (of any kind): If SF is not assigned to the subsequent segment, then the layer (or interval of possible layers) of embedding is increased by 1 (i.e., the beginning of parenthesis). Otherwise, the layer is increased by 2 (i.e., parenthesis with a deeply embedded unit).

Closing bracket (of any kind): If it is preceded by the opening bracket of the same kind, then the layer of embedding is set to the same value(s) as the segment preceding the opening bracket has. Otherwise, the layer does not change (this condition handles the cases of the list *a)... b)...*).

Coordinating conjunction: The layer remains unchanged.

Colon: If SF is not assigned to the subsequent segment, then the upper limit remains unchanged (i.e., coordination or apposition); the lower limit is increased by 1 (i.e., the beginning of (a part of) embedded clause or beginning of direct speech (together with a quotation mark)). Otherwise, the upper limit is increased by 1 and the lower limit is increased by 2 (i.e., deeper embedded (part of) clause).

Question mark, exclamation mark: The lower limit is decreased by 1, the upper limit is set to 0 (i.e., the end of any number of embedded clauses).

[2] Let us repeat that we assume the input text being already divided into sentences.

Semicolon: The lower limit of the interval of layers remains unchanged, the upper limit is set to 0.
Vertical bar, dash, quotation marks: The layer remains unchanged.[3]

These rules define a set of segmentation charts for each morphologically analyzed input sentence.

4 Evaluation and Analysis of the Results

4.1 Evaluation Data and Possible Evaluation Measures

In the previous sections, we have described the basic experiments with the automatic identification of segmentation charts from plain texts and from trees from PDT. For further development and improvement of these rules, we had to create a test set of sentences with correctly identified segmentation charts.

We chose a set of suitable sentences from development data of PDT 2.0 (the 'dtest' data, 5228 sentences) – we focused only on such sentences that contain at least five segments (707 sentences). Then we manually determined a segmentation chart for every tenth sentence from the set. Thus we received 71 relatively structurally complex sentences with attached segmentation charts.

Let us emphasize that the selection of such complex sentences (in average 6.49 segments per sentence) made the measured results significantly worse in comparison with random sample of sentences (the average number of segments per sentence in the full dtest data is 2.72).

Note also that many of the testing sentences are ambiguous, i.e. they have more (potential) syntactic trees. However, sentences in PDT are disambiguated, only one of all possible structures is stored there. When identifying the appropriate segmentation chart we consider only the structure captured in PDT. Every sentence has been assigned a single chart (e.g., sentence (1) got the only segmentation chart (01221)).

There are several possibilities how to evaluate the proposed rules. The simplest measure consists in counting the cases of correct assignment of layers to individual segments. We call this basic measure ρ.

When looking at the results of experiments, we have found out that in many cases the wrong assignment of a layer for one segment has resulted in incorrectly identified layers of other segments. However, the relationship among individual segments may be recognized correctly. For example, the sentence (3) has a correct segmentation chart (2233110). The algorithm for PDT yields the chart (1122000); although almost all relations between segments are identified correctly, there is only one correctly assigned layer and $\rho = 1/7$.

(3) *„ Když to odečtete od výplaty spolu se ztrátou při výměně slovenských korun za české a za pojištění , které se musí platit tam i u nás , nezbude manželovi z výplaty ani polovina ," zlobí se paní Krajčová .*

[3] Quotation marks are used in Czech either for direct speech – then they are accompanied with other boundary as comma or colon (which ensures the lower layer) or they are use for emphasizing, where the layer should stay unchanged.

[,, When you deduct this from your earnings together with the losses when exchanging Slovak crowns for Czech crowns and for insurance , which must be paid there as well as here , less then half of the sum will remain from my husband's salary ," says Mrs. Krajčová with angry .]

This drawback of the basic measure may be eliminated if we allow 'shifting' of the whole resulting segmentation chart. E.g., if we shift the vector for the sentence (3) by +1 we get (2233111) – the layers of six segments (out of seven) are identified correctly. The measure with optimal shifting will be called σ (thus $\sigma = 6/7$ for the sentence (3)).

As we are primarily interested in the relationship among segments we consider also the measure evaluating the correctness of the proposed relationship of two adjacent segments. E.g., charts (101) and (211) have the same relationship between the first and second segment (difference -1), but different relationship between the second and third segment). We call this measure δ.

4.2 Evaluation of Rules for Syntactic Trees

The proposed set of segmentation rules from PDT identifies exactly one segmentation chart for each input sentence. When evaluating these rules, we adopt only *accuracy* measure (standard *recall* and *precision* measures are equal). The results are summarized in Table 1.

Table 1. The evaluation of the proposed set of rules for segmentation charts from the trees from PDT

accuracy:	basic measure		measure with 'shifting'	
# of segments	# correct	ρ	# correct	σ
461	264	0,57	335	0,73

When evaluating the proposed relationship of two adjacent segments, the rules are reaching $\delta = 0.70$ (274 of 390 relations among segments have been proposed correctly).

Let us mention three main problems that decrease the success of the proposed rules for determining segmentation charts from the analytical trees.

1. The sentence member forming a separate segment is assigned a higher layer (by 1) than the segments with its governing member. E.g., the sentence _ *Včera , kdy tak pršelo , přišli .* [_ Yesterday , when it rained so much , they came.] with the correct segmentation chart (010) gets incorrect segmentation chart (120).
2. We postponed special (but relatively frequent) Czech construction with two subordinating expressions (underlined) appearing in one segment just one after another, as e.g. *Nevěděl, že když jsem se probral, zavolal jsem policii.* [He didn't know that when I woke up, I called the police.]
3. Coordination (and apposition) are another widespread phenomena which deserve a special treatment, especially those of more than two members.

4.3 Evaluation of Rules for Plain Text

The evaluation of rules for assigning segmentation chart to plain text consists in testing whether the resulting interval for individual segments contains the correct layer of embedding of this segment (thus we measure only *recall*), see Table 2.

Table 2. The evaluation of the proposed set of rules for charts from plain text

recall:	basic measure		measure with 'shifting'	
# of segments	# correct	ρ	# correct	σ
461	302	0,66	354	0,77

The average number of segmentation charts per sentence from our testing data is 2.17 whereas the average number of ambiguity for the entire dtest data is 1.32.

Let us mention here at least two phenomena that the proposed set of rules does not solve adequately. These phenomena have to be a subject of more precise specifications (which would require detailed linguistic examination).

1. We have not specified the segmentation rules for direct and semidirect speech. E.g., the layers of the first four segments of sentence (3) are not deep enough in all assigned segmentation charts (these segments get (1122) instead of the correct (2233) segmentation vector).
2. The case of several coordinated clauses with repeated subordinating expressions is not treated correctly yet. E.g., *Jak účelně větrat, jak nepřetápět, jak spotřebu měnit a podle toho účtovat.* [How to ventilate effectively, how to not overheat, how to change the consumption and pay according to it.] This sentence obtains the wrong chart (0122) instead of the correct one (0000).)

5 Conclusions

Segments are easily automatically detectable and linguistically motivated units that form (complex) sentences. Their mutual relationship – especially relations of coordination and apposition, relation of subordination as well as parenthesis – is captured in a form of a segmentation chart. The segmentation chart describes the overall structure of a sentence prior to its complete syntactic analysis.

We focused on the description of a framework that allows us to formulate and refine linguistically motivated rules for automatic detection of segmentation charts for given sentences. We have also introduced appropriate measures for evaluating segmentation analysis.

At this stage, two sets of rules were implemented, rules operating on analytical trees from the Prague Dependency Treebank and rules operating on plain text enriched with morphological analysis. We have compared the results reached with those rules using the manually annotated sample of sentences from PDT.

The experiments brought clear specification of segmentation charts and exact rules for manual annotation. The results show that for further research it is

necessary to work with a large set of reliably annotated data. It turns out that these data cannot be obtained without extensive (semi)manual annotation of a large set of sentences. Such data would also allow us to adopt machine learning techniques for automatic identification of segmentation charts.

References

1. Holan, T.: O složitosti Vesmíru. In: Obdržálek, D., Štanclová, J., Plátek, M. (eds.) Malý informatický seminář MIS 2007, pp. 44–47. MatFyz Press, Praha (2007)
2. Abney, S.: Parsing By Chunks. In: Berwick, R., Abney, S., Tenny, C. (eds.) Principle-Based Parsing, pp. 257–278. Kluwer Academic Publishers, Dordrecht (1991)
3. Abney, S.: Partial Parsing via Finite-State Cascades. Journal of Natural Language Engineering 2, 337–344 (1995)
4. Brants, T.: Cascaded Markov Models. In: Proceedings of EACL 1999, pp. 118–125. University of Bergen (1999)
5. Ciravegna, F., Lavelli, A.: Full Text Parsing using Cascades of Rules: An Information Extraction Procedure. In: Proceedings of EACL 1999, pp. 102–109. University of Bergen (1999)
6. Kuboň, V.: Problems of Robust Parsing of Czech. Ph.D. Thesis, MFF UK, Prague (2001)
7. Kuboň, V., Lopatková, M., Plátek, M., Pognan, P.: A Linguistically-Based Segmentation of Complex Sentences. In: Wilson, D.C., Sutcliffe, G.C.J. (eds.) Proceedings of FLAIRS Conference, pp. 368–374. AAAI Press, Menlo Park (2007)
8. Pardubská, D., Plátek, M.: On Parallel Communicating Grammar Systems and Correctness Preserving Restarting Automata. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 1–18. Springer, Heidelberg (2009)
9. Jones, B.E.M.: Exploiting the Role of Punctuation in Parsing Natural Text. In: Proceedings of the COLING 1994, Kyoto, pp. 421–425 (1994)
10. Ohno, T., Matsubara, S., Kashioka, H., Maruyama, T., Inagaki, Y.: Dependency Parsing of Japanese Spoken Monologue Based on Clause Boundaries. In: Proceedings of COLING and ACL, pp. 169–176 (2006)
11. Hajič, J.: Disambiguation of Rich Inflection (Computational Morphology of Czech), UK, Nakladatelství Karolinum, Praha (2004)
12. Spoustová, D., Hajič, J., Votrubec, J., Krbec, P., Květoň, P.: The Best of Two Worlds: Cooperation of Statistical and Rule-Based Taggers for Czech. In: Proceedings of Balto-Slavonic NLP Workshop, pp. 67–74. ACL, Prague (2007)
13. Hajič, J., Hajičová, E., Panevová, J., Sgall, P., Pajas, P., Štěpánek, J., Havelka, J., Mikulová, M.: Prague Dependency Treebank 2.0. LDC, Philadelphia (2006)

A Note on the Generative Power of Some Simple Variants of Context-Free Grammars Regulated by Context Conditions

Tomáš Masopust

Brno University of Technology, Faculty of Information Technology
Božetěchova 2, Brno 61266, Czech Republic
masopust@fit.vutbr.cz

Abstract. This paper answers three open questions concerning the generative power of some simple variants of context-free grammars regulated by context conditions. Specifically, it discusses the generative power of so-called context-free semi-conditional grammars (which are random context grammars where permitting and forbidding sets are replaced with permitting and forbidding strings) where permitting and forbidding strings of each production are of length no more than one, and of simple semi-conditional grammars where, in addition, no production has attached both a permitting and a forbidding string. Finally, this paper also presents some normal form results, an overview of known results, and unsolved problems.

1 Introduction

It is well-known that context-free languages play an important role in the theory and practice of formal languages in computer science. However, there is a lot of interesting and simple languages that are not context-free. According to the Chomsky hierarchy, such languages are treated as being context-sensitive. On the other hand, in the theory of regulated rewriting, many of these languages can be generated by regulated grammars using the benefits of applying only context-free productions.

The present paper discusses two simple variants of context-free grammars regulated by context conditions; both variants are special cases of so called *context-free random context grammars* (defined and studied by van der Walt in [1]), which are context-free grammars where two sets of symbols (*conditions*) are attached to each production—a permitting and a forbidding set. In addition, the grammars studied in this paper require that both the permitting and the forbidding sets contain no more than one symbol. A production of such a grammar is applicable to a sentential form provided that the symbol from the attached permitting set (the *permitting condition*) occurs in the sentential form while, simultaneously, the symbol from the attached forbidding set (the *forbidding condition*) does not. These grammars were defined by Păun [2] in 1985 and called *semi-conditional grammars of degree* $(1, 1)$. In general, semi-conditional

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 554–565, 2009.

grammars are defined to be of any degree (i, j), for $i, j \geq 0$, where the degree (i, j) means that all permitting and forbidding conditions (that are strings, in general, not only symbols) are of length no more than i and j, respectively. In addition, semi-conditional grammars where each production has no more than one condition in the union of its permitting and forbidding sets are referred to as *simple semi-conditional grammars* (see [3]).

Since their introduction, it has been an open problem whether every semi-conditional grammar can be converted to an equivalent simple semi-conditional grammar (of the same degree); cf. [4, page 90]. This paper answers this question so that it demonstrates how to convert any semi-conditional grammar to an equivalent simple semi-conditional grammar of the same degree. In fact, this demonstration is given for both semi-conditional grammars with and without erasing productions. In addition, this paper also shows that semi-conditional grammars of degree $(1, 1)$ characterize the family of recursively enumerable languages, which is a question left unsolved in [2] and still formulated as open in [4]. As an immediate consequence of these two results, it follows that simple semi-conditional grammars of degree $(1, 1)$ characterize the family of recursively enumerable languages, too. Furthermore, this paper presents three normal form results. Specifically, it proves that (i) for any (simple) semi-conditional grammar, there is an equivalent simple semi-conditional grammar of the same degree with the property that its (core context-free) productions can be decomposed into two disjoint sets in such a way that in one set, all productions have attached only permitting conditions, while in the other set, all productions have attached only forbidding conditions; it means that if $u_1, u_2, \ldots, u_k$ are all conditions attached to a production $A \rightarrow \alpha$, then all of them are either permitting or forbidding; (ii) for any context-sensitive language, there is a simple semi-conditional grammar of degree (i, j), $i, j \in \{1, 2\}$, $i \neq j$, without erasing productions and without conditions containing terminal symbols that satisfies the property from (i); and (iii) for any recursively enumerable language, there is a simple semi-conditional grammar of degree $(1, 1)$ without conditions containing terminal symbols that satisfies the property from (i).

In its conclusion, this paper gives an overview of known results concerning the generative power of discussed grammars, including the results concerning the descriptional complexity, presents a simple semi-conditional grammar of degree $(1, 1)$ without erasing productions that generates a nontrivial context-sensitive language, and discusses open problems.

2 Preliminaries and Definitions

In this paper, we assume that the reader is familiar with formal language theory and with the theory of regulated rewriting (see [5,6]). For an alphabet (finite nonempty set) V, V^* represents the free monoid generated by V. The unit of V^* is denoted by ε. Set $V^+ = V^* - \{\varepsilon\}$. For $w \in V^*$, $|w|$ denotes the length of w, and $sub(w) = \{u : u \text{ is a substring of } w\}$.

Let **RE**, **REC**, **CS**, **CF** denote the families of recursively enumerable, recursive, context-sensitive, and context-free languages, respectively. In addition, let $\mathbf{RC_{ac}}$, **RC**, and **fRC** denote the families of languages generated by random context grammars with appearance checking, random context grammars where each forbidding set is empty (*permitting grammars*), and random context grammars where each permitting set is empty (*forbidding grammars*), respectively. Moreover, superscript ε is added if erasing productions are allowed.

A *semi-conditional grammar* (see [2]) is a quadruple $G = (N, T, P, S)$, where N and T are the alphabets of nonterminals and terminals, respectively, such that $N \cap T = \emptyset$, $V = N \cup T$, $S \in N$ is the start symbol, and P is a finite set of productions of the form $(X \to \alpha, u, v)$ such that $X \to \alpha$ is a context-free production and $u, v \in V^+ \cup \{0\}$, where $0 \notin V$ is a special symbol. If for each production $(X \to \alpha, u, v) \in P$, $u \neq 0$ implies that $|u| \leq i$ and $v \neq 0$ implies that $|v| \leq j$, then G is said to be of *degree* (i, j). G is said to be *simple* if for each production $(X \to \alpha, u, v) \in P$ we have $0 \in \{u, v\}$.

For $x_1, x_2 \in V^*$, $x_1 X x_2 \Rightarrow x_1 \alpha x_2$ provided that

1. $(X \to \alpha, u, v) \in P$,
2. $u \neq 0$ implies that $u \in sub(x_1 X x_2)$, and
3. $v \neq 0$ implies that $v \notin sub(x_1 X x_2)$.

As usual, $\Rightarrow$ is extended to $\Rightarrow^i$, for $i \geq 0$, $\Rightarrow^+$, and $\Rightarrow^*$. The language of G is defined as $L(G) = \{w \in T^* : S \Rightarrow^* w\}$. The family of languages generated by semi-conditional grammars of degree (i, j) is denoted by $\mathbf{SC}^{\varepsilon}(\mathbf{i}, \mathbf{j})$, or $\mathbf{SC}(\mathbf{i}, \mathbf{j})$ if erasing productions are not allowed. Analogously, the family of languages generated by simple semi-conditional grammars of degree (i, j) is denoted by $\mathbf{SSC}^{\varepsilon}(\mathbf{i}, \mathbf{j})$, or $\mathbf{SSC}(\mathbf{i}, \mathbf{j})$ if erasing productions are not allowed.

3 Main Results

As stated above, this paper concentrates its attention on language families $\mathbf{SSC}^{\varepsilon}(\mathbf{1}, \mathbf{1})$ and $\mathbf{SSC}(\mathbf{1}, \mathbf{1})$. First, it answers three questions formulated as open in [4] (see also [2]) concerning the relations among the families $\mathbf{SSC}^{\varepsilon}(\mathbf{1}, \mathbf{1})$, $\mathbf{SC}^{\varepsilon}(\mathbf{1}, \mathbf{1})$, and **RE** (Theorems 3 and 2 and Corollary 1), and between the families $\mathbf{SSC}(\mathbf{1}, \mathbf{1})$ and $\mathbf{SC}(\mathbf{1}, \mathbf{1})$ (Theorem 1). Then, it gives an overview of known results, demonstrates the generative power of non-erasing simple semi-conditional grammars, and discusses open problems.

Theorem 1. *For any* $i, j \geq 1$, $\mathbf{SSC}(\mathbf{i}, \mathbf{j}) = \mathbf{SC}(\mathbf{i}, \mathbf{j})$.

Proof. Let $L \in \mathbf{SC}(\mathbf{i}, \mathbf{j})$, for some $i, j \geq 1$. Then, there is a semi-conditional grammar $G = (N, T, P, S)$ of degree (i, j) without erasing productions such that $L(G) = L$. Construct a simple semi-conditional grammar $G' = (N', T, P', S_1)$, where S_1 is a new start symbol, $N' = N \cup \{S_1\} \cup \{[A] : A \in V\} \cup \{A', A'' : A \in N\} \cup \{[pA], [p_0 A], [p_1 A], [p_2 A], [p_3 A], [p'A], [p''A] : p = (A \to \alpha, u, v) \in P\}$, $P' = \{(S_1 \to [S], 0, 0)\} \cup \{([a] \to a, 0, 0) : a \in T\}$, and for each $p = (A \to \alpha, u, v) \in P$, the following productions are added to P':

Case 1: For $B \in V$

1. $([B] \to [pB], u, 0)$,
2. $([B] \to [pB], [B]u', 0)$, for $u = Bu'$,
3. $([pB] \to [p_0B], 0, v)$,
4. $([p_0B] \to [p_1B], 0, \gamma)$, where $\gamma = \begin{cases} [p_0B]v' & \text{if } v = Bv' \\ 0 & \text{otherwise} \end{cases}$
5. $(A \to A', [p_1B], 0)$,
6. $(A' \to A'', 0, A'')$,
7. $([p_1B] \to [p_2B], A'', 0)$,
8. $([p_2B] \to [p_3B], 0, A')$,
9. $(A'' \to \alpha, [p_3B], 0)$,
10. $([p_3B] \to [B], 0, A'')$.

Case 2: The first nonterminal of a sentential form is replaced.

11. $([A] \to [p'A], u, 0)$,
12. $([A] \to [p'A], [A]u', 0)$, for $u = Au'$,
13. $([p'A] \to [p''A], 0, v)$
14. $([p''A] \to [B]\beta, 0, \gamma)$, where $\alpha = B\beta$, $B \in V$, $\gamma = \begin{cases} [p''A]v' & \text{if } v = Av' \\ 0 & \text{otherwise.} \end{cases}$

To prove that $L(G) \subseteq L(G')$, consider a derivation of G. Such a derivation is of the form $S \Rightarrow^* Bw_1Aw_2 \Rightarrow Bw_1\alpha w_2$, where the last derivation step is made by a production $p = (A \to \alpha, u, v) \in P$, and $B \in V$. Then, G' derives as follows (numbers in square brackets denote (classes of) productions applied in given derivation steps):

$$\begin{aligned} S_1 &\Rightarrow [S] \\ &\Rightarrow^* [B]w_1Aw_2 \\ &\Rightarrow [pB]w_1Aw_2 \text{ [1 or 2]} \\ &\Rightarrow [p_0B]w_1Aw_2 \text{ [3]} \\ &\Rightarrow [p_1B]w_1Aw_2 \text{ [4]} \\ &\Rightarrow [p_1B]w_1A'w_2 \text{ [5]} \\ &\Rightarrow [p_1B]w_1A''w_2 \text{ [6]} \\ &\Rightarrow [p_2B]w_1A''w_2 \text{ [7]} \\ &\Rightarrow [p_3B]w_1A''w_2 \text{ [8]} \\ &\Rightarrow [p_3B]w_1\alpha w_2 \text{ [9]} \\ &\Rightarrow [B]w_1\alpha w_2 \text{ [10].} \end{aligned}$$

If the derivation is of the form $S \Rightarrow^* Aw \Rightarrow \alpha w = B\beta w$ in G, $B \in V$, i.e., the first nonterminal of the sentential form is replaced, then G' derives

$$\begin{aligned} S_1 &\Rightarrow [S] \\ &\Rightarrow^* [A]w \\ &\Rightarrow [p'A]w \text{ [11 or 12]} \\ &\Rightarrow [p''A]w \text{ [13]} \\ &\Rightarrow [B]\beta w \text{ [14].} \end{aligned}$$

The proof now proceeds by induction.

On the other hand, to prove that $L(G') \subseteq L(G)$, consider a sentential form $[A]w$ and assume that a production constructed in 11 or 12 is applied. Then, the only derivation is

$$\begin{aligned}[A]w &\Rightarrow [p'A]w \text{ [11 or 12]}\\ &\Rightarrow [p''A]w \text{ [13]}\\ &\Rightarrow [B]\beta w \text{ [14]},\end{aligned}$$

where $p = (A \to B\beta, u, v) \in P$, $B \in V$. From this, by a production constructed in 11 or 12, it follows that u (different from 0) is a substring of Aw, and, by productions constructed in 13 and 14, v is not a substring of Aw. Thus, $Aw \Rightarrow B\beta w$ by $(A \to B\beta, u, v) \in P$ in G.

Now, assume that a production constructed in 1 or 2 is applied to a sentential form $[B]w_1Aw_2$. Then, the only derivation is of the form

$$\begin{aligned}[B]w_1Aw_2 &\Rightarrow [pB]w_1Aw_2 \text{ [1 or 2]}\\ &\Rightarrow [p_0B]w_1Aw_2 \text{ [3]}\\ &\Rightarrow [p_1B]w_1Aw_2 \text{ [4]}\\ &\Rightarrow [p_1B]w_1A'w_2 \text{ [5]}\\ &\Rightarrow [p_1B]w_1A''w_2 \text{ [6]}\\ &\Rightarrow [p_2B]w_1A''w_2 \text{ [7]}\\ &\Rightarrow [p_3B]w_1A''w_2 \text{ [8]}\\ &\Rightarrow [p_3B]w_1\alpha w_2 \text{ [9]}\\ &\Rightarrow [B]w_1\alpha w_2 \text{ [10]},\end{aligned}$$

where $p = (A \to \alpha, u, v) \in P$. Surely, by a production constructed in 1 or 2, it follows that u (different from 0) is a substring of Bw_1Aw_2, and, by productions constructed in 3 and 4, it follows that v is not a substring of Bw_1Aw_2. Moreover, by a production constructed in 6, only one A' can be replaced with A'', and a production constructed in 8 can be applied only if there is no A'. Therefore, only one A is replaced with A' by a production constructed in 5 (the one later replaced with A''). Thus, only one A is replaced with α, i.e., $Bw_1Aw_2 \Rightarrow Bw_1\alpha w_2$ by $(A \to \alpha, u, v) \in P$ in G.

We have proved that $\mathbf{SC}(\mathbf{i}, \mathbf{j}) \subseteq \mathbf{SSC}(\mathbf{i}, \mathbf{j})$. The other inclusion follows immediately from the definition. Hence, the theorem holds. □

Considering Case 2 of the previous construction, it is not hard to see that this construction is not valid for grammars with erasing productions; by an erasing production, the special first nonterminal of the form $[A]$ would be eliminated and the derivation would be blocked. However, a simple modification of the previous construction proves the following theorem.

Theorem 2. *For any $i, j \geq 1$, $\mathbf{SSC}^{\varepsilon}(\mathbf{i}, \mathbf{j}) = \mathbf{SC}^{\varepsilon}(\mathbf{i}, \mathbf{j})$.*

Proof. Let $L \in \mathbf{SC}^{\varepsilon}(\mathbf{i}, \mathbf{j})$, for some $i, j \geq 1$. Then, there is a semi-conditional grammar $G = (N, T, P, S)$ of degree (i, j) such that $L(G) = L$. Construct a

simple semi-conditional grammar $G' = (N', T, P', S_1)$, where $N' = N \cup \{S_1, X\} \cup \{A', A'' : A \in N\} \cup \{[p], [p_0], [p_1], [p_2] : p = (A \to \alpha, u, v) \in P\}$, S_1 and X are new symbols not in N, $P' = \{(S_1 \to XS, 0, 0), (X \to \varepsilon, 0, 0)\}$, and for each $p = (A \to \alpha, u, v) \in P$, the following productions are added to P':

1. $(X \to [p], u, 0)$,
2. $([p] \to [p_0], 0, v)$,
3. $(A \to A', [p_0], 0)$,
4. $(A' \to A'', 0, A'')$,
5. $([p_0] \to [p_1], A'', 0)$,
6. $([p_1] \to [p_2], 0, A')$,
7. $(A'' \to \alpha, [p_2], 0)$,
8. $([p_2] \to X, 0, A'')$.

The rest of the proof is analogous to the proof of Theorem 1 and is left to the reader. □

The following theorem answers the question left unsolved in [2] of what is the relation between the families $\mathbf{SC}^{\varepsilon}(\mathbf{1}, \mathbf{1})$ and $\mathbf{RE}$?

Theorem 3. $\mathbf{SC}^{\varepsilon}(\mathbf{1}, \mathbf{1}) = \mathbf{RE}$.

Proof. The proof is a straightforward consequence of the proof given in [7, Section 3.2], where for each recursively enumerable language L, a random context grammar G is given such that $L(G) = L$ and each of permitting and forbidding sets contains no more than one symbol. The main idea of the proof is based on the fact that any recursively enumerable language can be generated by an unordered scattered context grammar. Then, such an unordered scattered context grammar in a special normal form generating L is consider and transformed into a random context grammar. For more details, the reader is referred to Lemma 6 in [7].

Thus, we obtain the required semi-conditional grammar by replacing one-element sets with their elements and empty sets with 0. □

As an immediate consequence, we have the following result.

Corollary 1. $\mathbf{SSC}^{\varepsilon}(\mathbf{1}, \mathbf{1}) = \mathbf{RE}$.

As no context-free production in the constructions of Theorems 1 and 2 has attached both a permitting and a forbidding condition, the following corollary holds. It says that the core context-free productions can be decomposed into two disjoint sets of productions—the productions with only permitting conditions (*permitting productions*) and the productions with only forbidding conditions (*forbidding productions*). Note that in case of erasing productions, such systems have been studied (using a different technique) in [8] (cf. Corollary 4). Thus, the following consequences of the previous results of this paper complement [8] in case of non-erasing productions, and, in addition, use much simpler proofs than used in [8].

Corollary 2. *For any semi-conditional grammar G' of degree (i,j) without erasing productions, $i,j \geq 1$, there is an equivalent simple semi-conditional grammar $G = (N,T,P,S)$ of the same degree without erasing productions such that $(A \to \alpha,u,0) \in P$ and $(A \to \alpha,0,v) \in P$ imply that $0 \in \{u,v\}$.*

In addition, by a standard technique, it can be proved that conditions u and v contain only nonterminals, i.e., $u,v \in N^+ \cup \{0\}$, so that each production $(A \to \alpha,u,v)$ is replaced with $(A \to h(\alpha),h(u),h(v))$, where h is a homomorphism defined as $h(A) = A$, for $A \in N \cup \{0\}$, and $h(a) = a'$, for $a \in T$, where a' is a new nonterminal, and $([a] \to a,0,0)$ is replaced with $([a] \to t_a,0,0)$ and $(t_a \to a,0,0)$, where t_a is a new nonterminal for all $a \in T$. Finally, $(a' \to a,t_b,0)$, for $b \in T$, are added for all $a \in T$. In case of erasing productions, Theorem 2, $(X \to \varepsilon,0,0)$ is replaced with $(X \to Y,0,0)$ and $(Y \to \varepsilon,0,0)$, where Y is a new nonterminal, and $(a' \to a,Y,0)$ are added for all $a \in T$.

By 4 of Theorem 5, we have the following normal form theorem.

Corollary 3. *For any context-sensitive language L, there is a simple semi-conditional grammar $G = (N,T,P,S)$ of degree (i,j), for $i,j \in \{1,2\}$, $i \neq j$, without erasing productions such that $L(G) = L$ and*

1. *$(A \to \alpha,u,v) \in P$ implies that $u,v \in N^+ \cup \{0\}$, and*
2. *$(A \to \alpha,u,0) \in P$ and $(A \to \alpha,0,v) \in P$ imply that $0 \in \{u,v\}$.*

In addition, by Theorem 3, Corollary 3 can be modified to obtain the following normal form theorem.

Corollary 4. *For any recursively enumerable language L, there is a simple semi-conditional grammar $G = (N,T,P,S)$ such that $L(G) = L$ and*

1. *$(A \to \alpha,u,v) \in P$ implies that $u,v \in N \cup \{0\}$ (i.e., G is of degree $(1,1)$), and*
2. *$(A \to \alpha,u,0) \in P$ and $(A \to \alpha,0,v) \in P$ imply that $0 \in \{u,v\}$.*

4 Overview of Results and Open Problems

This section presents an overview of results concerning simple semi-conditional grammars known so far. In addition, it also presents an overview of open problems.

Theorem 4. *The following holds for grammars with erasing productions.*

1. $\mathbf{SSC}^{\varepsilon}(\mathbf{0},\mathbf{0}) = \mathbf{CF}$.
2. $\mathbf{CF} \subset \mathbf{SSC}^{\varepsilon}(\mathbf{0},\mathbf{1}) \subseteq \mathbf{fRC}^{\varepsilon} \subset \mathbf{REC}$.
3. $\mathbf{CF} \subset \mathbf{SSC}^{\varepsilon}(\mathbf{1},\mathbf{0}) \subseteq \mathbf{RC}^{\varepsilon} \subset \mathbf{REC}$.
4. $\mathbf{SSC}^{\varepsilon}(\mathbf{1},\mathbf{1}) = \mathbf{SC}^{\varepsilon}(\mathbf{1},\mathbf{1}) = \mathbf{RE}$.

Proof. The inclusions in 2 and 3 are straightforward; the proofs of the proper inclusions can be found, e.g., in [4] and [9], respectively. □

Theorem 5. *The following holds for grammars without erasing productions.*

1. $\mathbf{SSC}(\mathbf{0},\mathbf{0}) = \mathbf{CF}$.
2. $\mathbf{CF} \subset \mathbf{SSC}(\mathbf{0},\mathbf{1}) \subseteq \mathbf{fRC} \subset \mathbf{CS}$.

3. $\mathbf{CF} \subset \mathbf{SSC}(\mathbf{1},\mathbf{0}) \subseteq \mathbf{RC} \subset \mathbf{CS}$.
4. $\mathbf{SSC}(\mathbf{2},\mathbf{1}) = \mathbf{SSC}(\mathbf{1},\mathbf{2}) = \mathbf{CS}$.
5. $\mathbf{SSC}(\mathbf{1},\mathbf{1}) = \mathbf{SC}(\mathbf{1},\mathbf{1}) \subseteq \mathbf{RC_{ac}} \subset \mathbf{CS}$.

Proof. The inclusions in 2 and 3 are straightforward; the proofs of the proper inclusions can be found, e.g., in [10] and [11], respectively. Results of 4 are proved in [4]. □

Note that the generative power of simple semi-conditional grammars of degree $(0,i)$ and $(i,0)$ (with or without erasing productions), for $i \geq 2$, are not known. However, if more than one forbidding string is allowed to be attached to a production (i.e., there are sets of forbidding conditions instead of only one condition), it is known that such grammars (referred to as *generalized forbidding grammars*) are computationally complete. In addition, it is sufficient to have no more than four forbidding conditions each of which is of length no more than two to characterize the family of recursively enumerable languages (see [12, Corollary 6]). On the other hand, however, the question of what is the generative power of *generalized permitting grammars* (defined in the same manner) is an open problem.

Note also that the precise relation between $\mathbf{SSC}(\mathbf{1},\mathbf{1})$ and $\mathbf{RC_{ac}}$ is not known. However, the following theorem illustrates the generative power of simple semi-conditional grammars so that it shows that they are powerful enough to generate nontrivial languages, such as prime numbers, i.e., the language $\mathbf{P} = \{a^p : p \text{ is a prime number}\}$.

Theorem 6. $\mathbf{P} \in \mathbf{SSC}(\mathbf{1},\mathbf{1})$.

Proof. Let $G = (N, \{a\}, P, S')$ be a simple semi-conditional grammars, where N follows from P that is constructed as follows:

1. $(S' \to a^2, 0, 0)$
2. $(S' \to S, 0, 0)$
3. $(S \to SCC, 0, 0)$
4. $(S \to AAX, 0, 0)$
5. $(A \to \bar{A}, X, 0)$
6. $(C \to \bar{C}, X, 0)$
7. $(\bar{A} \to A', 0, A')$
8. $(\bar{C} \to C', 0, C')$
9. $(X \to Z_1, A', 0)$
10. $(X \to Y_1, 0, A)$
11. $(X \to Q_1, 0, C)$
12. $(X \to F, 0, C)$
13. $(Z_1 \to Z_2, C', 0)$
14. $(Z_2 \to Z_3, 0, \bar{A})$
15. $(Z_3 \to Z, 0, \bar{C})$
16. $(A' \to B, Z, 0)$
17. $(C' \to D, Z, 0)$
18. $(Z \to Z_4, 0, A')$
19. $(Z_4 \to X, 0, C')$
20. $(Y_1 \to Y_2, 0, \bar{A})$
21. $(Y_2 \to Y, 0, A')$
22. $(B \to A, Y, 0)$
23. $(Y \to X, 0, B)$
24. $(Q_1 \to Q_2, 0, \bar{C})$
25. $(Q_2 \to Q_3, 0, C')$
26. $(Q_3 \to Q_4, 0, \bar{A})$
27. $(Q_4 \to Q_5, A', 0)$
28. $(Q_5 \to Q, A, 0)$
29. $(D \to \bar{D}, Q, 0)$
30. $(D \to C, Q, 0)$
31. $(B \to A, Q, 0)$
32. $(A' \to A, Q, 0)$
33. $(\bar{D} \to D_1, 0, D_1)$
34. $(Q \to Q_6, 0, D)$
35. $(Q_6 \to Q_7, 0, \bar{D})$
36. $(Q_7 \to Q_8, D_1, 0)$
37. $(D_1 \to A, Q_8, 0)$
38. $(Q_8 \to Q_9, 0, D_1)$
39. $(Q_9 \to Q_{10}, 0, B)$
40. $(Q_{10} \to X, 0, A')$
41. $(A \to a, F, 0)$
42. $(F \to a, 0, A)$

We prove that $L(G) = \mathbf{P}$. Clearly, a^2 is in $\mathbf{P}$. Thus, consider a terminal derivation beginning by an application of production 2. Then, only productions 3 and 4 are applicable, generating the sentential form $AAX(CC)^n$, for some $n \geq 0$, i.e., from now on, any sentential form is of length $2k+1$, for some $k \geq 1$.

Now, only productions 5, 6, 9, 10, 11, and 12 are applicable; of course, if productions 5 and 6 are applicable, then they are applied before any of productions 9, 10, 11, or 12.

A. Let production 9 be applied. Then, clearly, productions 7 and 8 had to be applied before productions 13 and 9, respectively. Then, by productions 13 to 19, the derivation continues according to these productions as follows:

$$A^q B^m C^r D^m X \Rightarrow^* A^{q-1} B^{m+1} C^{r-1} D^{m+1} X \,.$$

(Note that symbols of sentential forms are written in the alphabetic order, rather than in the actual possible order, because the order is not important.) Informally, this phase of the derivation replaces one A with B and one C with D, respectively.

B. Let production 10 be applied. Then, by productions 20 to 23, the derivation replaces each B with A, i.e.,

$$B^n C^r D^{tn} X \Rightarrow^* A^n C^r D^{tn} X \,.$$

Together with the previous phase, these two phases try to divide $2k+1$ by n, where $n \geq 2$.

C. Let production 11 be applied. Then, by productions 24 to 40, the derivation continues so that it verifies that there is no C (including C' and $\bar{C}$) and $\bar{A}$ and that there is A' and A in the current sentential form. Then, precisely one D_1 is generated from D, and each other D is replaced with C. Finally, it verifies that all symbols B and A' are replaced with A. Thus, we have

$$A^{n-m} B^m D^{tn+m} X \Rightarrow^* A^{n+1} C^{tn+m-1} X \,.$$

This phase verifies that n does not divide $2k+1$ so that it requires the reminder to be at least one (symbols A and A' are required to be in the sentential form; one of them is compared against the symbol X, the other is the nonzero remainder). More precisely, if there were $m \geq 2$ such that $2k+1 = mn$, then

$$A^n C^{n-1} D^{(m-2)n} X \Rightarrow^* A' B^{n-1} D^{n-1} D^{(m-2)n} Q_5$$

and the derivation would be blocked (see production 28).

D. Let production 12 be applied. Then, by productions 41 and 42, the derivation continues according to these productions as follows:

$$A^{2k} X \Rightarrow A^{2k} F \Rightarrow^* a^{2k+1} \,,$$

where $2k+1$ is a prime number because the derivation has verified that there is no $n \in \{2, 3, \ldots, 2k-1\}$ such that n divides $2k+1$.

Thus, the whole derivation is of the form

$$\begin{aligned} A^2C^{2(k-1)}X &\Rightarrow^* B^2C^{2(k-2)}D^2X \\ &\Rightarrow^* B^2D^{2(k-1)}X \\ &\Rightarrow^* A^2D^{2(k-1)}X \\ &\Rightarrow^* A^3C^{2(k-1)-1}X \\ &\Rightarrow^* A^4C^{2(k-2)}X \\ &\Rightarrow^* A^{2k}X \\ &\Rightarrow^* a^{2k+1}, \end{aligned}$$

where $2k+1$ is a prime number, i.e., $L(G)=\{a^p : p \text{ is a prime number}\}=\mathbf{P}$. □

5 Conclusion

From both theoretical and practical points of view, it is of a great interest to know the amount of resources needed to characterize any recursively enumerable language by (simple) semi-conditional grammars. This section summarizes results concerning the descriptional complexity of (simple) semi-conditional grammars known so far.

Let $(A \to \alpha, u, v)$ be a production of a semi-conditional grammar. If $u = v = 0$, then the production is said to be context-free; otherwise, it is said to be *conditional*.

Theorem 7 ([13]). *Every recursively enumerable language is generated by a simple semi-conditional grammar of degree* $(3,1)$ *with no more than eight conditional productions and eleven nonterminals.*

Theorem 8 ([14]). *Every recursively enumerable language is generated by a simple semi-conditional grammar of degree* $(2,1)$ *with no more than nine conditional productions and ten nonterminals.*

In case of semi-conditional grammars that are not simple, the previous result can be improved as follows.

Theorem 9 ([15]). *Every recursively enumerable language is generated by a semi-conditional grammar of degree* $(2,1)$ *with no more than seven conditional productions and eight nonterminals.*

Finally, note that Example 4.1.1 in [5] shows that there is no bound for the number of nonterminals for (simple) semi-conditional grammars of degree $(1,1)$ if terminal symbols are not allowed to appear in the conditions. More specifically, the example shows that any (simple) semi-conditional grammar of degree $(1,1)$ generating the language $T_n = \bigcup_{i=1}^{n}\{a_i^j : j \geq 1\}$, where conditions are nonterminal symbols, requires, in the nonerasing case, exactly $n+1$ nonterminal symbols, and, in the erasing case, at least $f(n)$ nonterminal symbols, for some unbounded mapping $f : \mathbb{N} \to \mathbb{N}$. In general, however, as terminal symbols are

allowed to appear in the conditions, and $G = (\{S, A\}, \{a_1, a_2, \ldots, a_n\}, \{(S \to a_iA, 0, 0), (S \to a_i, 0, 0), (A \to a_iA, a_i, 0), (A \to a_i, a_i, 0) : 1 \leq i \leq n\}, S)$ is a simple semi-conditional grammar of degree $(1, 1)$ that generates T_n, the question of whether analogous descriptional complexity results can be achieved for general (simple) semi-conditional grammars of degree $(1, 1)$ is open. Furthermore, other cases not presented above are open, too.

To summarize the main results, this paper has answered three questions formulated as open in [4, page 90] (see also [2], where semi-conditional grammars were introduced and studied). Specifically, it has proved that

1. every semi-conditional grammar (with or without erasing productions) can be converted to an equivalent simple semi-conditional grammar (with or without erasing productions, respectively) of the same degree,
2. semi-conditional grammars of degree $(1, 1)$ characterize the family of recursively enumerable languages,
3. and, as a consequence, simple semi-conditional grammars of degree $(1, 1)$ characterize the family of recursively enumerable languages.

In addition, it has also presented some normal form results and an overview of known results, demonstrated the generative power of simple semi-conditional grammars of degree $(1, 1)$ without erasing productions, and discussed open problems.

Acknowledgments

The author thanks the anonymous referees for their helpful suggestions.

This work was supported by the Czech Ministry of Education under the Research Plan No. MSM 0021630528 and, partially, by the Czech Grant Agency project No. 201/07/0005.

References

1. van der Walt, A.P.J.: Random context grammars. In: Proceedings of the Symposium on Formal Languages (1970)
2. Păun, G.: A variant of random context grammars: Semi-conditional grammars. Theoretical Computer Science 41, 1–17 (1985)
3. Gopalaratnam, A., Meduna, A.: On semi-conditional grammars with productions having either forbidding or permitting conditions. Acta Cybernetica 11(4), 307–324 (1994)
4. Meduna, A., Švec, M.: Grammars with Context Conditions and Their Applications. John Wiley & Sons, New York (2005)
5. Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. Springer, Berlin (1989)
6. Salomaa, A.: Formal languages. Academic Press, New York (1973)
7. Mayer, O.: Some restrictive devices for context-free grammars. Information and Control 20, 69–92 (1972)
8. Masoput, T., Meduna, A.: On context-free rewriting with a simple restriction and its computational completeness. In: RAIRO-ITA (to appear)

9. Bordihn, H., Fernau, H.: Accepting grammars and systems. Technical Report 9/94 (1994)
10. van der Walt, A.P.J., Ewert, S.: A shrinking lemma for random forbidding context languages. Theoretical Computer Science 237(1-2), 149–158 (2000)
11. Ewert, S., van der Walt, A.P.J.: A pumping lemma for random permitting context languages. Theoretical Computer Science 270(1–2), 959–967 (2002)
12. Masopust, T., Meduna, A.: Descriptional complexity of generalized forbidding grammars. In: Proceedings of 9th International Workshop on Descriptional Complexity of Formal Systems, High Tatras, Slovakia, pp. 170–177 (2007)
13. Vaszil, G.: On the descriptional complexity of some rewriting mechanisms regulated by context conditions. Theoretical Computer Science 330, 361–373 (2005)
14. Masopust, T., Meduna, A.: Descriptional complexity of grammars regulated by context conditions. In: Pre-proceedings of 1st International Conference on Language and Automata Theory and Application (LATA 2007), Tarragona, Spain, pp. 403–411 (2007)
15. Masopust, T., Meduna, A.: Descriptional complexity of semi-conditional grammars. Information Processing Letters 104(1), 29–31 (2007)

Efficiency of the Symmetry Bias in Grammar Acquisition

Ryuichi Matoba, Makoto Nakamura, and Satoshi Tojo

School of Information Science,
Japan Advanced Institute of Science and Technology,
1–1, Asahidai, Nomi, Ishikawa, 923–1292, Japan
{r-matoba,mnakamur,tojo}@jaist.ac.jp

Abstract. It is well known that the symmetry bias much accelerates the vocabulary learning. In particular, the bias helps infants to connect objects with their names easily. However, grammar learning is another important aspect of language acquisition. In this study, we propose that the symmetry bias also would help to acquire grammar rules faster. We employ Iterated Learning Model, and revise it to include the symmetry bias. The result of the experiments shows that infants could abduce the meanings from incomprehensible utterances using the symmetry bias, and acquire the compositional grammar from a reduced amount of learning data.

1 Introduction

It is well known that infants under 17 months old only slowly acquire lexical items, and these lexicons are hardly fixed. Also, infants in these ages tend to misapply words to objects. On the other hand, infants over 18 months old can acquire new words very rapidly, e.g. 7∼15 words a day, and such lexical misapplication subsides [1]. We cannot explain this phenomenal learning, considering that infants cannot take account of all possible meanings from an utterance and they learn meanings of words only from a few limited examples. For this problem, several studies have suggested that infants infer meanings efficiently to limit possibilities in a situation using constraints called cognitive biases [2,3] D A simple expression of lexical acquisition is a mapping between an object and a lexical label. This mapping is considered to be generally difficult as is well known as 'gavagai problem' [4], though infants achieve this operation.

It has been reported that various kinds of cognitive biases work for infants to limit the possible word meanings [2,5,6,7,8,9], and among which, the *symmetry bias* is said to be saliently effective. The bias says, if infants are taught an object P has a lexical label Q, then they apply the label Q to the object P without being taught. This tendency is said to be one of the peculiar human skills, and many experiments have endorsed that other animals cannot do this reverse implication [10,11,12].

Our study is based on the Iterated Learning Model (ILM, hereafter) by Kirby [13], who introduced the notion of *compositionality* and *recursion* as fundamental features of grammar, and showed that they made it possible for a human

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 566–577, 2009.

to acquire compositional language without LAD (Language Acquisition Device), opposing to Chomsky's idea [14]. Also, he adopted the idea of two different domains of language [15,14,16,17], namely, I–language and E–language; I–language is the internal language corresponding to speaker's intention or meaning, while E–language is the external language, that is, utterances [16]. In his model, he regarded that a parent is a speaker agent and her infant is a listener agent. The speaker agent gives the listener agent a pair of a string of symbols as an utterance (E–language), and a predicate–argument structure (PAS) as its meaning (I–language). A number of utterances would form compositional grammar rules in listener's mind, being substrings are chunked. This process is iterated generation by generation, and finally, a certain generation would acquire a compact, limited number of grammar rules. We include the symmetry bias into this process. We implement agents with the bias in a virtual world, and make them learn a grammar by computer simulation.

This paper is organized as follows: in Section 2 we explain Kirby's ILM [13] and revise it to include the symmetry bias. Section 3 presents the details of our experimental model, and gives specific experiment designs. We analyze our experimental results in Section 4, and conclude and discuss our issues in Section 5.

2 Iterated Learning Model with the Symmetry Bias

2.1 Utterance Rule of Kirby's Model

According to Kirby's model, we show a pair of I–language and E-language as follows.

$$S/eat(john, apple) \rightarrow \text{eatjohnapple}$$

where a speaker's intention is a PAS '$eat(john, apple)$' and its utterance becomes 'eatjohnapple'; the symbol 'S' stands for that they belong to category Sentence. Thus, as far as a listener is given an utterance paired with its meaning (PAS), the listener can understand the speaker's intention precisely at all times. However, compared to the actual situation, it seems a very strong assumption. In our model, we loosen this assumption and regard that some utterances lack meanings, to show the efficacy of the cognitive bias.

2.2 Rule Subsumption

The listener agent has an ability to change his knowledge with learning. The learning algorithm consists of the following three operations such as *chunk*, *merge*, and *replace* [13].

Chunk. This operation takes pairs of rules and looks for the least–general generalization. For example,

$$\begin{cases} S/read(john, book) \rightarrow \text{ivnre} \\ S/read(mary, book) \rightarrow \text{ivnho} \end{cases} \Rightarrow \begin{cases} S/read(x, book) \rightarrow \text{ivn}N/x \\ N/john \rightarrow \text{re} \\ N/mary \rightarrow \text{ho} \end{cases}$$

Merge. If two rules have the same meanings and strings, replace their nonterminal symbols for one common symbol.

$$\begin{cases} S/read(x, book) \rightarrow \text{ivn}A/x \\ A/john \rightarrow \text{re} \\ A/mary \rightarrow \text{ho} \\ S/eat(x, apple) \rightarrow \text{apr}B/x \\ B/john \rightarrow \text{re} \\ B/pete \rightarrow \text{wqi} \end{cases} \Rightarrow \begin{cases} S/read(x, book) \rightarrow \text{ivn}A/x \\ A/john \rightarrow \text{re} \\ A/mary \rightarrow \text{ho} \\ S/eat(x, apple) \rightarrow \text{apr}A/x \\ A/pete \rightarrow \text{wqi} \end{cases}$$

Replace. If a rule is embeddable in another rule, replace the latter for a compositional rule with variables.

$$\begin{cases} S/read(pete, book) \rightarrow \text{ivnwqi} \\ B/pete \rightarrow \text{wqi} \end{cases} \Rightarrow \begin{cases} S/read(x, book) \rightarrow \text{ivn}B/x \\ B/pete \rightarrow \text{wqi} \end{cases}$$

2.3 Abductive Reasoning

An utterance lacking its meaning (PAS) corresponds to the following situations.

- A listener cannot understand what the intention of a speaker's utterance is.
- The listener fails to communicate with the speaker because of lacking other modalities, like finger pointing.

Even such cases, the listener attempts to complement the speaker's intention by using his own previously acquired knowledge. We define this notion as the *symmetry bias*. For example, if the listener cannot get the meaning of the left-hand side of '$\rightarrow$' in

$$S/p(a, b) \rightarrow \text{fjaljla},$$

The listener guesses its meaning backward, namely:

$$??? \leftarrow \text{fjaljla}.$$

This backward directional guess is regarded as the effect of the symmetry bias, and we build this process into our model. In ordinary circumstances, this abductive reasoning can be a mistake though there is a possibility to accelerate the language acquisition.

2.4 Extensional Model by the Symmetry Bias

The relation between an utterance and a meaning is not only one–to–one mapping, but also one–to–many or many–to–one. However, from the viewpoint of our linguistic performance, our language strongly favors one–to–one mapping. Thus, it can be reasonably expected that infants use the symmetric bias to obtain a meaning of the utterance if the infants possess a certain level of grammatical/lexical knowledge. Now, we summarize our intuition which we mentioned so far, as a policy of our study.

1. In the actual world, a listener cannot always understand a speaker's intention from an utterance.
2. However, the listener accepts this utterance as a reasonable linguistic representation, and he has an ability to understand it using his own knowledge.
3. An utterance is composed by the speaker's intention, and the utterance reflects the speaker's intention.
4. The process inferring the meaning from an utterance has the reverse direction of the utterance generation.
5. We regard this inference is caused by the symmetry bias in language acquisition.
6. If most of meaning/utterance pairs are one-to-one, the symmetry bias must work effectively.

For verifying the above intuition, we have the following conjectures.

Conjecture 1. If a listener receives only an utterance, he looks for similar utterances that he has once received. If the utterance is new, he has to generate an appropriate meaning. In terms of computer simulation, this may increase computational time due to the addition of learning process, compared with Kirby's model.

Conjecture 2. However, if the listener's partial grammatical/lexical knowledge is sufficient, he may be able to complement meanings of incomprehensible utterances. Thus, the learning process saves memory space.

In the next section, we show the adequacy of these conjectures by computer simulation.

3 Experiments in Symmetry Bias Model

In this section, we show the procedure and the result of our experiment. The purpose of the experiment is to demonstrate acquisition of compositional grammar, even in a case that a listener agent may not always understand the meaning of an utterance. In order to examine the efficacy of our model, we compare our symmetry bias model to the original model.

First of all, we reiterate the experiment of Kirby's model as a pilot one, to grasp the features of the original model; how many generations are needed to organize a compositional grammar, when an agent acquires a grammar that can represent the whole meaning space, how many grammar rules the agent acquires, and so on.

After the pilot experiment, we examine the following three strategies when a listener cannot understand the meaning of an utterance.

(I) The listener ignores such utterances, and does not use them in his learning.
(II) The listener complements meanings of such utterances, randomly assigning his previous meanings.
(III) The listener applies the symmetry bias to such utterances to complement their meanings, and uses them in his learning.

The purpose of experiment (I) is to observe differences of acquired grammar, dependent on the data amount for learning, by comparison with the pilot experiment. Also, we compare experiment (I) to experiment (III) to observe the effect of complemental process. Next, we compare experiment (II) to experiment (III) to observe the superiority of the symmetry bias to the mere simple meaning complementation.

3.1 Experimentation Environment

In our model, we have employed the following five predicates and five object words, which are the same as Kirby's experiment [13].

predicates:	admire, detest, hate, like, love
objects:	gavin, heather, john, mary, pete

The arguments of a predicate must not be identical, i.e., representations like $love(pete, pete)$ are prohibited. This implies that there are 100 distinct meanings (5 predicates $\times$ 5 possible first arguments $\times$ 4 possible second arguments). Algorithm 1 is the procedure of the simulation.

```
generation := 0;
repeat
    a speaker := a parent;
    a listener := an infant;
    for 1 to 50 do
        the speaker chooses a meaning (PAS) from the whole meaning space;
        the speaker generates an utterance by her own grammar rules;
        if the speaker could not generate one then
            attach random string to the meaning (PAS);
            put the rule into her grammar set;
        end
        give string–PAS to the listener;
        the listener guesses grammar rules in his own mind;
    end
    discard the speaker, together with her grammar;
    a parent := the listener; /*the infant becomes a new parent*/
    an infant is created;
    generation ++;
until the grammar converges ;
```

Algorithm 1. The simulation algorithm

Because the number of utterances is limited to 50 times, the listener cannot learn the whole meaning space, the size of which is 100. To obtain the whole meaning space, the listener has to generalize his own knowledge by learning.

To evaluate the accomplishment of the learning, we investigate *expressivity* and *compositional level* in the following definitions, as well as the number of grammar rules.

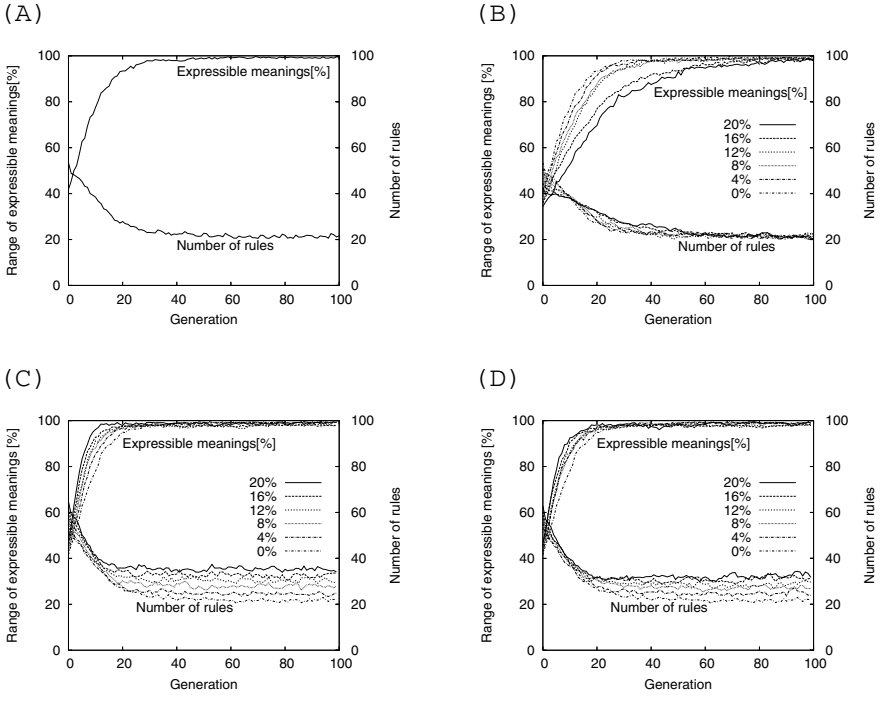


Fig. 1. The movement of the number of rules and expressivity per generation. (A). The result of Kirby(2002). (B). Result of experiment (I). (C). Result of experiment (II). (D). Result of experiment (III).

Definition 1 (Compositional rule, Holistic rule, and Lexical rule). *A grammar rule including non-terminal symbols for categories is called a compositional rule while the others holistic rules. Also, a rule which consists of a monadic terminal constant is called a lexical rule*[1]*.*

Definition 2 (Compositional level and Expressivity). *Compositional level is the number of variables included in a compositional rule. Expressivity is the ratio of the utterable meanings against the whole meaning space.*

The experiment was carried out until the 100th generation. In fact, we have carried out until the 1000th generation; however, both of expressivity and the number of rules converged until the 100th generation, and thus we discuss the result derived by the 100th generation.

3.2 Pilot Experiment: Kirby's Model

Figure 1(A) is the result of the average of 100 trials, and shows the tendency of the number of rules and expressivity per generation. In the early stages, the language has low expressivity and a large size of grammar rules; however, through

[1] E.g., $N/john \rightarrow$ j.

the generations, the language universally acquires higher expressivity and the number of grammar rules subsides.

In the early stages, the increase of holistic rules leads the growth of expressivity, and the acceleration of generalization. This process leads the grammar to be compositional, so most of the rules in the later stages become compositional.

In fact, we have found a grammar with 100 % of expressivity consisting of 11 rules, consisting of ten lexical rules and one level–3 compositional rule (see Definition 2).

3.3 Experiment (I): Listener Ignores Incomprehensible Utterances

In this experiment, the listener does not use incomprehensible utterances for learning, i.e., the experimental setting is the same as the pilot experiment, except that the number of available utterances is reduced.

Figure 1(B) is the result of the average of 100 trials. Each line denotes the rate of the incomprehensible utterances to the received utterances (50 times), that is, 0 %, 4 %, 8 %, 12 %, 16 %, and 20 %.

From Figure 1(B), we can observe that the grammars anyhow converge regardless of the incomprehensible rate, and there is not a significant difference in the number of rules. However, there is a difference in convergence generation. When the incomprehensible rate is 0 %, the number of grammar rules converges around the 30th generation, but when the rate is 20 %, it becomes around the 50th generation and the expressivity does not converge until around the 80th generation. This indicates that the listener with more amount of learning data can acquire the higher expressivity and the smaller grammar rules in a shorter period.

3.4 Experiment (II): Listener Arbitrarily Interprets Incomprehensible Utterances

In this experiment, the listener chooses a meaning from the meaning space arbitrarily, and puts it to an incomprehensible utterance.

Figure 1(C) is the result of the average of 100 trials. Each line denotes the rate of the incomprehensible utterances to the received utterances as 0 %, 4 %, 8 %, 12 %, 16 %, 20 %, as Figure 1(B).

As compared to experiment (I) (Figure 1(B)), the result of experiment (II) (Figure 1(C)) shows high expressivity in shorter period. Complementing meanings to incomprehensible utterances, the listener can superficially increase his grammar knowledge. However, these ungrounded rules become noise to generalize the knowledge, and thus the size of the grammar rules becomes larger. In terms of expressivity, there is an insignificant difference between experiment (I) and experiment (II) after convergence.

3.5 Experiment (III): Listener Applies the Symmetry Bias to Interpret Incomprehensible Utterances

If the listener receives incomprehensible utterances, he follows Algorithm 2.

```
if there is a string among compositional level 0 rules which exactly matches the
utterance then
    apply the meaning of this rule;
else
    if the listener can derive the same string as the utterance using rules whose
    compositional levels are greater than or equal to 1 then
        apply the meaning derived from these rules;
    else
        Decompose the string to substrings;
        Search lexical rules which match longest to any one of the substrings;
        if there exists such lexical rules then
            The listener chooses utterances which include any one of such words;
            From the chosen utterances, the listener picks out utterances as
            candidates which are uttered most frequently;
            if the candidate is only one then
                apply the meaning of this candidate to the incomprehensible
                utterance;
            else
                Among the other substrings, search lexical rules which correspond
                to one of the substrings longest;
                Choose the most frequent candidates;
                if such candidate is only one then
                    apply the meaning of this candidate to the incomprehensible
                    utterance;
                else
                    Choose one from the candidates;
                end
            end
        else
            Choose a meaning from the meaning space randomly, and apply it to
            the incomprehensible utterance.
        end
    end
end
```

Algorithm 2. The bias algorithm

Similar to the previous three experiments, the incomprehensible rates are set to 0 %, 4 %, 8 %, 12 %, 16 %, and 20 %. However, different from previous three experiments, incomprehensible utterances are given at the tail of the 50 utterances, e.g., in case of 20%, the first 40 sentences are paired with PAS while the rest 10 utterances are meaningless. This is because Algorithm 2 must be evoked after a certain accumulation of grammar knowledge. Namely, the listener could consider in such a way that the intention of the speaker was not clear but referring back to the previous knowledge the listener could partially guess what the speaker had said. We contend that this is exactly the inference by the symmetry bias.

Figure 1(D), the result of the average of 100 trials, shows that the expressivity is higher when the listener does not understand all the utterances, since the listener augments the number of grammar rules as experiment (II).

4 Comparison of Experimental Results

In this chapter, we compare the result of experiment (I) to that of experiment (III) and the result of experiment (II) to that of experiment (III), where we use the results of incomprehensible rate is 20 % as prominent examples. Table 1 shows the average number of rules at the 100th generation.

Firstly, we compare experiment (I) to experiment (III). By the effect of inference, the listener can get more learning data excluding noise data for leaning. This is why we can observe the expressivity of experiment (III) converges faster than experiment (I), i.e., inferring the meanings of incomprehensible utterances is more efficient in terms of convergence speed.

Here, we show how the symmetry bias works to construct compositional grammars using a concrete example from experiment (III) where the incomprehensible rate is 20 %, and the speaker's 45th utterance is 'esk'. The listener has the following lexical rule:

$$A/love \rightarrow \mathrm{s}$$

The utterance 'esk' being decomposed to substrings, 's' corresponds to the lexical rule, so the listener infers as follows:

$$love(???, ???) \leftarrow \mathrm{esk}$$

Investigating the frequency of the words which co-occur with the word *love* from the speaker's utterances up to this time, the listener finds *gavin* has occurred three times, *john* has occurred twice, and *heather*, *mary*, *pete* has occurred once, respectively. Thus, the listener chooses *gavin*. Next, the listener chooses either one of the followings based on frequency:

$$love(gavin, ???)$$

$$love(???, gavin)$$

Table 1. Average number of rules: Generation 100

	experiment (I)	experiment (II)	experiment (III)
0 %	21.1	22.1	22.1
4 %	22.3	24.9	24.0
8 %	20.9	28.0	26.8
12 %	22.6	29.6	29.7
16 %	20.0	33.6	31.3
20 %	20.5	34.1	31.8

Looking back to the log, *mary* has occurred once in the form of $love(mary, gavin)$ and the others have not occurred. Thus, the listener adds the following rule to his knowledge:

$$love(mary, gavin) \rightarrow \text{esk}$$

This enables the listener to learn a lexical rule '$A/love \rightarrow$ s', so his grammar changes as follows:

$$\begin{cases} A/love \rightarrow \text{s} \\ S/p(mary, gavin) \rightarrow \text{ek } A/p \end{cases}$$

Thus, the listener acquires the compositional rule. Since the compositional level of the added rules is higher, the language necessarily has higher expressivity. Therefore, compared experiment (I) to experiment (III), the expressivity of the latter converges in shorter period.

When the complement of meanings by inference is incomplete, the listener acquires rules which were originally not included in the speaker's grammars, so the listener cannot generalize such rules well. Thus, there remains low level compositional rules in the listener's grammar. This is why Table 1 shows that the number of rules of experiment (I) is less than that of experiment (III) as 20.5 versus 31.8.

The listener's strategy of experiment (I) which does not treat incomprehensible utterances to learning, decrease data for learning. On the other hand, the listener's strategy of experiment (III) keeps the same amount of data for learning. The symmetry bias being applied, the grammar converges in shorter period, although the number of rules increases. In terms of convergence speed, it is efficient to apply the symmetry bias to learning.

Next, we compare experiment (II) to experiment (III) under the incomprehensible rate is 20 %. There are not so much of differences in convergence speed and the expressivity between these experiments. On the other hand, there is a difference in the number of rules between them, as 34.1 and 31.8. This difference indicates the efficiency of the symmetry bias in learning.

Figure 2 shows the number of guessed meanings by the symmetry bias under the rate is 20 % by the average of 100 trials. In early stages, because of the failure of the

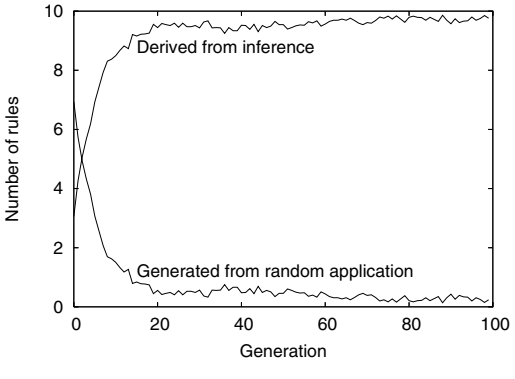


Fig. 2. The number of rules which are derived from inference vs. random application

inference, the listener has applied meanings to the incomprehensible utterances randomly. In later generations, the listener becomes to be able to apply a meaning by the inference. From Figure2, we can observe that the number of rules with random PAS decreases, while those which own correct PAS increases. Also, from figure1(D), we can see that the expressivity and the number of rules converges around the 20th generation. Thus, we can observe that the inference failure had steadily decreased after the agent acquired a certain level of grammar knowledge.

If some rules are added to the grammar knowledge by the symmetry bias, the compositional rules are inevitably generated. On the other hand, in the case of random meaning application, it is not always true that the learning produces compositional rules.

5 Conclusion

In this paper, we verified the efficacy of the symmetry bias not only in the lexical acquisition, but also in the grammar acquisition. For this purpose, we have revised Kirby's model [13], and have built the symmetry bias into our model. In the simulation, the listener received utterances without the speaker's intention at a certain rate, and the listener interpreted such incomprehensible utterances with three kinds of strategies such as (I) ignoring, (II) attaching meanings randomly, and (III) attaching meanings by the symmetry bias. For each of the strategies, we observed expressivity, number of rules, and compositional level. As a result of the experiments, the listener who has employed strategy (III),

- could acquire high expressivity faster than those who took the strategy (I).
- could construct his grammar with a higher compositional level, i.e., the number of rules was smaller than that of the listener who employed strategy (II).

Our future works are summarized as follows. Firstly, we should consider to include other cognitive biases than the symmetry bias. Secondly, we should expand the meaning space and grammars of our model and restudy the effectiveness, since the effect of the bias was not so prominant at the current simple and limited language. Thirdly, what happens if we attach incorrect PASs intentionally? If infants are much more versatile in learning language, they also may be able to learn a grammar robustly.

Acknowledgment

The authors would like to thank Shoki Sakamoto of JAIST who contributed to implement our experimental system.

References

1. Schafer, G., Plunkett, K.: Rapid word learning by 15-month-olds under tightly controlled conditions. Child Development 69, 309–320 (1998)
2. Imai, M., Gentner, D.: Children's theory of word meanings: The role of shape similarity in early acquisition. Cognitive Development 9(1), 45–75 (1994)

3. Markman, E.M.: Categorization and naming in children: Problems of induction. MIT Press, Cambridge (1989)
4. Quine, W.V.O.: Word and Object. MIT Press, Cambridge (1960)
5. Imai, M., Gentner, D.: A crosslinguistic study of early word meaning: Universal ontology and linguistic influence. Cognition 62(2), 169–200 (1997)
6. Landau, B., Smith, L.B., Jones, S.S.: The importance of shape in early lexical learning. Cognitive Development 3(3), 299–321 (1988)
7. Landau, B., Smith, L.B., Jones, S.S.: Syntactic context and the shape bias in children's and adult's lexical learning. Journal of Memory and Language 31(6), 807–825 (1992)
8. Markman, E.M.: Constraints children place on word meanings. Cognitive Science: A Multidisciplinary Journal 14(1), 57–77 (1990)
9. Markman, E.M., Wasow, J.L., Hansen, M.B.: Use of the mutual exclusivity assumption by young word learners. Cognitive Psychology 47(3), 241–275 (2003)
10. Hattori, M.: Adaptive Heuristics of Covariation Detection: A Model of Causal Induction. In: Proceedings of the 4th International Conference on Cognitive Science and the 7th Australasian Society for Cognitive Science Joint Conference (ICCS/ASCS 2003), vol. 1, pp. 163–168 (2003)
11. Sidman, M., Rauzin, R., Lazar, R., Cunningham, S., Tailby, W., Carrigan, P.: A search for symmetry in the conditional discriminations of rhesus monkeys, baboons, and children. Journal of the Experimental Analysis of Behavior 37(1), 23–44 (1982)
12. Yamazaki, Y.: Logical and illogical behavior in animals. Japanese Psychological Research 46(3), 195–206 (2004)
13. Kirby, S.: Learning, bottlenecks and the evolution of recursive syntax. In: Linguistic Evolution through Language Acquisition. Cambridge University Press, Cambridge (2002)
14. Chomsky, N.: Knowledge of Language. Praeger, New York (1986)
15. Bickerton, D.: Language and Species. University of Chicago Press (1990)
16. Hurford, J.: Language and Number: the Emergence of a Cognitive System. Blackwell, Oxford (1987)
17. Kirby, S.: Function, Selection, and Innateness: The Emergence of Language Universals. Oxford University Press, Oxford (1999)

A Series of Run-Rich Strings

Wataru Matsubara[1], Kazuhiko Kusano[1], Hideo Bannai[2], and Ayumi Shinohara[1]

[1] Graduate School of Information Science, Tohoku University, Aramaki aza Aoba 6-6-05, Aoba-ku, Sendai 980-8579, Japan
{matsubara@shino.,kusano@shino.,ayumi@}ecei.tohoku.ac.jp
[2] Department of Informatics, Kyushu University, 744 Motooka, Nishiku, Fukuoka 819-0395 Japan
bannai@i.kyushu-u.ac.jp

Abstract. We present a new series of run-rich strings, and give a new lower bound 0.94457567 of the maximum number of runs in a string. We also introduce the general conjecture about a asymptotic behavior of the numbers of runs in the strings defined by any recurrence formula, and show the lower bound can be improved further to 0.94457571235.

1 Introduction

Repetitions in strings is an important element in the analysis and processing of strings. It was shown in [1] that when considering *maximal repetitions*, or *runs*, the maximum number of runs $\rho(n)$ in any string of length n is $O(n)$, leading to a linear time algorithm for computing all the runs in a string. Although they were not able to give bounds for the constant factor, there have been several works to this end [2,3,4,5,6,7,8]. The currently known best upper bound[1] is $\rho(n) \leq 1.029n$, obtained by calculations based on the proof technique of [5,8]. The technique bounds the number of runs for each string by considering runs in two parts: runs with long periods, and runs with short periods. The former is more sparse and easier to bound while the latter is bounded by an exhaustive calculation concerning how runs of different periods can overlap in an interval of some length.

On the other hand, an asymptotic lower bound on $\rho(n)$ was first presented in [9], where it is shown that for any $\varepsilon > 0$, there exists an integer $N > 0$ such that for any $n > N$, $\rho(n) \geq (\alpha - \varepsilon)n$, where $\alpha = \frac{3}{1+\sqrt{5}} \approx 0.927$. Although it was conjectured in [10] that this bound is optimal, a new bound was shown in [11], improving the lower bound to $\alpha = 174719/184973 \approx 0.944565$. The bound was obtained by considering the runs of an infinite series of strings $w, w^2, w^3, \ldots$, based on a run-rich string w. To the best of our knowledge, the current best lower bound is $\alpha = 27578248/29196442 \approx 0.9445756438404378$ achieved by a run-rich string discovered by Simon Puglisi and Jamie Simpson[2].

In this paper, we design a new series of run-rich strings defined by a simple recurrence formula, and improve the bound further to 0.94457567. We give a

[1] Presented on the website http://www.csd.uwo.ca/faculty/ilie/runs.html
[2] Personal communication.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 578–587, 2009.

conjecture for the exact number of runs contained in each string of the series, and show that the series improves the bound further to $\alpha \approx 0.94457571235$.

2 Preliminaries

Let Σ be a finite set of symbols, called an *alphabet*. Strings x, y and z are said to be a *prefix*, *substring*, and *suffix* of the string $w = xyz$, respectively.

The length of a string w is denoted by $|w|$. The i-th symbol of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$, and the substring of w that begins at position i and ends at position j is denoted by $w[i:j]$ for $1 \leq i \leq j \leq |w|$. A string w has period p if $w[i] = w[i+p]$ for $1 \leq i \leq |w| - p$. A string w is called *primitive* if w cannot be written as u^k, where k is a positive integer, $k \geq 2$.

A string u is a *run* if it is periodic with (minimum) period $p \leq |u|/2$. A substring $u = w[i:j]$ of w is a *run in* w if it is a run of period p and neither $w[i-1:j]$ nor $w[i:j+1]$ is a run of period p, that means the run is maximal. We denote the run $u = w[i:j]$ in w by the triple $\langle i, j-i+1, p\rangle$ consisting of the begin position i, the length $|u|$, and the minimum period p of u. For a string w, we denote by $run(w)$ the number of runs in w.

For example, the string $\mathtt{aabaabaaaacaacac}$ contains the following 7 runs: $\langle 1,2,1\rangle = \mathtt{a}^2$, $\langle 4,2,1\rangle = \mathtt{a}^2$, $\langle 7,4,1\rangle = \mathtt{a}^4$, $\langle 12,2,1\rangle = \mathtt{a}^2$, $\langle 13,4,2\rangle = \mathtt{(ac)}^2$, $\langle 1,8,3\rangle = \mathtt{(aab)}^{\frac{8}{3}}$, and $\langle 9,7,3\rangle = \mathtt{(aac)}^{\frac{7}{3}}$. Thus $run(\mathtt{aabaabaaaacaacac}) = 7$.

We are interested in the behavior of the *maxrun function* defined for all $n > 0$ by

$$\rho(n) = \max\{run(w) \mid w \text{ is a string of length } n\}.$$

Franěk, Simpson and Smyth [10] showed a beautiful construction of a series of strings which contain many runs, and later Franěk and Qian Yang [9] formally proved a family of true asymptotic lower bounds arbitrarily close to $\frac{3}{1+\sqrt{5}}n$ as follows.

Theorem 1 ([9]). *For any $\varepsilon > 0$ there exists a positive integer N so that $\rho(n) \geq \left(\frac{3}{1+\sqrt{5}} - \varepsilon\right) n$ for any $n \geq N$.*

3 A New Series of Run-Rich Strings

In this section, we show a construction of a series of *run-rich* binary strings, and we give new lower bound of the number of runs in string. The series $\{t_n\}$ of strings is defined by

$$\begin{aligned} t_0 &= \mathtt{0110101101001011010}, \\ t_1 &= \mathtt{0110101101001}, \\ t_2 &= \mathtt{01101011010010110101101010}, \\ t_k &= \begin{cases} t_{k-1}t_{k-2} & (\text{if } k \mod 3 = 0), \\ t_{k-1}t_{k-4} & (\text{otherwise}), \end{cases} \\ &\quad \text{for any integer } k > 2. \end{aligned} \tag{1}$$

Table 1 shows the length of $\{t_n\}$ and the number of runs in $\{t_n\}$ for $i = 0, 1, \ldots, 44$. We actually counted the number of runs by implementing the linear-time algorithm proposed by Kolpakov and Kucherov [1] combined with the space-effiecient algorithm to compute Lempel Ziv Factorization proposed by Crochemore et al. [12]. In our PC with 18GB RAM, t_{44} was the longest possible string to be handled. As we can see, these strings contain many runs and the ratio $run(t_n)/|t_n|$ in the third column is monotonically increasing as n grows. We are interested in its limit value, and we will try to estimate it in Section 4.

Using this result in Table 1, we improve the bound. $\{t_n\}$ contains enough runs, but we can improve the bound further by considering the string t_n^k. First, we give a previous result about the number of runs in an infinite string obtained by concatenating the same string infinite many times.

Theorem 2 ([11]). *For any string w and any $\varepsilon > 0$, there exists a positive integer N such that for any $n \geq N$,*

$$\frac{\rho(n)}{n} > \frac{run(w^3) - run(w^2)}{|w|} - \varepsilon.$$

From Theorem 2, we show a new lower bound.

Theorem 3. *For any $\varepsilon > 0$ there exists a positive integer N so that $\rho(n) > (\alpha - \varepsilon)\, n$ for any $n \geq N$, where $\alpha = \frac{48396453}{51236184} \approx 0.94457567$.*

Proof. From Table 1, we have $|t_{41}| = 51236184$, $run(t_{41}) = 48396417$, $run(t_{41}^2) = 96792871$, and $run(t_{41}^3) = 145189324$. Therefore from Theorem 2, we have

$$\frac{\rho(n)}{n} > \frac{145189324 - 96792871}{51236184} - \varepsilon. \qquad \square$$

Needless to say this bound is not optimal. If we can calculate $run(t_n)$ for larger n, we would be able to obtain better bounds.

4 Analysis of Asymptotic Behavior

In this section, we analyze the asymptotic behavior of the number of runs in $\{t_n\}$. We conjecture that $\lim_{n\to\infty} run(t_n)/|t_n| \approx 0.94457571235$.

To make the analysis easier, we classify the strings of $\{t_n\}$ into the following three forms and we focus attention on $\{a_n\}$.

$$\begin{aligned} a_n &= t_{3m} = b_{n-1}c_{n-1}, \\ b_n &= t_{3m+1} = a_n a_{n-1}, \\ c_n &= t_{3m+2} = b_n b_{n-1}. \end{aligned}$$

Table 1. The length of $\{t_n\}$ and number of runs in $\{t_n\}$

n	$\lvert t_n\rvert$	$run(t_n)$	$run(t_n)/\lvert t_n\rvert$	$run(t_n^2)$	$run(t_n^3)$	$run(t_n^k)/k\lvert t_n\rvert$
0	19	13	0.6842105263	29	44	0.7894736842
1	13	7	0.5384615385	19	30	0.8461538462
2	24	17	0.7083333333	39	60	0.8750000000
3	37	28	0.7567567568	62	95	0.8918918919
4	56	47	0.8392857143	99	150	0.9107142857
5	69	56	0.8115942029	120	183	0.9130434783
6	125	110	0.8800000000	227	343	0.9280000000
7	162	143	0.8827160494	295	446	0.9320987654
8	218	197	0.9036697248	402	606	0.9357798165
9	380	346	0.9105263158	704	1061	0.9394736842
10	505	467	0.9247524752	943	1418	0.9405940594
11	667	617	0.9250374813	1246	1874	0.9415292354
12	1172	1094	0.9334470990	2200	3305	0.9428327645
13	1552	1451	0.9349226804	2916	4380	0.9432989691
14	2057	1930	0.9382596014	3872	5813	0.9436071949
15	3609	3391	0.9395954558	6799	10206	0.9440288168
16	4781	4501	0.9414348463	9016	13530	0.9441539427
17	6333	5964	0.9417337755	11945	17925	0.9442602242
18	11114	10480	0.9429548317	20977	31473	0.9443944574
19	14723	13887	0.9432180941	27793	41698	0.9444406711
20	19504	18405	0.9436525841	36827	55248	0.9444729286
21	34227	32307	0.9439039355	64636	96964	0.9445174862
22	45341	42808	0.9441344479	85635	128461	0.9445314395
23	60064	56712	0.9441928609	113446	170179	0.9445424880
24	105405	99540	0.9443574783	199102	298663	0.9445567098
25	139632	131868	0.9443966999	263760	395651	0.9445614186
26	184973	174698	0.9444513524	349418	524137	0.9445648824
27	324605	306586	0.9444894564	613199	919811	0.9445695538
28	430010	406152	0.9445175694	812328	1218503	0.9445710565
29	569642	538042	0.9445265623	1076111	1614179	0.9445722050
30	999652	944219	0.9445477026	1888465	2832710	0.9445737117
31	1324257	1250831	0.9445530588	2501691	3752550	0.9445742027
32	1754267	1657010	0.9445597506	3314047	4971083	0.9445745716
33	3078524	2907866	0.9445649928	5815764	8723661	0.9445750626
34	4078176	3852116	0.9445683560	7704261	11556405	0.9445752219
35	5402433	5102974	0.9445696041	10205980	15308985	0.9445753423
36	9480609	8955120	0.9445722316	17910272	26865423	0.9445755014
37	12559133	11863017	0.9445729255	23726068	35589118	0.9445755531
38	16637309	15715165	0.9445737288	31430362	47145558	0.9445755921
39	29196442	27578212	0.9445744108	55156461	82734709	0.9445756438
40	38677051	36533368	0.9445748074	73066770	109600171	0.9445756606
41	51236184	48396417	0.9445749707	96792871	145189324	0.9445756733
42	89913235	84929820	0.9445752897	N/A	N/A	N/A
43	119109677	112508068	0.9445753765	N/A	N/A	N/A
44	157786728	149041473	0.9445754715	N/A	N/A	N/A

By definition, we can get the closed form of $\{a_n\}$ as follows:

$$\begin{aligned} a_n &= b_{n-1}c_{n-1} \\ &= b_{n-1}b_{n-2}b_{n-1} \\ &= a_{n-1}a_{n-2}a_{n-2}a_{n-3}a_{n-1}a_{n-2}. \end{aligned}$$

So we will analyze the length of $\{a_{2n}\}$ in Section 4.1, and the number of runs in Section 4.2.

4.1 Length

At first we give the generating function of $|a_{2n}| = |t_{6n}|$.

Lemma 1. *Let $L_A(z)$ be the generating function of $|a_{2n}|$. $L_A(z)$ can be represented as follows:*

$$L_A(z) = \frac{-17z^2 + 65z - 19}{z^3 - 5z^2 + 10z - 1}.$$

Proof.

$$\begin{aligned} |a_k| &= |a_{k-1}a_{k-2}a_{k-2}a_{k-3}a_{k-1}a_{k-2}| \\ &= 2|a_{k-1}| + 3|a_{k-2}| + |a_{k-3}|. \end{aligned}$$

Let $g_n = |a_n|$,

$$\begin{aligned} g_0 &= |a_0| = 19, \\ g_1 &= |a_1| = 37, \\ g_2 &= |a_2| = 125, \\ g_n &= 2g_{n-1} + 3g_{n-2} + g_{n-3} \ (\ n > 2\ ). \end{aligned}$$

Therefore, we have general term of g_n as follows:

$$g_n = 2g_{n-1} + 3g_{n-2} + g_{n-3} + 19_{[n=0]} - 1_{[n=1]} - 6_{[n=2]},$$

where the expression $m_{[n=k]}$ means the function such that the result is m if $n = k$, and 0 if $n \neq k$.

Let $L(z)$ be the generating function of g_n. We have

$$\begin{aligned} L(z) &= 2\sum_n g_{n-1}z^n + 3\sum_n g_{n-2}z^n + \sum_n g_{n-3}z^n \\ &\quad + \sum_n (19_{[n=0]} - 1_{[n=1]} - 6_{[n=2]})z^n \\ &= 2zL(z) + 3z^2L(z) + z^3L(z) + 19 - z - 6z^2 \\ &= \frac{6z^2 + z - 19}{z^3 + 3z^2 + 2z - 1}. \end{aligned}$$

By definition, $|a_{2n}| = |t_{6n}| = |t_{3(2n)}| = g_{2n}$,

$$\begin{aligned}\sum_n g_{2n} z^{2n} &= \frac{1}{2}\left(L(z) + L(-z)\right) \\ &= \frac{1}{2}\left(\frac{6z^2+z-19}{z^3+3z^2+2z-1} + \frac{6z^2-z-19}{-z^3+3z^2-2z-1}\right) \\ &= \frac{1}{2}\left(\frac{-17z^4+65z^2-19}{z^6-5z^4+10z^2-1}\right).\end{aligned}$$

Therefore, the generating function of $|a_{2n}|$ is as follows:

$$L_A(z) = \sum_n |a_{2n}| z^n = \sum_n g_{2n} z^n = \frac{1}{2}\left(\frac{-17z^2+65z-19}{z^3-5z^2+10z-1}\right). \qquad \square$$

To solve this generating function, we use the following theorem. If $A(z)$ is a power series $\sum_{k\geq 0} a_k z^k$, we will find it convenient to write $[z^n]A(z) = a_n$.

Theorem 4 (Rational Expansion Theorem for Distinct Roots [13]). *If $R(z) = P(z)/Q(z)$, where $Q(z) = q_0(1-\rho_1 z)\ldots(1-\rho_\ell z)$ and the numbers $(\rho_1, \ldots, \rho_\ell)$ are distinct, and if $P(z)$ is a polynomial of degree less than ℓ, then*

$$[z^n]R(z) = a_1\rho_1^n + \cdots + a_\ell\rho_\ell^n, \text{ where } a_k = \frac{-\rho_k P(1/\rho_k)}{Q'(1/\rho_k)}.$$

Using this theorem, we will show the general term of $|a_n|$. Let $Q(z) = z^3 - 5z^2 + 10z - 1$ and $Q^R(z) = -z^3 + 10z^2 - 5z + 1$. Therefore $Q^R(z)$ is the "reflected" polynomial of $Q(z)$. Let (α, β, γ) be the roots of $Q^R(z)$. Therefore $Q^R(z) = (z-\alpha)(z-\beta)(z-\gamma)$, and $Q(z) = (1-\alpha z)(1-\beta z)(1-\gamma z)$. By Theorem 4, we have the general term of $|a_n|$ as follows.

Theorem 5. *$|a_n| = f(\alpha)\alpha^n + f(\beta)\beta^n + f(\gamma)\gamma^n$ for $f(x) = \frac{x\left(19x^2-65x+17\right)}{10x^2-10x+3}$, where (α, β, γ) are the roots of the equation $-z^3+10z^2-5z+1=0$. The values of α, β, and γ are as follows:*

$$\alpha = \frac{10}{3} + \frac{1}{3}\sqrt[3]{\frac{1577}{2} - \frac{21\sqrt{69}}{2}} + \frac{1}{3}\sqrt[3]{\frac{1}{2}\left(1577 + 21\sqrt{69}\right)},$$

$$\beta = \frac{10}{3} - \frac{1}{6}\left(1 - i\sqrt{3}\right)\sqrt[3]{\frac{1577}{2} - \frac{21\sqrt{69}}{2}} - \frac{1}{6}\left(1 + i\sqrt{3}\right)\sqrt[3]{\frac{1}{2}\left(1577 + 21\sqrt{69}\right)},$$

$$\gamma = \frac{10}{3} - \frac{1}{6}\left(1 + i\sqrt{3}\right)\sqrt[3]{\frac{1577}{2} - \frac{21\sqrt{69}}{2}} - \frac{1}{6}\left(1 - i\sqrt{3}\right)\sqrt[3]{\frac{1}{2}\left(1577 + 21\sqrt{69}\right)}.$$

4.2 Number of Runs

Instead of trying to count the numbers of runs in the strings defined by the recurrence (1) only, we take a general approach here. We address ourselves to

Table 2. The length of $\{a_n\}$ and number of runs in $\{a_n\}$

n	$\lvert a_n\rvert$	$run(a_n)$	Δ_n	$\Delta_n - \Delta_{n-2}$
0	19	13		
1	37	28		
2	125	110		
3	380	346	29	
4	1172	1094	44	
5	3609	3391	55	26
6	11114	10480	70	26
7	34227	32307	80	25
8	105405	99540	95	25
9	324605	306586	105	25
10	999652	944219	120	25
11	3078524	2907866	130	25
12	9480609	8955120	145	25
13	29196442	27578212	155	25
14	89913235	84929820	170	25

find general formulae which express the numbers of runs in strings defined by some recurrence, or equivalently, by some morphism.

Let m, k, $\gamma_1, \gamma_2 \ldots \gamma_k$ be integers such that $1 \leq \gamma_j \leq m$ for any $1 \leq j \leq k$, and $s_0, s_1, \ldots, s_{m-1} \in \Sigma^+$ be any nonempty strings. We consider a series of strings $\{s_n\}$ defined by the recurrence formula

$$s_n = s_{n-\gamma_1} s_{n-\gamma_2} \ldots s_{n-\gamma_k} \quad (n \geq m).$$

We pay our attentions to the quantity $\Delta_n = run(s_n) - \sum_{i=1}^{k} run(s_{n-\gamma_i})$. It is the difference between the number of newly created runs and the number of merged runs by the concatenation. Let p be the least common multiple of the two integers γ_1 and γ_k. We observe that $\{\Delta_n\}$ is a mixture of p arithmetic progressions with the same common difference, except initial several terms. More formally, we have the following conjecture.

Lemma 2 (Conjecture). *There exist integers A and n_0 such that $\Delta_n = \Delta_{n-p} + A$ for any $n \geq n_0$.*

For example, in $\{a_n\}$ we have $\Delta_n = \Delta_{n-2} + 25$ for $n \geq 5$ (see Table 2). Unfortunately, we have not succeeded to give a formal proof to the conjecture at this time of writing. However, we have verified it for various instances and encountered no counter examples. Based on the conjecture, we have the following corollary, which is very useful to calculate the generating function.

Corollary 1 (Based on conjecture Lemma 2). *There exist integers $A, B_0, \ldots, B_{p-1}$, and n_0 such that*

$$run(s_{pn+i}) = \sum_{j=1}^{k} (run(s_{pn+i-\gamma_j})) + An + B_i,$$

for any $n \geq n_0$.

Note that $a_n = a_{n-1}a_{n-2}a_{n-2}a_{n-3}a_{n-1}a_{n-2}$, from Corollary 1, we have $p = 2$ and the recurrence formula of $run(a_n)$ for large n as follows:

$$\begin{aligned} run(a_{2n}) &= 2run(a_{2n-1}) + 3run(a_{2n-2}) + run(a_{2n-3}) + 25n - 5, \\ run(a_{2n+1}) &= 2run(a_{2n}) + 3run(a_{2n-1}) + run(a_{2n-2}) + 25n + 5. \end{aligned}$$

Let us consider the progression $\{r_n\}$ defined by

$$\begin{aligned} r_0 &= 15, \\ r_1 &= 27, \\ r_2 &= 110, \\ r_{2k} &= 2r_{2k-1} + 3r_{2k-2} + r_{2k-3} + 25k - 5 \quad (k \geq 2), \\ r_{2k+1} &= 2r_{2k} + 3r_{2k-1} + r_{2k-2} + 25k + 5 \quad (k \geq 1). \end{aligned}$$

We can see that $run(a_n) = r_n$ for any $n \geq 2$.

To analyze the asymptotic behavior of $run(a_n)$, we give the general term of $\{r_n\}$.

Let $X(z), Y(z)$ be the generating functions of $\{r_{2n}\}$ and $\{r_{2n+1}\}$:

$$\begin{aligned} X(z) &= \sum r_{2n} z^n, \\ Y(z) &= \sum r_{2n+1} z^n. \end{aligned}$$

Then,

$$\begin{aligned} X(z) &= 2zY(z) + 3zX(z) + z^2Y(z) + \tfrac{25z}{(1-z)^2} - \tfrac{5z}{1-z} - 15 + 9z, \\ Y(z) &= 2X(z) + 3zY(z) + zX(z) + \tfrac{25z}{(1-z)^2} + \tfrac{5z}{1-z} - 3. \end{aligned}$$

To solve above simultaneous equations, we have

$$X(z) = -\frac{19z^4 - 103z^2 + 164z^2 - 70z + 15}{(z-1)^2(z^3 - 5z + 10z - 1)}.$$

Let α, β, γ are the roots of equation $-z^3 + 10z^2 - 5z + 1 = 0$. We have the general term r_{2n} from $X(z)$ as follows:

$$r_{2n} = g(\alpha)\alpha^n + g(\beta)\beta^n + g(\gamma)\gamma^n + O(n)$$

where,

$$g(x) = \frac{x\left(15x^4 - 70x^3 + 164x^2 - 103x + 19\right)}{12x^4 - 52x^3 + 6x^2 - 28x + 5}.$$

Therefore we have the lower bounds of the maximal number of runs.

Theorem 6 (Based on Conjecture Lemma 2)

$$\frac{\rho(n)}{n} \geq 0.94457571235.$$

Proof.

$$\begin{aligned}
\rho(n) &\geq \lim_{n\to\infty} \frac{run(a_{2n})}{|a_{2n}|} = \lim_{n\to\infty} \frac{r_{2n}}{|a_{2n}|} \\
&= \lim_{n\to\infty} \frac{g(\alpha)\alpha^n + g(\beta)\beta^n + g(\gamma)\gamma^n + O(n)}{f(\alpha)\alpha^n + f(\beta)\beta^n + f(\gamma)\gamma^n} \\
&= \frac{g(\alpha)\alpha^n}{f(\alpha)\alpha^n} \qquad (|\alpha| > |\beta| = |\gamma|) \\
&= \frac{\frac{\alpha\left(15\alpha^4-70\alpha^3+164\alpha^2-103\alpha+19\right)}{12\alpha^4-52\alpha^3+6\alpha^2-28\alpha+5}}{\frac{\alpha(19\alpha^2-65\alpha+17)}{10\alpha^2-10\alpha+3}} \\
&= \frac{\left(3-10\alpha+10\alpha^2\right)\left(99-488\alpha+889\alpha^2\right)}{\left(17-65\alpha+19\alpha^2\right)\left(73-356\alpha+683\alpha^2\right)} \\
&= \frac{7714 - 109145\sqrt[3]{\frac{2}{-27669823+9298929\sqrt{69}}} + \sqrt[3]{\frac{-27669823+9298929\sqrt{69}}{2}}}{8079} \\
&\approx 0.94457571235.
\end{aligned}$$

□

5 Conclusion

In this paper, we showed a new series $\{t_n\}$ of run-rich strings defined by a simple recurrence formula, and we succeeded to improve the lower bound to 0.94457567 of the maximum number of runs in a string by using concrete string t_{41}. If we count the number of runs in a more longer strings t_n^2 and t_n^3 for $n > 41$, the bound can be improved further. Moreover, we gave a conjecture about the numbers of runs in the strings defined by any recurrence formula. Based on the conjecture, we evaluated the value $\lim_{n\to\infty} run(t_n)/|t_n|$ accurately, which yields the best lower bound so far. We are trying to give a proof of the conjecture.

Recently, Baturo et al. [6] derived an explicit formula for the number of runs in any standard Sturmian words. Moreover, they showed how to compute the number of runs in a standard Sturmian words in linear time with respect to the size of its compressed representation, that is, the recurrences describing the string. We are interested in extending it to a general strings described by any recurrences for further research.

References

1. Kolpakov, R., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: Proc. 40th Annual Symposium on Foundations of Computer Science (FOCS 1999), pp. 596–604 (1999)

2. Rytter, W.: The number of runs in a string: Improved analysis of the linear upper bound. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 184–195. Springer, Heidelberg (2006)
3. Rytter, W.: The number of runs in a string. Inf. Comput. 205(9), 1459–1469 (2007)
4. Puglisi, S.J., Simpson, J., Smyth, W.F.: How many runs can a string contain? Theoretical Computer Science 401(1–3), 165–171 (2008)
5. Crochemore, M., Ilie, L.: Maximal repetitions in strings. J. Comput. Syst. Sci. 74, 796–807 (2008)
6. Baturo, P., Piatkowski, M., Rytter, W.: The number of runs in Sturmian words. In: Ibarra, O., Ravikumar, B. (eds.) CIAA 2008. LNCS, vol. 5148, pp. 252–261. Springer, Heidelberg (2008)
7. Giraud, M.: Not so many runs in strings. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 245–252. Springer, Heidelberg (2008)
8. Crochemore, M., Ilie, L., Tinta, L.: Towards a solution to the "Runs" conjecture. In: Ferragina, P., Landau, G.M. (eds.) CPM 2008. LNCS, vol. 5029, pp. 290–302. Springer, Heidelberg (2008)
9. Franěk, F., Yang, Q.: An asymptotic lower bound for the maximal-number-of-runs function. In: Proc. Prague Stringology Conference (PSC 2006), pp. 3–8 (2006)
10. Franěk, F., Simpson, R., Smyth, W.: The maximum number of runs in a string. In: Proc. 14th Australasian Workshop on Combinatorial Algorithms (AWOCA 2003), pp. 26–35 (2003)
11. Matsubara, W., Kusano, K., Ishino, A., Bannai, H., Shinohara, A.: New lower bounds for the maximum number of runs in a string. In: Proc. The Prague Stringology Conference 2008, pp. 140–145 (2008)
12. Crochemore, M., Ilie, L., Smyth, W.F.: A simple algorithm for computing the lempel ziv factorization. In: Proc. DCC 2008, pp. 482–488 (2008)
13. Graham, R., Knuth, D., Patashnik, O.: Concrete mathematics, 2nd edn. Addison-Wesley, Reading (1995)

On Accepting Networks of Evolutionary Processors with at Most Two Types of Nodes

Victor Mitrana[1,⋆] and Bianca Truthe[2]

[1] Faculty of Mathematics, University of Bucharest
Str. Academiei 14, 70109 Bucharest, Romania
and
Department of Information Systems and Computation
Technical University of Valencia,
Camino de Vera s/n. 46022 Valencia, Spain
mitrana@fmi.unibuc.ro

[2] Faculty of Computer Science, University of Magdeburg
P.O.Box 4120, 39016 Magdeburg, Germany
truthe@iws.cs.uni-magdeburg.de

Abstract. In this paper we investigate the computational power of accepting networks of evolutionary processors (ANEP for short) with filters defined in two ways: by regular sets and random contexts conditions, respectively. We consider ANEPs with all the nodes specialized in only one evolutionary operation (substitution, insertion and deletion) or in two operations out of these three.

1 Introduction

Motivated by some models of massively parallel computer architectures (see [1], [2]), networks of language processors have been introduced in [3]. Such a network can be considered as a graph where the nodes are sets of productions and at any moment of time a language generated by these productions is associated with every node.

Inspired by biological processes, a different variant of network of language processors which is called network with evolutionary processors has been introduced in [4]. The productions of the nodes in a network with evolutionary processors might be viewed as formal specifications of the point mutations known from biology. Each node is associated with a set of rules and all rules of a node are of the same type, namely either substitutions of one letter by another letter or insertions of letters or deletions of letters; each node is then called substitution node or insertion node or deletion node, respectively. The action of each node on the data it contains is precisely defined. An implicit assumption is that arbitrarily many copies of every word are available such that when a production can be applied at two or more sites in a word the production is applied at each site in different copies of the word.

⋆ Work supported by the Alexander von Humboldt Foundation.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 588–600, 2009.

In [5], a characterization of the complexity class $\mathbb{NP}$ based on accepting networks of evolutionary processors was presented. In [6], it was shown that every recursively enumerable language can be accepted by an accepting network of evolutionary processors with 24 nodes and random context filters. Such a network contains all types of nodes.

Accepting networks of evolutionary processors with regular filters were first investigated in [7], where only networks without insertion nodes were considered. In [8], the generative capacity of networks with regular filters and only two types of nodes was studied. In the present paper, we try to complete the picture of computational power of ANEPs with at most two types of nodes.

2 Some Notations and Definitions

Throughout the paper we assume that the reader is familiar with the basic notions of the theory of formal languages. We here only recall some notation and notions as they are used in the paper.

An *alphabet* is a finite and nonempty set of symbols. The cardinality of a finite set A is written $card(A)$. Any sequence of symbols from an alphabet V is called *word* over V. The set of all words over V is denoted by V^* and the empty word is denoted by ε. A language over V is a subset of V^*.

The length of a word x is denoted by $|x|$ while $alph(x)$ denotes the (with respect to inclusion) minimal alphabet W such that $x \in W^*$.

We say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ is a *substitution rule* if both a and b are not ε; it is a *deletion rule* if $a \neq \varepsilon$ and $b = \varepsilon$; it is an *insertion rule* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all substitution, deletion, and insertion rules over an alphabet V are denoted by Sub_V, Del_V, and Ins_V, respectively.
Given a rule σ as above and a word $w \in V^*$, we define the following *actions* of σ on w:

- If $\sigma \equiv a \to b \in Sub_V$, then $\sigma^*(w) = \begin{cases} \{ubv \mid \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \ \text{otherwise} \end{cases}$
- If $\sigma \equiv a \to \varepsilon \in Del_V$, then $\sigma^*(w) = \begin{cases} \{uv \mid \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \ \text{otherwise} \end{cases}$

$$\sigma^r(w) = \begin{cases} \{u \mid w = ua\}, \\ \{w\}, \ \text{otherwise} \end{cases} \qquad \sigma^l(w) = \begin{cases} \{v \mid w = av\}, \\ \{w\}, \ \text{otherwise} \end{cases}$$

- If $\sigma \equiv \varepsilon \to a \in Ins_V$, then

$$\sigma^*(w) = \{uav \mid \exists u, v \in V^* \ (w = uv)\}, \ \sigma^r(w) = \{wa\}, \ \sigma^l(w) = \{aw\}.$$

The action $\alpha \in \{*, l, r\}$ expresses the way of applying a substitution, a deletion or an insertion rule to a word, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. For every rule σ, any action $\alpha \in \{*, l, r\}$, and any $L \subseteq V^*$, we define the *α-action of σ on L* by $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. Given a finite set of rules M, we define the *α-action of M* on the word w and the language L by:

$$M^{\alpha}(w) = \bigcup_{\sigma \in M} \sigma^{\alpha}(w) \quad \text{and} \quad M^{\alpha}(L) = \bigcup_{w \in L} M^{\alpha}(w),$$

respectively. In what follows, we shall refer to the rewriting operations defined above as *evolutionary operations* since they may be viewed as linguistic formulations of local DNA mutations.

If $\theta : V^* \longrightarrow \{0,1\}$ is a predicate and $L \subseteq V^*$, we write $\theta(L) = L \cap \theta^{-1}(1)$.

We are interested in some special predicates. For two disjoint subsets P and F of an alphabet V, a regular set R over V, and a word x over V, we define the predicates

$\theta^{s,P,F}(x) = 1$ if and only if $P \subseteq alph(x)$ and $F \cap alph(x) = \emptyset$,
$\theta^{w,P,F}(x) = 1$ if and only if $alph(x) \cap P \neq \emptyset$ and $F \cap alph(x) = \emptyset$,
$\theta^{R}(x) = 1$ if and only if $x \in R$.

The conditions of the first two predicates is based on *random context conditions* defined by the two sets P (*permitting contexts/symbols*) and F (*forbidding contexts/symbols*). Informally, the first condition requires (s stands for strong) that all permitting symbols are and no forbidding symbol is present in x, while the second (w stands for weak) is a weaker variant such that at least one permitting symbol appears in x but still no forbidding symbol is present in x. We call these two predicates random context predicates. The third predicate asks for membership in a regular set, and is called a regular predicate.

An *evolutionary processor over* V is a tuple (M, φ, ψ), where:

– M is either a set of substitution, deletion, or insertion rules over the alphabet V; formally, $M \subseteq Sub_V$ or $M \subseteq Del_V$ or $M \subseteq Ins_V$. The set M represents the set of evolutionary rules of the processor. As one can see, a processor is "specialized" in one evolutionary operation, only,

– φ is the *input predicate*, while ψ is the *output predicate* of the processor.

Informally, these two predicates work as filters. A word w can enter or leave the processor, if it satisfies the predicate $\varphi(w)$ or $\psi(w)$, respectively.

An evolutionary processor (M, φ, ψ) is *non-inserting*, if M is either a set of substitution or deletion rules, it is *non-deleting*, if M is either a set of substitution or insertion rules, it is *non-substituting*, if M is either a set of insertion or deletion rules. Further, a node is an insertion, a deletion, or a substitution node, if $M \subseteq Ins_V$, or $M \subseteq Del_V$, or $M \subseteq Sub_V$, respectively. The set of all evolutionary processors is denoted by EP_V. The set of all non-inserting, non-deleting, non-substituting, insertion, deletion, and substitution processors over V is denoted by NIEP_V, NDEP_V, NSEP_V, IEP_V, DEP_V, or SEP_V, respectively.

We are interested in two types of filters: filters defined by random context conditions and filters defined by regular set conditions. An evolutionary processor $X \in \{\mathrm{NIEP}_V, \mathrm{NDEP}_V, \mathrm{NSEP}_V, \mathrm{IEP}_V, \mathrm{DEP}_V, \mathrm{SEP}_V\}$ is called a random context X processor, denoted by rc$X$, if its both predicates are of the form $\theta^{s,P,F}$ or of the form $\theta^{w,P,F}$ for certain disjoint subsets P and F of V. Such a processor is called regular processor, denoted by reg$X$ if its both predicates are of the form θ^{R} for some regular set $R \subseteq V^*$.

An *accepting network of evolutionary processors* (ANEP for short) is a 7-tuple $\Gamma = (V, U, G, N, \alpha, x_{In}, Out)$, where:

- V and U are the input and network alphabet, respectively, satisfying $V \subseteq U$.
- $G = (X_G, E_G)$ is an undirected graph without loops with the set of vertices X_G and the set of edges E_G. G is called the *underlying graph* of the network.
- $N : X_G \longrightarrow EP_V$ is a mapping which associates with each node $x \in X_G$ the evolutionary processor $N(x) = (M_x, \varphi_x, \psi_x)$.
- $\alpha : X_G \longrightarrow \{*, l, r\}$ is a mapping which associates with each node a type of action; $\alpha(x)$ gives the action mode of the rules of node x on the words existing in that node.
- $x_{In} \in X_G$ is the *input* node of Γ.
- $Out \subset X_G$ is the set of *output* nodes of Γ.

In a similar way one can define the accepting networks of non-inserting, non-deleting, non-substituting, insertion, deletion, and substitution processors denoted by ANNIEP, ANNDEP, ANNSEP, ANIEP, ANDEP, and ANSEP, respectively.

An accepting network as above is a random context or regular network if all its processors are random context or regular, respectively.

We say that $card(X_G)$ is the size of Γ. A *configuration* of an ANEP Γ as above is a mapping $C : X_G \longrightarrow 2_f^{V^*}$ which associates a finite set of words with every node of the graph. A configuration may be understood as the sets of words which are present in any node (or in the associated processor) at a given moment. Given a word $z \in V^*$, the initial configuration of Γ on z is defined by $C_0^{(z)}(x_{In}) = \{z\}$ and $C_0^{(z)}(x) = \emptyset$ for all $x \in X_G \setminus \{x_{In}\}$.

A configuration can change either by an *evolutionary step* or by a *communication step*. When changing by an evolutionary step, each component $C(x)$ of the configuration C is changed in accordance with the set of evolutionary rules M_x associated with the node x and the way of applying these rules $\alpha(x)$. Formally, we say that the configuration C' is obtained in *one evolutionary step* from the configuration C, written as $C \Longrightarrow C'$, if and only if

$$C'(x) = M_x^{\alpha(x)}(C(x)) \text{ for all } x \in X_G.$$

When changing by a communication step, each node processor $x \in X_G$ sends one copy of each word it has, which is able to pass the output filter of x, to all the node processors connected to x and receives all the words sent by any node processor connected with x provided that they can pass its input filter.

Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, if and only if

$$C'(x) = (C(x) - \psi_x(C(x))) \cup \bigcup_{\{x,y\} \in E_G} (\psi_y(C(y)) \cap \varphi_x(C(y))) \text{ for all } x \in X_G.$$

Note that words that cannot pass the output filter of a node remain in that node and can be further modified in the subsequent evolutionary steps, while

words that can pass the output filter of a node but cannot pass the input filter of any node are lost.

Let Γ be an ANEP, the computation of Γ on the input word $z \in V^*$ is a sequence of configurations $C_0^{(z)}, C_1^{(z)}, C_2^{(z)}, \ldots$, where $C_0^{(z)}$ is the initial configuration of Γ on z, $C_{2i}^{(z)} \Longrightarrow C_{2i+1}^{(z)}$ and $C_{2i+1}^{(z)} \vdash C_{2i+2}^{(z)}$, for all $i \geq 0$. Note that the configurations are changed by alternative steps. By the previous definitions, each configuration $C_i^{(z)}$ is uniquely determined by the configuration $C_{i-1}^{(z)}$. A computation *halts* (and it is said to be *weak (strong) halting*) if one of the following two conditions holds:

(i) There exists a configuration in which the set of words existing in at least one output node (all output nodes) is non-empty. In this case, the computation is said to be a weak (strong) *accepting computation*.

(ii) There exist two identical configurations obtained either in consecutive evolutionary steps or in consecutive communication steps.

The *languages weakly or strongly accepted* by Γ are defined as

$$L_{wa}(\Gamma) = \{z \in V^* \mid \text{ the computation of } \Gamma \text{ on } z \text{ is a weak accepting}\} \text{ and}$$
$$L_{sa}(\Gamma) = \{z \in V^* \mid \text{ the computation of } \Gamma \text{ on } z \text{ is strong accepting}\}.$$

In the theory of networks, some types of underlying graphs are common like *rings, stars, grids*, etc. Networks of evolutionary words processors, seen as language generating or accepting devices, with underlying graphs having these special forms have been considered in several papers, see, e. g., [9] for an early survey. We focus here on *complete* networks i. e., networks having a complete underlying graph. Therefore, in what follows we replace the graph G in the definition of an accepting network by the set of its nodes usually denoted by χ.

Moreover, we present an evolutionary network by its nodes x and the parameters corresponding to x, where instead of $\varphi^{\beta,PI_x,FI_x}$ and ψ^{β,PO_x,FO_x}, in case of random context processors, and instead of φ^{R_x} and $\varphi^{R'_x}$ for regular processors, we only mention $PI_x,\ FI_x,\ PO_x,\ FO_x, \beta$ and R_x, R'_x, respectively.

For an accepting mode $x \in \{wa, sa\}$, a type $y \in \{\text{rc}, \text{reg}\}$ of filters, and $Z \in \{\text{ANNIEP}, \text{ANNDEP}, \text{ANNSEP}, \text{ANIEP}, \text{ANDEP}, \text{ANSEP}\}$, by $\mathcal{L}_x(yZ)$ we denote the set of all languages which can be weakly/strongly accepted by networks of type Z with filters of type y. Note that we ignore the empty word when we define a language and the empty set when we define a class of languages.

3 Computational Power of Random Context Networks

3.1 Networks with Substitution Nodes

We start this section by establishing a relationship between the two classes $\mathcal{L}_{wa}(\text{ANSEP})$ and $\mathcal{L}_{sa}(\text{ANSEP})$. As it was expected, we have

Theorem 1. $\mathcal{L}_{wa}(\text{rcANSEP}) \subseteq \mathcal{L}_{sa}(\text{rcANSEP})$.

Proof. Actually, we prove a bit more general result, namely that for every ANSEP Γ there exists an ANSEP Γ' with one output node only and

$$L_{wa}(\Gamma) = L_{wa}(\Gamma') = L_{sa}(\Gamma').$$

Without loss of generality, we assume that the set of rules in every output node of Γ is empty and that all its filter types are strong. Indeed, if the filter type of one node is a weak one with P its input set of permitting symbols and F its input set of forbidding symbols, then this node can be replaced by $2^{card(P)} - 1$ output nodes, each of them having a strong filter type where the input set of permitting symbols is a nonempty subset of P and the input set of forbidding symbols is F, respectively. Further on, the output set of permitting and forbidding symbols of every such node is $\{Z\}$ and the empty set, respectively, where Z is a new symbol. Now, in order to get Γ', we add one more node to Γ, which is the unique output node of Γ'. This node can receive only those words containing the new symbol Z. We now associate with each output node of Γ a set of substitution rules formed by one substitution only, namely $X \to Z$, where X is an arbitrary symbol from the input set of permitting symbols of that node applied in the $*$ mode (if the input set of permitting symbols is empty, we take all substitution rules $X \to Z$ with X from the network alphabet). □

The class of languages computed by rcANSEPs is rather strange because it contains all regular languages over the unary alphabet, non-context-free languages but there are even singleton languages that fail to be accepted in any way by rcANSEPs.

The last assertion is immediate as any rcANSEP accepting a word of the form axb, where a and b are not necessarily equal, but x is a word of length at least two containing two distinct symbols, accepts also all words ayb with y being an arbitrary permutation of x.

Theorem 2. *The class* $\mathcal{L}_{wa}$(rcANSEP) *is closed under union.*

Proof. According to [7], the class $\mathcal{L}_{wa}$(rcANNIEP) is closed under union. There, for two networks Γ_1 and Γ_2, a network Γ is constructed using the networks Γ_1, Γ_2 and three additional substitution nodes such that Γ weakly accepts the language $L_{wa}(\Gamma_1) \cup L_{wa}(\Gamma_2)$. If both networks Γ_1 and Γ_2 have only substitution nodes, then also Γ has only substitution nodes. □

Theorem 3. *Every language R that is commutative and semi-linear can be weakly accepted by an* rcANSEP.

Proof. For every commutative, semi-linear language $R \subseteq \{a_1, a_2, \ldots, a_d\}^*$, there exist natural numbers $n \geq 1$, $r_i \geq 0$ for $1 \leq i \leq n$ and d-dimensional vectors p_i and $q_{i,j}$ for $1 \leq i \leq n$, $1 \leq j \leq r_i$ such that

$$R = \psi^{-1}(\bigcup_{i=1}^{n} \{p_i + \sum_{j=1}^{r_i} \alpha_{i,j} q_{i,j} \mid \alpha_{i,j} \in \mathbb{N} \text{ for } 1 \leq j \leq r_i\})$$

where ψ is the Parikh mapping. For $1 \leq i \leq n$, let

$$H_i = \{a_1^{p_{i1}} a_2^{p_{i2}} \dots a_d^{p_{id}} a_1^{\alpha_{i,1} q_{i,1_1}} a_2^{\alpha_{i,1} q_{i,1_2}} \dots a_d^{\alpha_{i,1} q_{i,1_d}} a_1^{\alpha_{i,r_i} q_{i,r_{i1}}} \dots a_d^{\alpha_{i,r_i} q_{i,r_{id}}} \mid \alpha_{i,j} \in \mathbb{N}, 1 \leq j \leq r_i\}.$$

Then, $R = \bigcup_{i=1}^{n} COMM(H_i)$ where $COMM(H_i)$ is the commutative closure of H_i. An rcANSEP accepting H_i, for some $1 \leq i \leq n$, can be constructed that contains $r_i + 1$ subnetworks. A subnetwork substitutes p_{i_1} occurrences of a_1 in the input word by the symbols $(a_1, p_{i_1}, 1), (a_1, p_{i_1}, 2), \dots, (a_1, p_{i_1}, p_{i_1})$, then p_{i2} occurrences of a_2 by the symbols $(a_2, p_{i2}, 1), (a_2, p_{i2}, 2), \dots, (a_2, p_{i2}, p_{i2})$ and so on. The derivation is not successful if the a_ks are consumed before (a_k, p_{ik}, p_{ik}) is written $(1 \leq k \leq d)$.

Then the subnetwork for a number j, $1 \leq j \leq r_i$ is choosen. This subnetwork receives words containing the symbols $(a_1, p_{i1}, 1), \dots, (a_d, p_{id}, p_{id})$ and possibly $\bar{a}$. It substitutes q_{i,j_k} occurrences of a_k by the symbols $(a'_k, q_{i,j_k}, 1)$, $(a'_k, q_{i,j_k}, 2), \dots, (a'_k, q_{i,j_k}, q_{i,j_k})$ for $k = 1, \dots, d$ as shown before. As soon as a word contains the symbols $(a'_k, q_{i,j_k}, q_{i,j_k})$ for $1 \leq k \leq d$, it enters a designated node where all these symbols are replaced by $\bar{a}$. The obtained word is now again an "input" for some subnetwork for a number j' until no occurrence of a_k, $1 \leq k \leq d$ is observed in the current word.

The output node of the network can receive only those words without any occurrence of $a_1, \dots, a_d$ but containing a symbol $(a_d, q_{i,j_d}, q_{i,j_d})$ for some number $j \in \{1, 2, \dots, r_i\}$. The network informally described here weakly accepts H_i.

By Theorem 2 (closure under union), there is also an rcANSEP that accepts the language R. □

Corollary 1. *Every regular language R over a unary alphabet can be weakly accepted by an* rcANSEP.

Proposition 1. *The class $\mathcal{L}_{wa}$*(rcANSEP) *contains non-context-free languages.*

Proof. We consider the language $L = \{w \in \{a, b, c\}^+ \mid |w|_a = |w|_b = |w|_c\}$ which is not context-free. We explain how this language can be weakly accepted by an rcANSEP. Exactly one occurrence of a, b, and c, in this order, is replaced by a', b', c', respectively. Words containing c' enter a node where the all the primed symbols are substituted by barred copies. Now the process resumes. When a word contains only barred symbols, it can enter the output node. □

3.2 Networks with Deletion Nodes

The results on the computational power of rcANDEPs we present in this section are rather similar to those presented in the previous section. Namely, we show that non-context-free languages can be weakly and strongly accepted by

rcANDEPs but there are singleton languages that cannot be accepted in any mode by any rcANDEP.

Proposition 2. *The classes* $\mathcal{L}_{wa}$(rcANDEP) *and* $\mathcal{L}_{sa}$(rcANDEP) *contain non-context-free languages.*

Proof. Let Γ be the following rcANDEP over the input alphabet $\{a, b, c, d, \#\}$ with one output node only:

$$x_{In} : \begin{cases} M = \{a \to \varepsilon\}, \\ PI = \{a, b, c, d, \#\}, FI = \emptyset, \\ PO = \emptyset, FO = \emptyset, \\ \alpha = *,\ \beta = s, \end{cases} \quad x_1 : \begin{cases} M = \{b \to \varepsilon\}, \\ PI = \{b, c, d, \#\}, FI = \emptyset, \\ PO = \emptyset, FO = \emptyset, \\ \alpha = *,\ \beta = s, \end{cases}$$

$$x_2 : \begin{cases} M = \{c \to \varepsilon\}, \\ PI = \{c, d, \#\}, FI = \{a, b\}, \\ PO = \emptyset, FO = \emptyset, \\ \alpha = *,\ \beta = s, \end{cases} \quad x_3 : \begin{cases} M = \{d \to \varepsilon\}, \\ PI = \{d, \#\}, FI = \{a, b\}, \\ PO = \emptyset, FO = \emptyset, \\ \alpha = *,\ \beta = s, \end{cases}$$

$$x_{Out} : \begin{cases} M = \emptyset, \\ PI = \{\#\}, FI = \{a, b, c, d\}, \\ PO = \emptyset, FO = \emptyset, \\ \alpha = *,\ \beta = s. \end{cases}$$

Let x be an input for this network. We assume that x contains all the letters of the alphabet $\{a, b, c, d, \#\}$. In the input node x_{In}, an arbitrary occurrence of a is deleted from x. All words obtained in this way are then received by x_1, where an arbitrary occurrence of b is deleted. All these words move to x_{In} again if a and b still occur. This "ping-pong" process continues until no occurrence of a and b is observed in the current word. If $|x|_a = |x|_b$, then the process of removing all occurrences of a and b is successful, while this process will eventually get stuck provided that $|x|_a \neq |x|_b$. Now all the words obtained from x by a successful process as above collapsed into only one word, say y, which contains occurrences of c, d and $\#$ only. This word enters simultaneously x_2 and x_3, where an occurrence of c and d is deleted, respectively. Again a "ping-pong" game between the nodes x_2 and x_3 similar to that presented above takes place. All occurrences of c and d of y can be successfully deleted if and only if either $|x|_c = |x|_d$ or $|x|_d = |x|_c + 1$. After they were successfully removed, the resulting word enters x_{Out} and the computation ends by accepting the input word x. It follows $L_{wa}(\Gamma) \cap a^+c^+b^+\#d^+ = \{a^n c^m b^n \# d^p \mid n, m \geq 1, (p = m) \vee (p = m+1)\}$, which implies that $L_{wa}(\Gamma)$ is not context-free. Moreover, $L_{wa}(\Gamma) = L_{sa}(\Gamma)$ holds, which completes the proof. □

We finish this section with the following result.

Theorem 4. *A language over the unary alphabet is weakly/strongly accepted by an* rcANDEP *if and only if it is one of these languages:* $\{a\}$, $\{aa\}$, $\{a\}\{a\}^*$, *or* $\{aa\}\{a\}^*$.

Proof. The reader can easily find rcANDEPs that weakly/strongly accept each of the languages mentioned in the statement.

Conversely, one can easily prove that an rcANDEP having exactly one deletion node (except the output ones) can accept $\{a\}$, $\{a\}\{a\}^*$, or $\{aa\}\{a\}^*$. Further on, every rcANDEP with at least two deletion nodes (except the output ones) can accept one of the languages $\{a\}$, $\{aa\}$, $\{a\}\{a\}^*$, or $\{aa\}\{a\}^*$. □

Corollary 2. *Neither the class* $\mathcal{L}_{wa}$(rcANDEP) *nor* $\mathcal{L}_{sa}$(rcANDEP) *is closed under union.*

3.3 Networks with Insertion Nodes

This case is pretty simple and can be solved completely. As far as the computational power of rcANIEPs is concerned, we can characterize precisely the class of languages computed by these networks.

Theorem 5. *A language* L *over the alphabet* V *is weakly/strongly accepted by an* rcANIEP *if and only if there are the subsets* $V_1, V_2, \ldots, V_n$ *of* V*, for some* $n \geq 1$*, not necessarily pairwise disjoint, such that*

$$L = \bigcup_{i=1}^{n} \{x \in V_i^+ \mid |x|_a \geq 1, \text{ for all } a \in V_i\}.$$

Proof. It is plain that the language $\bigcup_{i=1}^{n} \{x \in V_i^+ \mid |x|_a \geq 1, \text{ for all } a \in V_i\}$ can be weakly accepted by an rcANIEP for any subsets $V_1, V_2, \ldots, V_n$ of an alphabet V. The converse statement follows immediately as soon as we note that if a word z is weakly/strongly accepted by an rcANIEP Γ, then all words of the language $\{x \in alph(z)^+ \mid |x|_a \geq 1, \text{ for all } a \in alph(z)\}$ are also accepted by Γ. □

From Theorem 5, we obtain the following results.

Corollary 3. $\mathcal{L}_{wa}$(rcANIEP) = $\mathcal{L}_{sa}$(rcANIEP).

Corollary 4. *Both classes* $\mathcal{L}_{wa}$(rcANIEP) *and* $\mathcal{L}_{sa}$(rcANIEP) *are closed under union.*

3.4 Networks with Non-insertion Nodes

Networks with substitution and deletion nodes only have been studied by J. Dassow and V. Mitrana in [7]. Here, we recall only two results.

Theorem 6. ([7])

1. $\mathcal{L}_{wa}$(rcANNIEP) ⊆ $\mathcal{L}_{sa}$(rcANNIEP) ⊆ $\mathcal{L}(CS)$.
2. The class $\mathcal{L}_{wa}$(rcANNIEP) *contains all linear context-free languages and non-semi-linear languages.*

3.5 Networks with Non-substitution Nodes

Similar to Theorem 1 we can prove that $\mathcal{L}_{wa}(\text{rcANNSEP}) \subseteq \mathcal{L}_{sa}(\text{rcANNSEP})$ (instead of substitution rules, we use insertion rules in the output nodes). The addition of insertion nodes brings more computational power to rcANDEPs as well as the addition of deletion nodes brings more computational power to rcANIEPs. The proofs of the next two results are skipped by limited space reason.

Proposition 3. *For every $k \geq 1$, the language $L_k = \{a^n \mid 1 \leq n \leq k\}$ belongs to $\mathcal{L}_{wa}(\text{rcANNSEP})$.*

Two networks Γ_1, Γ_2 can be combined with additional nodes to a network Γ such that a word is accepted by Γ if and only if it is accepted by Γ_1 or Γ_2.

Theorem 7. *The class $\mathcal{L}_{wa}(\text{rcANNSEP})$ is closed under union.*

3.6 Networks with Non-deletion Nodes

We can show the following result by the same proof as for Theorem 1 since the constructed network differs from the original one only in substitution nodes.

Theorem 8. $\mathcal{L}_{wa}(\text{rcANNDEP}) \subseteq \mathcal{L}_{sa}(\text{rcANNDEP})$.

Analogously to Theorem 2, we can show

Theorem 9. *The class $\mathcal{L}_{wa}(\text{rcANNDEP})$ is closed under union.*

Let $A = \{a\}$ be a unary input alphabet. Let $\mathcal{I}$, $\mathcal{D}$, $\mathcal{S}$, $\mathcal{NI}$, $\mathcal{ND}$, and $\mathcal{NS}$ be the set of all languages weakly accepted by an insertion network over the alphabet A, by a deletion network, by a substitution network, by a non-inserting network, by a non-deleting network, and by a non-substituting network, respectively.

By the Theorems 5 and 4, Corollary 1, Proposition 3, we obtain the inclusions shown in Figure 1.

A (solid) arrow stands for (proper) inclusion. The sets $\mathcal{NI}$ and $\mathcal{NS}$ are not necessarily uncomparable.

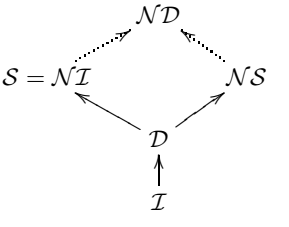


Fig. 1. Hierarchy

4 Computational Power of Regular Networks

4.1 Networks with Non-insertion Nodes

In [7], it was shown that regular networks with only substitution and deletion nodes always accept context-sensitive languages and that every context-sensitive language can be accepted by such a network where the number of nodes depends linearly in the number of rules necessary for generating the language. Here, we show that one substitution node alone is sufficient.

Theorem 10. *For any context-sensitive language L, there is an accepting regular network Γ with exactly one substitution node and one output node without rules that weakly and strongly accepts the language L.*

Proof. Let L be a context-sensitive language and $G = (N, T, P, S)$ be a grammar in Kuroda normal form with $L(G) = L$. Let $R_1, R_2, \ldots, R_8$ be the following sets (A, B, C, D denote non-terminal symbols):

$$
\begin{aligned}
R_1 &= \{\, x \to x_{p,0},\ x_{p,0} \to A \mid A \to x \in P,\ A \in N,\ x \in N \cup T \,\},\\
R_2 &= \{\, C \to C_{p,1} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P \,\},\\
R_3 &= \{\, D \to D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P \,\},\\
R_4 &= \{\, C_{p,1} \to C_{p,3} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P \,\},\\
R_5 &= \{\, D_{p,2} \to D_{p,4} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P \,\},\\
R_6 &= \{\, C_{p,3} \to A \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P \,\},\\
R_7 &= \{\, D_{p,4} \to B \mid p = AB \to CD \in P \,\},\\
R_8 &= \{\, D_{p,4} \to \_ \mid p = A \to CD \in P \,\}.
\end{aligned}
$$

We construct a network Γ of with the input alphabet T, the network alphabet

$$V = N \cup T \cup \{\_\} \bigcup_{p=A\to x} \{\, x_{p,0} \,\} \cup \bigcup_{\substack{p=A\to CD\\ p=AB\to CD}} \{\, C_{p,1}, D_{p,2}, C_{p,3}, D_{p,4} \,\}$$

and two nodes. The input node is $(M_1, \emptyset, O_1)$ with

$$
\begin{aligned}
M_1 &= R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6 \cup R_7 \cup R_8, \text{ and}\\
O_1 &= \{\_\}^*\{S\}\{\_\}^* \cup \{\varepsilon\} \cup V^* \setminus ((N \cup T \cup \{\_\})^* \overline{O} (N \cup T \cup \{\_\})^*),
\end{aligned}
$$

where

$$
\begin{aligned}
\overline{O} = {} & \{\, \varepsilon \,\} \cup \{\, x_{p,0} \mid p = A \to x \in P,\ A \in N,\ x \in N \cup T \,\}\\
& \cup \{\, C_{p,1} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P \,\}\\
& \cup \{\, C_{p,1}\omega D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \omega \in \{\_\}^* \,\}\\
& \cup \{\, C_{p,3}\omega D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \omega \in \{\_\}^* \,\}\\
& \cup \{\, C_{p,3}\omega D_{p,4} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \omega \in \{\_\}^* \,\}\\
& \cup \{\, A\omega D_{p,4} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \omega \in \{\_\}^* \,\}.
\end{aligned}
$$

The output node is $(\emptyset, I_2, \emptyset)$ with $I_2 = \begin{cases} \{\_\}^*\{S\}\{\_\}^* \cup \{\varepsilon\} & \text{if } \varepsilon \in L,\\ \{\_\}^*\{S\}\{\_\}^* & \text{otherwise.} \end{cases}$

The network Γ has only one output node. Therefore, there is no difference between weak and strong acceptance.

By a case distinction on the rules of G, one can prove that any derivation $w \Longrightarrow^* v$ with $v \neq \varepsilon$ of the grammar G can be simulated by the network Γ in reverse direction (by a reduction $v \Longrightarrow^* w$) and, on the other hand, a word $v \in N \cup T$ can only be transformed into a word $w \in N \cup T$ if the derivation $w \Longrightarrow^* v$ exists in G. Since a substitution node cannot delete letters, we replace letters by the symbol _ instead. When simulating a derivation $w \Longrightarrow v$ in G, we have to take into account that the corresponding word in the substitution node may contain gaps in form of several occurrences of the special symbol _. □

4.2 Networks of Non-deleting Nodes

The main difference between context-sensitive and non-context-sensitive grammars is that, in arbitrary phrase structure grammars, erasing rules are allowed. In order to simulate an erasing rule in reverse direction, we introduce an insertion node.

Theorem 11. *For any recursively enumerable language L, there is an accepting network of evolutionary processors with exactly one substitution node, one insertion node and one output node without rules that weakly and strongly accepts the language L.*

Proof. The idea of the proof is to extend the network Γ constructed in the proof of Theorem 10 by an inserting processor who is responsible for the reverse simulation of erasing rules. Between two simulation phases, the substitution node can mark a symbol such that the word can leave the node and enter the insertion node. This processor inserts a non-terminal that belongs to an erasing rule and returns the word to the substitution node. This processor then has to unmark the primed symbol. If marking or unmarking is not performed in the correct moment, the word will be lost. □

4.3 Networks without Substitution Processors

In [8], we have shown that every recursively enumerable language can be generated by a network of one inserting processor and one deleting processor. Similar to the proof of that statement, we can prove the following result.

Theorem 12. *For any recursively enumerable language L, there is an accepting network of evolutionary processors with exactly one deletion node, one insertion node and one output node without rules that weakly and strongly accepts the language L.*

Acknowledgements. We thank Jürgen Dassow for stimulating discussions and ideas on this topic.

References

1. Hillis, W.: The Connection Machine. MIT Press, Cambridge (1985)
2. Fahlman, S., Hinton, G., Seijnowski, T.: Massively parallel architectures for AI: NETL, THISTLE and Boltzmann Machines. In: Proc. AAAI National Conf. on AI, pp. 109–113. Morgan Kaufmann, Los Altos (1983)
3. 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)
4. Castellanos, J., Martín-Vide, C., Mitrana, M., Sempere, J.: 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. Margenstern, M., Mitrana, V., Perez-Jimenez, M.: Accepting hybrid networks of evolutionary processors. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 235–246. Springer, Heidelberg (2005)
6. Manea, F., Mitrana, V.: All NP-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size. Information Processing Letters, 112–118 (2007)
7. Dassow, J., Mitrana, V.: Accepting networks of non-inserting evolutionary processors. In: Petre, I., Rozenberg, G. (eds.) Proceedings of NCGT 2008 – Workshop on Natural Computing and Graph Transformations, Leicester, United Kingdom, September 8, 2008, pp. 29–41. University of Leicester (2008)
8. Alhazov, A., Dassow, J., Martín-Vide, C., Rogozhin, Y., Truthe, B.: On networks of evolutionary processors with nodes of two types. Fundamenta Informaticae (in press, 2009)
9. Martín-Vide, C., Mitrana, V.: Networks of evolutionary processors: results and perspectives. In: Molecular Computational Models: Unconventional Approaches, pp. 78–114. Idea Group Publishing, Hershey (2005)

The Halting Problem and Undecidability of Document Generation under Access Control for Tree Updates

Neil Moore

University of Kentucky, Department of Computer Science
Lexington, Kentucky 40506-0046
neil@cs.uky.edu

Abstract. We show by reduction from the halting problem for Turing machines that typical rule-based models of fine-grained access control of trees make impossible certain forms of analysis, limiting the ability to audit existing policies and evaluate new ones. Fine-grained access control is the problem of specifying the set of operations that may be performed on a complex structure. For tree-structured databases and documents, particularly XML, a rule-based approach is most common. In this model, access control policies consist of rules that select the allowed or disallowed targets of queries based on their hierarchical relationships to other nodes.

We consider the problem of determining whether a given document (that is, a rooted vertex-labelled tree) could have been produced in accordance with a particular access control policy for updates. We show that, for rule-based policies based on a simple fragment of XPath, this problem is undecidable. This result shows that rule-based access control policies based on XPath languages are, in some sense, *too* powerful, demonstrating the need for a model of access control of tree updates that bridges the gap between expressive and analyzable policies.

1 Motivation and Related Work

In the past few years the topic of fine-grained access control (FGAC) for hierarchical structures has been heavily studied, particularly in the context of Extensible Markup Language (XML) [1] documents. The problem is to specify and enforce an access control policy that determines which portions of a document or collection of documents may be queried or modified by particular users.

A rule-based approach is prevalent in the literature [2,3,4]: in this model, an **access control policy** comprises a collection of **rules**. Each rule permits or denies a user, group, or role (the **subject**) to use a particular kind of operation on certain parts of the document (the **object**). Objects are typically nodes or subtrees of the document, specified with a language of **path expressions**, usually XPath [5] or some fragment thereof [6,7].

While most of the early research on FGAC for XML focused on query operations [2,3,8], there has more recently been work in applying FGAC to update operations [9,10,11,12]. Two problems have been particularly well-studied:

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 601–613, 2009.

specifying and formalizing the semantics of FGAC policies [4,12,13,14], and safely and efficiently enforcing these policies [2,11]. We consider another problem, related to the analysis of policies: to determine a posteriori whether a document could have been constructed under a given access control policy. This problem, which we call PolicyGenerates, is motivated by a number of database administration and collaborative editing scenarios. For example, an administrator might wish to audit a document by verifying that it was (or at least could have been) created under the existing policy; this is particularly important when logs of previous operations are missing or unavailable. Another application is to evaluate a proposed policy change, by verifying that existing documents could be reconstructed under the policy. Finally, an algorithm for PolicyGenerates would allow administrators to verify that particular undesirable document states are disallowed: that is, that the policy does not generate particular trees. We will show in Section 3 that this problem is undecidable.

2 Definitions

We begin with a collection of definitions. We use the term "tree" to refer to a rooted unordered tree, with nodes bearing a labels from some given set $\mathcal{L}$.

Definition 1. *An **operation** on a tree T is a tuple having the form* $(\mathsf{insert}, \mu, \ell)$, $(\mathsf{rename}, \nu, \ell)$, *or* (delete, ν), *where ν is a node of T, μ is either ε or a node of T, and ℓ is a label. The second member of an operation (μ or ν) is called its **target**.*

Definition 2. *The result $O(T)$ of an operation O on tree T is:*

- *If* $O = (\mathsf{insert}, \nu, \ell)$, *the tree obtained from T by adding a single additional node labelled ℓ as a child of ν.*
- *If* $O = (\mathsf{insert}, \varepsilon, \ell)$ *and T is empty, the tree with a single node labelled ℓ.*
- *If* $O = (\mathsf{insert}, \varepsilon, \ell)$ *and T is not empty, T.*
- *If* $O = (\mathsf{rename}, \nu, \ell)$, *the tree obtained from T by giving node ν the label ℓ.*
- *If* $O = (\mathsf{delete}, \nu)$, *the tree obtained by removing the subtree rooted at ν.*

The result of a sequence of operations $\langle o_1, \ldots, o_n \rangle$ on a tree T is the tree $o_n(o_{n-1}(\cdots o_1(T) \cdots))$.

Remark 1. Our operations are loosely based on those of XUpdate [15]. The rename and delete operations are unchanged from their counterparts in XUpdate. Insert serves as an order-insensitive version of the XUpdate operations InsertBefore, InsertAfter, and Append. Furthermore, unlike in XUpdate, insert adds only a single node at a time; more complicated insertions can be accomplished by a sequence of operations We do not include an replace operation, as its effects can be duplicated by a sequence of insert and delete operations.

Rules in FGAC policies identify potential targets of operations by means of **path expressions**. Typically, path expressions are expressed in XPath [5] or

some XPath fragment such as Core XPath or XPattern [16]. Our rules will use the $\mathrm{XP}^{\{[],*,//\}}$ fragment of XPath, containing predicates, path expressions, and the child and descendant axes. This fragment of XPath has been particularly well studied [6,7].

Briefly, a path expression consists of a sequence of location steps that select nodes along the path from the root to the target according to their labels. If a location step is preceded by /, it selects (appropriately-labelled) children of the preceding node; if preceded by //, it selects descendants of that node. A location step may be followed by a number of **predicates**, each a path expression surrounded by square brackets []; the location step selects a node only if each predicate selects some descendant of that node. Some examples:

- /∗ selects the root node, regardless of its name.
- //m selects every node labelled m.
- /w/x//y selects every y descendant of an x child of the root w node.
- //x[∗/q]/z selects every z node that is the child of some x node that has a grandchild q.

Definition 3. *An **access control rule** is a tuple with the form* $(s, \mathsf{insert}, P, L)$, (s, delete, P), *or* $(s, \mathsf{rename}, P, L)$, *where: s is either $+$ (**positive**) or $-$ (**negative**); P is a path expression (possibly the empty path expression ε); and L is either a label or the symbol $*$, matching all labels.*

Definition 4. *A rule R **matches** the operation O on tree T, written $R \sim_T O$, if and only if one of the following holds:*

- $R = (s, \mathsf{insert}, \varepsilon, L)$, $O = (\mathsf{insert}, \varepsilon, \ell)$, *$T$ is empty, and either $L = \ell$ or $L = *$;*
- $R = (s, \mathsf{insert}, P, L)$, $O = (\mathsf{insert}, \nu, \ell)$, *$P$ selects ν in T, and L is ℓ or $*$;*
- $R = (s, \mathsf{rename}, P, L)$, $O = (\mathsf{rename}, \nu, \ell)$, *$P$ selects ν in T, and L is ℓ or $*$; or*
- $R = (s, \mathsf{delete}, P)$, $O = (\mathsf{delete}, \nu)$, *and P selects ν in T.*

Definition 5. *A positive rule R with path expression P is **active** on tree T if it matches some possible operation on T: that is, if $P(T)$ is not the empty set; or if T is the empty tree, $P = \varepsilon$, and R is an* insert *rule.*

Definition 6. *An access control **policy** is a finite set of access control rules. The policy $\mathcal{P}$ **permits** the operation O on tree T, written $\mathcal{P} \vdash_T O$, if there exists some positive rule $R \in \mathcal{P}$ such that $R \sim_T O$ and there does not exist a negative rule $R_- \in \mathcal{P}$ such that $R_- \sim_T O$.*

If $\mathcal{S} = \langle o_1, \ldots, o_n \rangle$ is a finite sequence of operations, we say that $\mathcal{P}$ permits $\mathcal{S}$ on tree T ($\mathcal{P} \vdash_T \mathcal{S}$) if $\mathcal{P} \vdash_{T_{i-1}} o_i$ for each $1 \leq i \leq n$, where $T_0 = T$ and $T_i = o_i(T_{i-1})$.

Remark 2. Our model uses "deny overwrites" and "default deny" conflict resolution [4,14]: if an operation is matched by both positive and negative rules, or is matched by no rule, it is not permitted.

Remark 3. In order to facilitate analysis of FGAC rules and policies, we have made some simplifying assumptions. First, we disregard subjects in our model of policies. Even when multiple users are present, many important questions can be expressed in terms of subject-less policies, either by considering only rules governing a particular subject, or by considering all rules regardless of subject. Secondly, we treat documents and databases as unordered trees. Our results may be easily extended to policies containing multiple subjects, and to ordered trees.

With these preliminary definitions out of the way, we may now formally define the problem PolicyGenerates.

Definition 7. *A policy $\mathcal{P}$ **generates** the tree T if there exists some finite sequence of operations $\mathcal{S}$ such that $\mathcal{P} \vdash_{T_0} \mathcal{S}$ and $\mathcal{S}(T_0) = T$, where T_0 is the empty tree. The problem* PolicyGenerates *is the set of pairs $(\mathcal{P}, T)$ such that $\mathcal{P}$ generates T.*

3 Undecidability of PolicyGenerates

We will prove that, in our model of FGAC policies for tree updates, PolicyGenerates is undecidable. Our proof proceeds by reduction from the halting problem for Turing machines. We will show how to encode a Turing machine M and initial tape S as an access control policy $\mathcal{P}_{M,S}$ such that $\mathcal{P}_{M,S}$ generates a particular tree T_{halt} if and only if M halts on input S.

We represent a Turing machine as a 7-tuple of a set of states, an initial state, a set of final (accepting) states, a tape alphabet, an input alphabet, a blank symbol, and a transition function.

$$M = (Q, q_0 \in Q, Q_F \subseteq Q, \Gamma, \Sigma \subset \Gamma, b \in \Gamma \setminus \Sigma, \delta)$$

We consider deterministic Turing machines with left and right moves only, where no transitions are permitted from final states: that is, the transition function δ maps from $(Q \setminus Q_F) \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$. Furthermore, we assume the tape is bounded on the left.

A **configuration** is a tuple (q, t, p) of a state $q \in Q$, a tape $t \in \Gamma^*$, and a tape position $1 \leq p \leq |t|$. We write $\mathcal{C}_0^{M,S}$ for the initial configuration of Turing machine M on input S, namely $(q_0, S, 1)$. We write Δ for the function taking a configuration to the succeeding configuration.

3.1 Modelling Turing Configurations as Trees

A tree generated by $\mathcal{P}_{M,S}$ will represent a configuration of M, as well as additional bookkeeping information necessary to the simulation.

Definition 8. *Let $\mathcal{C} = (q, t, p)$ be a configuration of the Turing machine $M = (Q, q_0, Q_F, \Gamma, b, \Sigma, \delta)$. The **configuration tree** of $\mathcal{C}$,* conftree($\mathcal{C}$)*, is the tree described in Figure 1.*

- a root node labelled tm;
- three children, labelled ph, state, and tape;
- one child of the ph node, labelled run;
- one child of the state node, labelled q;
- one child C_1 of the tape node, labelled cell;
- each cell node C_i $(1 \leq i \leq |t|)$ having:
  - one child labelled sym, itself having a child labelled with the tape symbol $t[i]$;
  - if $i < |t|$, one child C_{i+1} labelled cell;
  - if $i = p$, one child labelled curr.

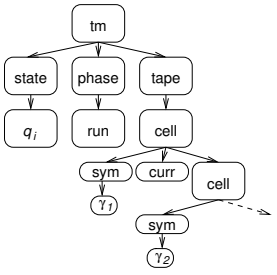


Fig. 1. A configuration tree

3.2 The Policy $\mathcal{P}_{M,S}$

Let $M = (Q, q_0, Q_F, \Gamma, \Sigma, b, \delta)$ be a Turing machine, and $S = S_1 S_2 \dots S_n \in \Sigma^*$ an initial tape for M. The policy $\mathcal{P}_{M,S}$ will simulate the action of M on input S. It consists of the following rules and (finite) rule schemata.

$(-,$ insert, /tm[ph], ph$)$ (1)
$(-,$ insert, /tm[state], state$)$ (2)
$(-,$ insert, /tm[tape], tape$)$ (3)
$(-,$ insert, /tm/ph[$*$], $*)$ (4)
$(-,$ insert, /tm/state[$*$], $*)$ (5)
$(-,$ insert, //$*$[cell], cell$)$ (6)
$(-,$ insert, //cell[curr], curr$)$ (7)
$(-,$ insert, //cell[new], new$)$ (8)
$(-,$ insert, //cell[sym], sym$)$ (9)
$(-,$ insert, //sym[$*$], $*)$ (10)
$(+,$ insert, ε, tm$)$ (11)
$(+,$ insert, /tm, ph$)$ (12)
$(+,$ insert, /tm/ph, build$)$ (13)
$(+,$ insert, /tm[ph/build], state$)$ (14)
$(+,$ insert, /tm[ph/build], tape$)$ (15)
$(+,$ insert, /tm[ph/build]/state, $q_0)$ (16)
$(+,$ insert, /tm[ph/build]/tape, cell$)$ (17)
$(+,$ insert, /tm[ph/build]/tape/cell, curr$)$ (18)
$(+,$ insert, /tm[ph/build]/tape//cell, sym$)$ (19)
$(+,$ insert, /tm[ph/build]/tape/cell/sym, $S_1)$ (20)

For each i, $1 < i \leq |S|$, let "$\cdots$" stand for $i-1$ repetitions of the path fragment "/cell":

$(+,$ insert, /tm[ph/build]/tape$\cdots$, cell$)$ (21)
$(+,$ insert, /tm[ph/build]/tape$\cdots$[sym/$*$]/cell/sym, $S_i)$ (22)

And a single rule where "$\cdots$" represents $|S|-1$ repetitions of the path fragment /cell:

$(+,$ rename, /tm[state/q_0][tape/cell[curr]$\cdots$/sym/$S_{|S|}$]/ph/build, run$)$ (23)

For each final state $q_f \in Q_F$:

$(+,$ insert, /tm[ph/run][state/q_f], finished$)$ (24)

For $(q, \gamma) \in (Q \setminus Q_F) \times \Gamma$, where $\delta(q, \gamma) = (q', \gamma', D)$:

$(+,$ rename, /tm[state/q][tape//cell[curr][sym/γ]]/ph/run, write-q-$\gamma)$ (25)
$(+,$ rename, /tm[ph/write-q-γ]/tape//cell[curr]/sym/γ, $\gamma')$ (26)
$(+,$ rename, /tm[ph/write-q-γ]/state/q, $q')$ (27)
$(+,$ rename, /tm[state/q'][tape//cell[curr]/sym/γ']ph/write-q-γ, move-q-$\gamma)$ (28)
$(+,$ delete, /tm[ph/move-q-γ][tape//new]/tape//curr$)$ (29)
$(+,$ rename, /tm[tape//cell/new]/ph/move-q-γ, cleanup$)$ (30)
$(-,$ rename, /tm[tape//cell/curr]/ph/move-q-γ, $*)$ (31)

If $D = L$:

$$(+, \mathsf{insert}, \mathsf{/tm[ph/move}\text{-}q\text{-}\gamma\mathsf{]/tape//cell[cell/curr]}, \mathsf{new}) \tag{32}$$

Otherwise, $D = R$:

$$(+, \mathsf{insert}, \mathsf{/tm[ph/move}\text{-}q\text{-}\gamma\mathsf{]/tape//cell[curr]}, \mathsf{cell}) \tag{33}$$

$$(+, \mathsf{insert}, \mathsf{/tm[ph/move}\text{-}q\text{-}\gamma\mathsf{]/tape//cell}, \mathsf{sym}) \tag{34}$$

$$(+, \mathsf{insert}, \mathsf{/tm[ph/move}\text{-}q\text{-}\gamma\mathsf{]/tape//cell[curr]/cell/sym}, b) \tag{35}$$

$$(+, \mathsf{insert}, \mathsf{/tm[ph/move}\text{-}q\text{-}\gamma\mathsf{]/tape//cell[curr]/cell[sym/}*\mathsf{]}, \mathsf{new}) \tag{36}$$

Finally:

$$(+, \mathsf{rename}, \mathsf{/tm[ph/cleanup]//new}, \mathsf{curr}) \tag{37}$$

$$(+, \mathsf{rename}, \mathsf{/tm[tape//curr]/ph/cleanup}, \mathsf{run}) \tag{38}$$

$$(+, \mathsf{delete}, \mathsf{/tm[finished]//}*) \tag{39}$$

3.3 Simulation Phases

Our Turing machine simulation proceeds from one configuration tree to the next by proceeding through a number of **phases**. The tree's current phase is indicated by the label of the child of the ph node. In each phase, a particular sequence of operations is permitted, ending with a transition into the successor phase. There are three independent phases build, run, and cleanup; and $2|Q \setminus Q_F| \cdot |\Gamma|$ phases that depend on the simulated machine's state q and the contents γ of its current tape cell; these phases are labelled write-q-γ and move-q-γ.

We begin by noting that the negative Rules 1–10 enforce certain constraints on the tree. These **structural rules** ensure that there is only one copy of each top-level node (ph, state, and tape); that the state and ph nodes have only a single child each; that no tape cell contains two symbols or two successor tape cells; and that no cell is marked as "current" or "new" twice.

At the beginning of our simulation, the tree is empty, and hence does not have a phase. In this stage, we construct enough of the tree to enter the build phase. Because every other positive rule in $\mathcal{P}_{M,S}$ requires the existence of a child of the /tm/ph node, only Rules 11–13 are active at this point. We therefore have the following:

Lemma 1. *Any sufficiently long sequence of operations that is permitted on the null tree must pass through an intermediate tree containing exactly: a* tm *root, one* ph *child of the root, and one* build *child of that node.*

Remark 4. Because an operation that would insert a root node is not permitted on a non-empty tree, Rule 11 can be used only once. Likewise, structural Rules 1 and 4 prevent Rules 12 and 13 from being activated again as long as the tree contains a ph node. Hence we shall disregard Rules 11–13 in the following lemmas.

Build Phase. In the build phase, we complete the initial configuration tree, including the state and tape nodes and their descendants. At the end of this phase, once the entire initial configuration tree is constructed, we permit transitioning to the run phase.

Lemma 2. *Any sufficiently long sequence of operations permitted on the null tree must produce the tree* conftree($\mathcal{C}_0^{M,S}$) *as an intermediate step.*

Proof. Lemma 1 established that such a sequence of operations must produce as an intermediate step a tree with a /tm/ph/build node, ph and tm ancestors, and no other nodes. On such a tree, only Rules 14 and 15 permit further operations.

Once a state node has been inserted by Rule 14, Rule 16 permits inserting a q_0 child; Rules 2 and 5 ensure that at most one state node and one child of that node are inserted.

After applying Rule 15 to insert a tape node, Rules 17–20 allow inserting a cell child; curr and sym grandchildren; and a S_1 child of the sym node. Once the cell node has been inserted, the instances of Rule 21 allow inserting $i - 1$ cell descendants, each a child of the previous node. Rule 6 ensures that neither the tape node nor the cell nodes may contain multiple cell children; and Rule 7 prevents applying Rule 18 again.

For each of the cell nodes, Rules 19 and 22 allow inserting sym children, and grandchildren labelled with the appropriate symbol. Furthermore, the [sym/$*$] predicate in the latter rule ensures that no symbol is inserted until the previous symbol (and hence all preceding symbols) are inserted.

Finally, Rule 23 allows renaming the build node to run, but only after the q_0 node and last cell's symbol (and hence all $|S|$ symbols) have been inserted.

Since all other positive rules contain predicates which require that the ph node contain some child other than build, these are the only rules that permit operations before Rule 23 is applied. Along with the structural rules, this ensures that any sequence of $3 + 2 + 1 + 4 + 3|S - 1| = 3|S| + 7$ operations must result in a tree where Rule 23 is the only applicable positive rule that is not blocked by negative rules. Applying this rule results in precisely the tree conftree($\mathcal{C}_0^{M,S}$). Hence any permitted sequence of length $3|S| + 8$ must yield this tree, and any longer permitted sequence will produce this tree as an intermediate result. □

Run Phase. The run phase represents a Turing machine configuration. This phase can be entered from the build phase (Rule 23), or from the cleanup phase (Rule 38). The only operations permitted in this phase are to rename the run node to the next phase, one of the write phases; to insert a finished node if the configuration is a final one; and to delete most of the tree when the finished node is present.

Lemma 3. *If $\mathcal{C}$ is not a final configuration (that is, its state is not a member of Q_F), then exactly one operation is permitted on* conftree($\mathcal{C}$)*, resulting in a tree T_w, otherwise identical to* conftree($\mathcal{C}$)*, but with a* write-q-γ *node replacing the* run *node, where q is the state of $\mathcal{C}$ and γ is the symbol in the current tape cell of $\mathcal{C}$.*

Proof. Other than Rules 11–13, which are blocked by structural rules, the only positive rules whose predicates are satisfied by a tree in the run phase without a

finished node are instances of Rule schemata 25 and 24; only the former is active for a non-final configuration. This rule transforms the tree into precisely T_w. □

Write Phases. For each transition, that is to say each pair $(q, \gamma) \in (Q \setminus Q_F) \times \Gamma$, the phase write-$q$-$\gamma$ allows changing the configuration tree's state and current tape symbol. It is followed by the move-q-γ and cleanup phases, where the position of the tape head is adjusted.

Lemma 4. *Let $\mathcal{C}$ be a Turing machine configuration with state q and current symbol γ; let T_w be the tree resulting from Lemma 3; and let $(q', \gamma', D) = \delta(q, \gamma)$. Any permitted sequence of operations on T_w that results in a tree without a* write-q-γ *node must as an intermediate step pass through a tree T_m that differs from T_w in that: the child of the* ph *node is labelled* move-q-γ*; the child of the* state *node is labelled q'; and the child node of the* sym *sibling of the* curr *node is labelled γ'.*

Proof. In this phase, only Rules 26–28 are active. Only the last of these rules permits renaming the write-q-γ node. This rule's predicates allow it to be used only if the state node has a child labelled q' and the current tape cell node ζ has a grandchild labelled γ'. These are precisely the state and tape value specified by $\delta(q, \gamma)$.

Rule 27 permits renaming the state node q to q'; and Rule 26 permits renaming the γ grandchild of ζ to γ'; since these are the only three active rules, no other operations are permitted in this phase. Hence, immediately after applying Rule 28, the tree is precisely the described T_m. □

Remark 5. Unlike in the other phases, it is not the case that any sufficiently long sequence of operations leads to the next phase. If a cell is being re-written with the same symbol, or the transition does not change the Turing machine state, is possible to apply Rule 26 or 27 an arbitrary number of times. However, since in this case the rule in question does not alter the tree, the result does not differ from applying the rule only once.

Move Phases. When the move-q-γ phase is entered, the tree's state and current tape symbol have been updated. It remains to move the tape head, adding a new cell if necessary. This is accomplished in two phases: in the move phase, we mark the updated tape head position as "new" and remove the "current" mark. Then, in the cleanup phase, we change the "new" mark to "current".

Lemma 5. *Let $\mathcal{C} = (q, t, p)$ be a Turing machine configuration; let T_m be the tree resulting from Lemma 4; and let D be the directional component of $\delta(q, \gamma)$.*

If the $p = 1$ and $D = L$ (a ***hanging configuration****), no operations are permitted on T_m. Otherwise, let ζ be the* cell *node of T_m containing a* curr *child. Any sufficiently long sequence of permitted operations of T_m must as an intermediate step pass through the tree T_c that is otherwise identical to T_m, but without a* curr *node, with the* move-q-γ *node replaced with* cleanup *node, and with changes depending on D and p:*

- *If $D = L$ and $p > 1$, then the parent of ζ in T_c contains a child labelled* new.
- *If $D = R$ and $p = |t|$, then ζ in T_c contains a new* cell *child, containing two children labelled* new *and* sym*, with the latter having a child node with the blank cell label b.*
- *Otherwise, $D = R$ and $p < |t|$; then the* cell *child of ζ in T_c contains a child labelled* new.

Proof. The positive rules active in this phase are instances of Rules 29 and 30; if $D = L$, Rule 32; and if $D = R$, Rules 33–36. Furthermore, the negative Rules 6, 8, 9, 10, and 31 are relevant to the operation of this phase. Rules 29 and 30 are not active on T_m, because the tree contains no new node.

We consider separately the two directions in which the tape head may move. If $D = L$, only Rule 32 is active on T_m. This rule permits inserting a new node as a child of the parent cell node of ζ. If $p = 1$, the parent of ζ is the tape node; hence no operations are permitted. Otherwise, after inserting the new node, Rule 8 prohibits inserting another such node; and Rule 29 becomes active. Once this rule is used to delete the curr node, Rule 31 is no longer active, leaving Rule 30 as the only active rule; applying this rule renames the move-q-γ node to cleanup, yielding T_c. Hence, any permitted sequence of three operations must yield T_c.

If $D = R$, there are two cases to consider. If $p < |t|$, only Rules 33 and Rule 36 are active; and because ζ has a cell child, Rule 6 prohibits inserting a node with the former rule. Hence the only permitted operation is to insert a new child of the cell child of ζ. After this, the argument from the $D = L$ case applies: the next operation must be to delete the curr node, then to rename the move-q-γ node. Hence any permitted sequence of three operations must yield T_c.

If $p = |t|$, Rule 36 is not active, because ζ has no cell children. Thus the only permitted operation is to insert a new cell child of ζ by Rule 33; Rule 6 prohibits inserting more than one such node. After this node has been inserted, Rule 34 permits inserting a sym node (but only once because of Rule 9). Then Rule 35 allows inserting a blank symbol node b as a child of this node, but only once because of Rule 10. Once these three operations have been performed, Rule 36 is active and the other rules inactive; then we may proceed as in the $p < |t|$ case. Hence any permitted sequence of six operations must yield T_c. □

Cleanup Phase. After the write and move phases, the tree has almost completely been updated to reflect the new Turing machine configuration. The only remaining steps are to replace the new marker node with a curr node, and to enter the run phase again, yielding the tree for the successor configuration.

Lemma 6. *Let $\mathcal{C}$ be a Turing machine configuration and let T_c be the tree resulting from applying Lemmas 3, 4, and 5 to* conftree($\mathcal{C}$)*. Any sufficiently long sequence of operations permitted on T_c must produce as an intermediate result the tree T', otherwise the same as T_c, but with a* curr *node in place of the* new *node, and a* run *node in place of the* cleanup *node.*

Proof. In the cleanup phase, positive Rules 37 and 38 are active; the latter is not active on T_c because it contains no curr node. Hence the only permitted

operation is to rename the new node to curr by the former rule. Performing this operation deactivates Rule 37, because there is no longer a new node. Hence the second operation must be to rename the cleanup node to run by Rule 38, resulting in the desired tree T'. Thus any permitted sequence of two operations on T_c yields T'. □

Halting. Finally, if the simulation reaches a final configuration, a finished node is inserted, and the tree is pared down to just the tm and finished nodes.

Lemma 7. *If $\mathcal{C} = (q_f, t, p)$ is a final configuration (that is, $q_f \in Q_F$), there is a sequence of operations permitted on $T = \text{conftree}(\mathcal{C})$ that yields the tree T_{halt} containing a root node labelled* tm*, a child node labelled* finished*, and no other nodes.*

Proof. The tree T contains nodes with paths /tm/ph/run and /tm/state/q_f; hence Rule 24 is active and it is permitted to insert a finished node. Once this node has been added, Rule 39 becomes active, allowing any non-root node in the tree to be deleted. In particular, it is permitted to delete the ph, state, and tape nodes, leaving precisely T_{halt}. □

3.4 Correctness of Simulation

Lemma 8. *Let $\mathcal{C}$ be a non-final Turing machine configuration, and let Q be any non-empty sequence of operations permitted on $T = \text{conftree}(\mathcal{C})$ such that $Q(\text{conftree}(\mathcal{C}))$ contains a node labelled* /tm/ph/run*. Then Q produces $\text{conftree}(\Delta(\mathcal{C}))$ as an intermediate result, representing the successor configuration $\Delta(\mathcal{C})$. Furthermore, if $\mathcal{C}$ is not a hanging configuration, such a sequence Q exists.*

Proof. After applying Lemmas 3 through 6 to the tree $T = \text{conftree}(\mathcal{C})$, we have a tree T' that is otherwise identical to T, except that: its state is q'; the symbol of the previous tape cell ζ (the cell that was current in $\text{conftree}(\mathcal{C})$) is γ'; the parent (if $D = L$) or child (if $D = R$) cell of ζ is marked as current; and if ζ had no cell children and $D = R$, a new child cell with symbol b has been inserted. These are precisely the changes necessary to transform $\text{conftree}(\mathcal{C})$ into $\text{conftree}(\Delta(\mathcal{C}))$. If any of these changes had not been performed, the resulting tree could not have a /tm/ph/run node. □

Figure 2 demonstrates the sequence of phases for a typical transition involving a rightwards move.

Theorem 1. *Let M be a Turing machine and $S \in \Sigma^*$ an initial tape of M; and let T_{halt} be the tree containing only a* tm *root and a* finished *child. Then $\mathcal{P}_{M,S}$ generates T_{halt} if and only if M halts on input S.*

Proof. By Lemmas 1 and 2, $\mathcal{P}_{M,S}$ generates the tree $T_1 = \text{conftree}(\mathcal{C}_0^{M,S})$, and any sufficiently long permitted sequence of operations must produce this tree as an intermediate step. If M halts on input S after k steps, then k applications

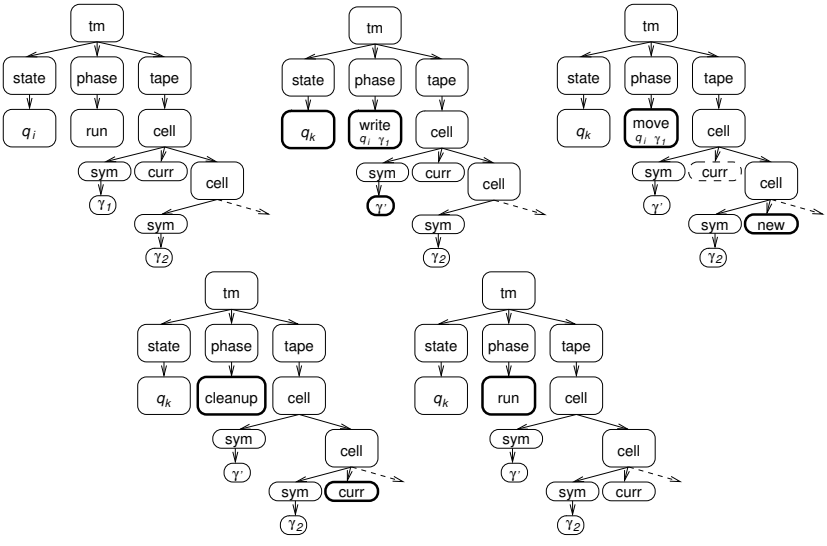


Fig. 2. Simulating the Turing machine transition $\delta(q_i, \gamma_1) = (q_k, \gamma', R)$. First row: a configuration tree in phase run, and the tree at the end of phases write, move. Second row: the tree at the end of phase cleanup, and the successor configuration tree.

of Lemma 8 produces a configuration tree for a final configuration. From such a tree, by Lemma 7 there is a permitted sequence of operations that yields T_{halt}.

Now suppose M does not halt on input S. Then there is an infinite sequence of configurations $\langle \mathcal{C}_0, \mathcal{C}_1, \ldots \rangle$ such that each $\mathcal{C}_{i+1} = \Delta(\mathcal{C}_i)$. Suppose $\mathcal{S}$ is a permitted sequence of operations resulting in T_{halt}. Then, as argued above, $\mathcal{S}$ must as an intermediate step produce a tree containing a run node and a $q_f \in Q_F$ node. By Lemma 8, this sequence must produce as an intermediate result the tree conftree($\mathcal{C}_1$). Repeated applications of Lemma 8 demonstrates that the sequence must subsequently produce intermediate results conftree($\mathcal{C}_2$), conftree($\mathcal{C}_3$), and in general conftree($\mathcal{C}_i$) for each $i \in \mathbb{N}$. However, this means that $\mathcal{S}$ is an infinite sequence, contradicting the assumption that it is a permitted sequence. Hence no such sequence exists, so $\mathcal{P}_{M,S}$ does not generate T_{halt}. □

Corollary 1. PolicyGenerates *is undecidable.*

Proof. Given a Turing machine M and input tape S, it is possible to construct the policy $\mathcal{P}_{M,S}$ algorithmically (in fact, in polynomial time). Combined with Theorem 1, this establishes a many-to-one reduction from the halting problem to PolicyGenerates. Since the halting problem is undecidable, so too is PolicyGenerates. □

4 Conclusions and Future Work

We have shown that, unfortunately, the most common forms of XML access control policies preclude certain kinds of analysis, because their expressiveness

makes PolicyGenerates undecidable. This points to the need for less expressive, but still practical and flexible, models of and languages for fine-grained access control of tree updates.

Restricting updates to only insert makes the problem decidable: each operation inserts a single node, so for a tree with n nodes we need check only sequences of operations of length n. However, access control policies that do not allow updates to existing data are useless for many practical purposes. Another change, restricting policies to only positive rules, might make PolicyGenerates decidable; this is a topic for future research. However, the promise of this approach is limited, as it may be shown by reduction from the true quantified boolean formula problem that this simplified subproblem is still PSpace-hard.

If we eliminate predicates in our path expressions, we have the fragment $XP^{\{*,//\}}$ of XPath [6]. This is a much simpler language, because nodes are selected based entirely on their sequence of ancestors. However, this path expression language is incapable of expressing important constraints such as uniqueness of a node or uniqueness of a node's children.

When the mechanical analysis and verifiability of policies is important, a different approach to fine-grained access control of tree updates may be necessary. Proposed models that warrant further investigation include multi-level security of trees [8] and schema-based access control [17].

Acknowledgments. I would like to thank my research advisor Dr. Jerzy W. Jaromczyk for very helpful advice, discussions, and support, without which this paper would not have been possible. I would also like to thank the reviewers, whose helpful comments led to numerous improvements in the final version of this paper.

References

1. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F., Cowan, J.: Extensible markup language (XML) 1.1. World Wide Web Consortium Recommendation (2004), http://www.w3.org/TR/2004/REC-xml11-20040204/
2. Bertino, E., Braun, M., Castano, S., Ferrari, E., Mesiti, M.: Author-X: A Java-based system for XML data protection. In: Proceedings IFIP TC11/WG11.3 Fourteenth Annual Working Conference on Database Security: Data and Application Security, Development and Directions, pp. 15–26 (2000)
3. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P.: Securing XML documents. In: Zaniolo, C., Grust, T., Scholl, M.H., Lockemann, P.C. (eds.) EDBT 2000. LNCS, vol. 1777, p. 121. Springer, Heidelberg (2000)
4. Hada, S., Kudo, M.: XML Access Control Language: Provisional authorization for XML documents. Technical Report, Tokyo Research Laboratory, IBM Research (2000), http://www.trl.ibm.com/projects/xml/xacl/xacl-spec.html
5. Clark, J., DeRose, S.: XML path language (XPath), version 1.0. World Wide Web Consortium Recommendation (1999), http://www.w3.org/TR/1999/REC-xpath-19991116
6. Deutsch, A., Tannen, V.: Containment of regular path expressions under integrity constraints. In: Knowledge Representation Meets Databases (2001)

7. Miklau, G., Suciu, D.: Containment and equivalence for a fragment of XPath. J. ACM 51(1), 2–45 (2004)
8. Cho, S., Amer-Yahia, S., Lakshmanan, L.V., Srivastava, D.: Optimizing the secure evaluation of twig queries. In: Proceedings 28th VLDB Conference (2002)
9. Lim, C.H., Park, S., Son, S.H.: Access control of XML documents considering update operations. In: Proceedings 2003 ACM Workshop on XML Security (2003)
10. Cautis, B., Abiteboul, S., Milo, T.: Reasoning about XML update constraints. In: PODS 2007: Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 195–204. ACM, New York (2007)
11. Damiani, E., Fansi, M., Gabillon, A., Marrara, S.: Securely updating XML. In: Apolloni, B., Howlett, R.J., Jain, L. (eds.) KES 2007, Part III. LNCS, vol. 4694, pp. 1098–1106. Springer, Heidelberg (2007)
12. Fundulaki, I., Maneth, S.: Formalizing XML access control for update operations. In: Lotz, V., Thuraisingham, B.M. (eds.) SACMAT, pp. 169–174. ACM, New York (2007)
13. Bertino, E., Catania, B., Ferrari, E., Perlasca, P.: A logical framework for reasoning about access control models. ACM Trans. Inf. Syst. Secur. 6(1), 71–127 (2003)
14. Fundulaki, I., Marx, M.: Specifying access control policies for XML documents with XPath. In: Proceedings 9th ACM Symposium on Access Control Models and Technologies, pp. 61–69 (2004)
15. Laux, A., Martin, L.: XUpdate—XML update language. Working Draft (2000), http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html
16. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. In: Proc. of the 28th International Conference on Very Large Data Bases (VLDB 2002) (2002)
17. Bravo, L., Cheney, J., Fundulaki, I.: ACCOn: checking consistency of XML write-access control policies. In: Kemper, A., Valduriez, P., Mouaddib, N., Teubner, J., Bouzeghoub, M., Markl, V., Amsaleg, L., Manolescu, I. (eds.) EDBT. ACM International Conference Proceeding Series, vol. 261, pp. 715–719. ACM, New York (2008)

Prediction of Creole Emergence in Spatial Language Dynamics

Makoto Nakamura[1], Takashi Hashimoto[2], and Satoshi Tojo[1]

[1] School of Information Science,
Japan Advanced Institute of Science and Technology
1-1, Asahidai, Nomi, Ishikawa, 923-1292, Japan
[2] School of Knowledge Science,
Japan Advanced Institute of Science and Technology
1-1, Asahidai, Nomi, Ishikawa, 923-1292, Japan
{mnakamur,hash,tojo}@jaist.ac.jp

Abstract. Creole is a new born language emerging in most cases where language contact takes place. Simulating behaviors that creole communities are formed in some environments, we could contribute to actual proof of some linguistic theories concerning language acquisition. Thus far, a simulation study of the emergence of creoles has been reported in the mathematical framework. In this paper we introduce a spatial structure to the framework. We show that local creole communities are organized, and creolization may occur when language learners learn often from non-parental language speakers, in contrast to the non-spatial model. The quantitative analysis of the result tells us that emergence of local colonies at the early stage tends to induce the full creolization.

1 Introduction

Computer simulation of diachronic change in human languages has widely been reported in the study of language evolution [1,2], where interactions among individuals affect language spoken throughout the community, dependent upon the abilities of individuals or the learning environment. Among those simulations, the emergence of pidgins and creoles is one of the most interesting phenomena in language change [3,4,5].

Pidgins are simplified tentative languages spoken in multilingual communities, which come into being where people need to communicate but do not have a language in common. On the other hand, creoles are full-fledged new languages based on the pidgins in later generations. For example, Hawaiian Creole English emerged among plantation workers coming from Hawaii, China, the Philippines, Japan, Korea, Portugal, Puerto Rico and so on in the 19th century to the beginning of the 20th century. Since they needed to communicate with farm owners, they first formed Hawaiian pidgin based on English; later their offspring immersed with the pidgin had developed the language to one with its own grammatical structure. In general, grammar of a creole is different from any contact languages, although its vocabulary is often borrowed from them. Our goal in this paper is to discover specific conditions under which creoles emerge.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 614–625, 2009.

Thus far, we proposed a mathematical framework for the emergence of creoles [6] based on the language dynamics equation by Nowak et al. [7], showing that creoles become dominant under specific conditions of similarity among languages and linguistic environment of language learners. Our purpose in the present study is to introduce a *spatial structure* to our model, in order to observe self-organization process of creole community. Especially, in this paper we compare behaviors of the two models. A related work for introducing a spatial structure into a mathematical model of language change has been done by Castelló et al. [8], who have analyzed a spatial version of a mathematical framework by Abrams et al. [9]. Different from Abrams-Strogatz's model, our model [6] is well-defined in terms of learning algorithm and a learning environment.

Introducing a spatial structure to a mathematical framework, we expect to observe a process of creolization and then to obtain more precise conditions from the model more similar to the environment where actual language phenomena took place. We recognize this study to fill the gap between the study of multi-agent models and mathematical models.

In Section 2, we describe the modified language dynamics model and a learning algorithm, and in Section 3 we define a creole in population dynamics. Section 4 reports our experiments, and we conclude in Section 5.

2 Population Dynamics for the Emergence of Creole

In this section, we briefly explain how to divert a mathematical model proposed by Nakamura et al. [6] to the one with a spatial structure.

The most remarkable point in the model of Nakamura et al. [6] is to introduce an *exposure ratio* α, which determines how often language learners are exposed to a variety of language speakers other than their parents. They modified the learning algorithm of Nowak et al. [7], taking the exposure ratio into account to model the emergence of creole community. Nakamura et al. [6] have shown that a certain range of α is necessary for a creole to emerge.

2.1 Language Dynamics Equation for the Emergence of Creole

In response to the language dynamics equation by Nowak et al. [7], Nakamura et al. [10] assumed that any language could be classified into one of a certain number (n) of grammars. Thus, the population of language speakers is distributed to $\{G_1 \dots G_n\}$. Let x_i be the proportion of speakers of G_i within the total population. Then, the language dynamics is modeled by an equation governing the transition of language population.

Because Nowak et al. [7] assumed that language speakers bore offspring in proportion to their successful communication, they embedded a fitness term in their model which determined the birth rate of each language group. The model for creolization has excluded the biological fitness, on the assumption that in the real world creoles did not emerge because creole speakers had more offspring than speakers of other pre-existing languages, that is:

$$\frac{dx_j(t)}{dt} = \sum_{i=1}^{n} \overline{q}_{ij}(t)x_i(t) - x_j(t) \ . \tag{1}$$

In the language dynamics equations, the similarity matrix S and the transition matrix $\overline{Q}(t)$ play important roles: the similarity matrix $S = \{s_{ij}\}$ is defined as a probability that a sentence of G_i is accepted also by G_j. Children learn a language in accordance with a learning algorithm, in which the accuracy varies depending on the similarity among languages. The transition matrix $\overline{Q}(t) = \{\overline{q}_{ij}(t)\}$ is defined as a probability that a child of G_i speaker acquires G_j, and is calculated based on the learning algorithm. Being different from the definition by Nowak et al. [7], the definition of $\overline{Q}(t)$ depends on the generation parameter t, as well as the S matrix and a learning algorithm.

2.2 Introducing Spatial Structure

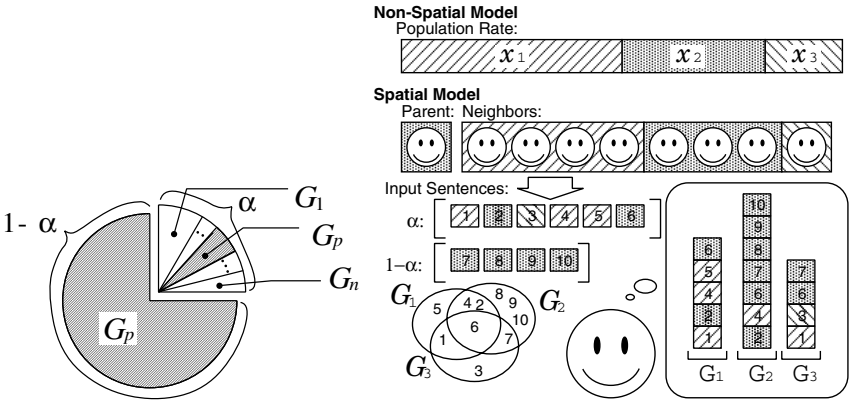
In the spatial model, we use the language distribution in neighbors surrounding each agent to calculate the local transition probability $\overline{Q}$, by which each agent acquires a language, while a child is exposed to the whole population in a non-spatial model.

Hereafter, replacing x_i for $x_i^{(l)}$ as a population rate of G_i speakers surrounding an agent at location l, equations are applicable to the spatial structure, too. We calculate $\overline{Q}^{(l)}(t)$ for each agent every generation.

2.3 Learning Algorithm

In some communities, a child learns language not only from his/her parents but also from other adults, whose language may be different from the parental one. In such a situation, the child is assumed to be exposed to other languages, and thus may acquire the grammar most efficient in accepting multiple language input. In order to assess how often the child is exposed to other languages, we divide the language input into two categories: one is from his/her parents, and the other is from other language speakers. We name the ratio of the latter to the total amount of language input an *exposure ratio* α. This α is subdivided into smaller ratios corresponding to those other languages, where each ratio is in proportion to the population of the language speakers. An example distribution of languages is shown in Figure 1(a). Suppose a child has parents who speak G_p, s/he receives input sentences from G_p on the percentage of the shaded part, $\alpha x_p + (1-\alpha)$, and from non-parental languages $G_i (i \neq p)$ on the percentage, αx_i.

We have adopted a batch learning algorithm, which resolves Niyogi [11]'s problem regarding an unrealistic Markov structure which implies that some children cannot learn certain kinds of language. From the viewpoint of universal grammar, that all conceivable grammars of human beings are restricted to a finite set [12], language learning is considered as a choice of a plausible grammar from them. The following algorithm realizes such learning as: 1) In a child's memory, there is supposed to be a score table of grammars. 2) The child receives a sentence uttered by an adult. 3) The acceptability of the sentence is tested using



(a) The exposure ratio α (b) The learning algorithm

Fig. 1. The learning algorithm including the exposure ratio α

each grammar. The grammar which accepts the sentence scores one point. 4) Steps 2) and 3) are repeated until the child receives a fixed number (w) of sentences, which is regarded as sufficient for the decision of the grammar selection. 5) The child adopts the grammar with the highest score.

The child is exposed to utterances of adult speakers of each language, the percentage of which is determined by the distribution of population and the exposure ratio α, while the S matrix determines the acceptability of a sentence. In Figure 1(b), we show an example where a child of G_2 speaker obtains G_2 after exposure to a variety of languages. The child receives sentences, which are boxes numbered from 1 to 10. The input sentences are divided into two sets according to the exposure ratio α. One of the sets consists of sentences of all grammars. The number of the sentences of each language is proportional to the population share of the language speakers. For example, the child hears sentences 1, 4 and 5 uttered by G_1 speakers. The other consists of sentences of his/her parents. Therefore, these sentences are acceptable by a particular grammar. Because his/her parental grammar is G_2, for example, the sentences 7 to 10 are randomly chosen from the language of G_2. The child counts acceptable sentences for each grammar. The sentence 1 can be accepted by G_3 as well as G_1, while it is uttered by a G_1 speaker. The Venn diagram in Figure 1(b) represents that each language shares sentences with others. In this case, because the sentence 1 is acceptable both by G_1 and by G_3, the child adds 1 to both of the counters in his/her mind.

2.4 Revised Transition Probability

Suppose that children hear sentences from adult speakers depending on the exposure ratio and on the distribution of population. A probability that a child whose parents speak G_i accepts a sentence by G_j is expressed by:

$$U_{ij} = \alpha \sum_{k=1}^{n} s_{kj} x_k + (1-\alpha) s_{ij} \quad . \tag{2}$$

After receiving a sufficient number of sentences for language acquisition, the child will adopt the most plausible grammar, as estimated by counting the number of sentences accepted by each grammar. This learning algorithm is simply represented in the following equation. Exposed to a variety of languages in proportion to the population share of adult speakers, children whose parents speak G_i will adopt G_{j^*} by:

$$j^* = \underset{j}{\operatorname{argmax}}\{U_{ij}\} \quad . \tag{3}$$

When the children hear w sentences, a probability that a child of G_i speaker accepts r sentences with G_j is given by a binomial distribution,

$$g_{ij}(r) = \binom{w}{r} (U_{ij})^r (1-U_{ij})^{w-r} \quad . \tag{4}$$

On the other hand, a probability that the child accepts less than r sentences with G_j is

$$h_{ij}(r) = \sum_{k=0}^{r-1} \binom{w}{k} (U_{ij})^k (1-U_{ij})^{w-k} \quad . \tag{5}$$

From these two probability distributions, the probability that a child of G_i speaker accepts k sentences with G_j, while less than $k-1$ sentences with the other grammars, comes to $g_{ij}(k) \prod_{l=1, l\neq j}^{n} h_{il}(k)$. For a child of G_i speaker to acquire G_j after hearing w sentences, G_j must be the most efficient grammar among n grammars; viz., G_j must accept at least $\lceil \frac{w}{n} \rceil$ sentences. Thus, the probability $\overline{q}_{ij}$ becomes the sum of the probabilities that G_j accepts $w, w-1, \cdots, \lceil \frac{w}{n} \rceil$ sentences. Because each of the sentences is uttered by a speaker and is accepted by at least one grammar, there must be a grammar which accepts $\lceil \frac{w}{n} \rceil$ or more out of w sentences. Thus, if G_j accepts less than $\lceil \frac{w}{n} \rceil$ sentences, the child does not acquire G_j. Therefore, $\overline{q}_{ij}$ becomes:

$$\overline{q}_{ij}(t) = \frac{\displaystyle\sum_{k=\lceil \frac{w}{n} \rceil}^{w} \left\{ g_{ij}(k) \prod_{\substack{l=1 \\ l\neq j}}^{n} h_{il}(k) + R(k,n) \right\}}{\displaystyle\sum_{m=1}^{n} \left[\sum_{k=\lceil \frac{w}{n} \rceil}^{w} \left\{ g_{im}(k) \prod_{\substack{l=1 \\ l\neq m}}^{n} h_{il}(k) + R(k,n) \right\} \right]} \quad , \tag{6}$$

where $R(k,n)$ is the sum total of the probabilities that the child would choose G_j when one or more other grammars accept the same number of sentences as G_j. When there are m candidate grammars including G_j, the probability becomes one divided by m. The following expression is an example when $n = 3$.

$$\begin{aligned} R_{ij}(k,3) = {} & \tfrac{1}{3}\{g_{ij}(k) g_{ij_2}(k) g_{ij_3}(k)\} \\ & + \tfrac{1}{2}\{g_{ij}(k) g_{ij_2}(k) h_{ij_3}(k) + g_{ij}(k) h_{ij_2}(k) g_{ij_3}(k)\} \\ & \qquad (j_2, j_3 \in \{1,2,3\}, j \neq j_2, j_3, j_2 \neq j_3) \end{aligned} \tag{7}$$

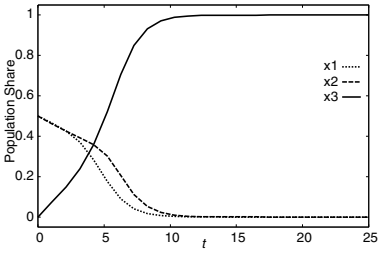


Fig. 2. Example of creolization $((a, b, c) = (0, 0.3, 0.4), w = 10, \alpha = 0.7)$

3 Creole in Population Dynamics

Creoles are considered as new languages. From the viewpoint of population dynamics, we define a creole as a transition of population of language speakers. A creole is a language which no one spoke in the initial state, but most people have come to speak at a stable generation. Therefore, creole is represented by G_c such that: $x_c(0) = 0, x_c(t) > \theta_c$, where $x_c(t)$ denotes the population share of G_c at a convergent time t, and θ_c is a certain threshold to be regarded as a dominant language. We set $\theta_c = 0.9$ through the experiments.

For convenience, we have mainly observed the behavior of the model using three grammars. The similarity matrix can be expressed as a symmetric matrix such that:

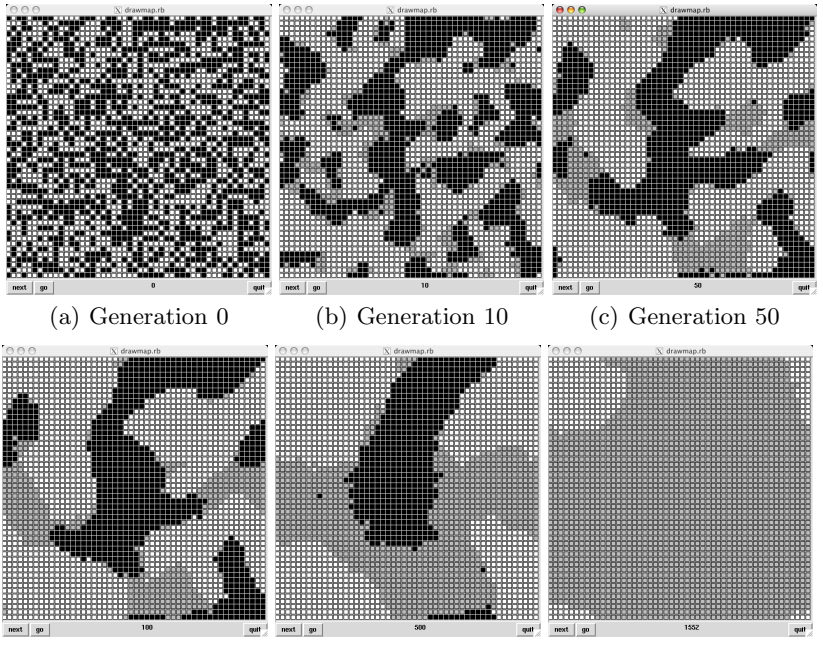
$$S = \begin{pmatrix} 1 & a & b \\ a & 1 & c \\ b & c & 1 \end{pmatrix} \quad . \tag{8}$$

Here, we regard G_3 as a creole grammar, giving the initial condition as $(x_1(0), x_2(0), x_3(0)) = (0.5, 0.5, 0)$. Therefore, the element a denotes the similarity between two pre-existing languages, and b and c are the similarities between G_1 and the creole, and between G_2 and the creole, respectively.

We show an example of creolization in Figure 2. The parameters were set to $(a, b, c) = (0, 0.3, 0.4)$, $w = 10$, and $\alpha = 0.7$. Note that the conditions in the parameter space for dominant creoles are limited [13].

4 Experiments and Results

The spatial structure is a toroidal 50-by-50 square grid. Each agent has 8 neighbors. Each agent chooses one of three languages every generation, two of which, G_1 and G_2, are pre-existing and randomly distributed with the same total number at the initial state. The remaining language, G_3, is a creole, having a certain similarity between two languages. The similarity means the probability that a sentence uttered by a G_i speaker is accepted by G_j. In this paper, We take the following values: $(a, b, c) = (0, 0.3, 0.4)$, and $w = 10$ for the number of input sentences.



(a) Generation 0 (b) Generation 10 (c) Generation 50

(d) Generation 100 (e) Generation 500 (f) Generation 1552

Fig. 3. Example of the spatial dynamics (white:G_1, black:G_2, gray:G_3; $\alpha = 0.7$)

4.1 Behaviors of Spatial Dynamics

We show an example of the spatial dynamics in Figure 3; (a) Only G_1 and G_2 are distributed at the initial stage. (b,c) Some local communities (hereafter colonies) of creole are organized at the early stage. (d,e) Both G_1, G_2 and creole coexist at a quasi-stable stage. (f) In this trial, the creole eventually becomes dominant at Generation 1552. Agents surrounded by both G_1 and G_2 neighbors are likely to acquire the creole. In fact, creole speakers often appear on the border between communities. This is because the large value of α makes the agents to be exposed to both languages, and the creole is the most efficient for accepting input utterances from both languages.

In general, learners tend to form a colony, regardless of the languages, affected from its neighbors during learning acquisition. However, the smaller the value of α, the slower the forming colonies. This is because the learners are hardly affected by neighbors, hearing their mother tongue from their parents. Note that even if $\alpha = 0$, it is possible for a learner to acquire a language other than his/her mother tongue due to the similarity among languages. The number of input sentences is also relevant to forming a colony. Learners hearing a lot of language input become conservative in terms of changing his/her language. That is, once a small colony has been formed at a generation, inhabitants in the colony come to choose the same language as the previous generation. On the contrary, if learners choose a

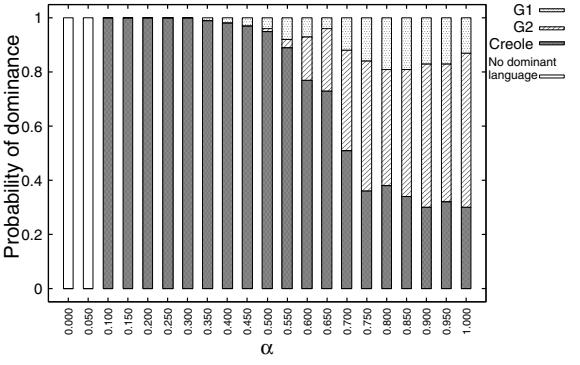


Fig. 4. Probability of dominant language in the spatial model

language with less language input, they tend to be undetermined in choosing a language. As a result, it is difficult to form a colony.

4.2 Comparing the Two Models with and without a Spatial Structure

We examine the probability of dominance for each language (Figure 4). Note that the spatial model is based on a stochastic dynamics. This graph is the result of 100 runs for 1,000,000 generations at each α value. The corresponding result in the non-spatial model is the population distribution at the stable generation, shown in Figure 5[1], since the non-spatial model is based on the deterministic dynamics. This parameter set makes creole dominant at the range $0.1 \lesssim \alpha \lesssim 0.8$. In the spatial model, the probability that the creole is dominant gradually decreases from $\alpha > 0.3$, and it becomes 0.3 around $\alpha > 0.8$.

In general, the larger the value of α, the more prominent the transition of population becomes, and in some cases, the transition leads to creolization. In Figure 5, however, the dominant language changes between the creole and G_2 at $\alpha \simeq 0.8$, and it is not always true that creoles are more likely to become dominant at the larger value of α (See [6]). Since children of a G_1 speaker become more exposed to G_2 in the larger value of α, it is possible for them to acquire G_2 directly instead of the creole, and vice versa. Therefore, the population of the creole remains small and the children of creole speakers are likely to acquire G_2 rather than G_1. Thus, the dominant language changes to G_2 at $\alpha \gtrsim 0.8$[2].

These differences between the results shown in Figure 4 and Figure 5 can be understood by considering local interaction and stochastic dynamics. The pre-existing language may be able to form a colony due to stochasticity. Once a colony with certain size is formed, agents in the colony are surrounded by the

[1] In other words, the result of the population distribution at the stable generation in Figure 2 is plotted at $\alpha = 0.7$ in Figure 5.

[2] In the experiments, we have chosen this parameter set with which creoles tend to appear in the wide range of α.

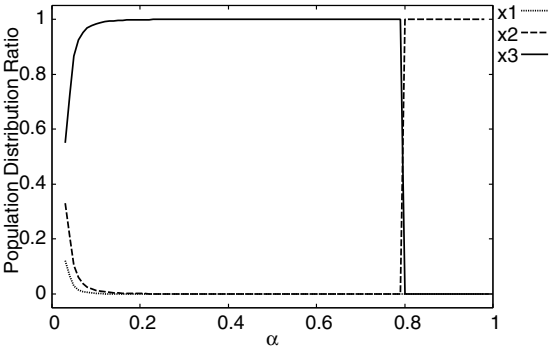


Fig. 5. Stable population distribution in the non-spatial model

same language and the exposure ratio effectively comes to $\alpha = 0$. This situation is hard for the creole speakers to organize a colony. Thus, the probability to be dominant is restrained by the pre-existing language at the middle-high range of α. At the higher range of α, the creole can organize a colony at the early stage with certain ratio through random migration. The colony can grow to the whole space.

4.3 Quantitative Analysis in Forming Creole Communities

Observing behaviors of the spatial dynamics, we realized that forming creole colonies at early stages plays a key role for creolization. Large creole colonies are difficult to vanish and are able to encroach upon a territory of other languages. In other words, if creole speakers fail to form a certain size of colony before the quasi-stable stage, it is difficult for creole to become dominant. Therefore, in this section we try to predict whether creolization takes place or not at the early stage, observing creole colonies quantitatively.

For a quantitative description of the emergence and dynamics of linguistic spatial domains we use the ensemble average interface density $\langle \rho \rangle$ as an order parameter, following a precedent work [8]. This is defined as the density of links joining nodes in the network which are in different states [14]. For associating with the population distribution, we use the inverse value as $\langle \overline{\rho} \rangle = \langle 1 - \rho \rangle$. The ensemble average, indicated as $\langle \cdot \rangle$, denotes average over realizations of the stochastic dynamics starting from different random distributions of initial conditions. As the time proceeds, the increase of $\overline{\rho}$ from its initial value describes the ordering dynamics, where linguistic spatial domains, in which agents are in the same state, grow in time. The maximum value $\overline{\rho} = 1$ corresponds to a stationary configuration in which all the agents belong to the same linguistic community. In addition, we defined $\overline{\rho}_i$ as the density of G_i speakers' neighbors which are in the same states.

We analyzed difference of the behaviors at the early stage among 100 trials every α, shown in Figure 4, classifying the trials into three, each of which denotes the corresponding language eventually becomes dominant. We recognize an early

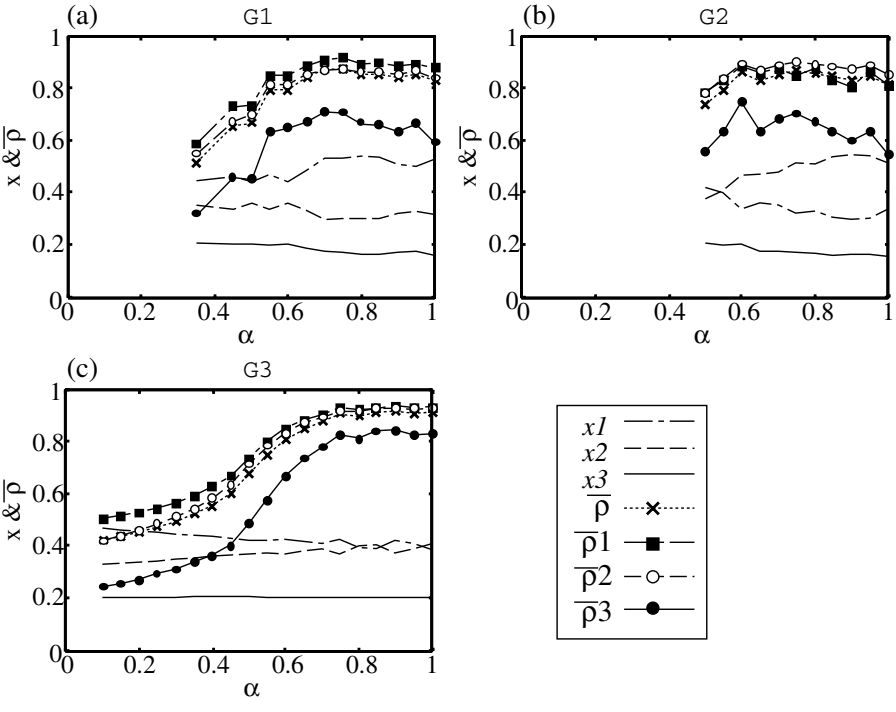


Fig. 6. Difference of x_i and $\langle\overline{\rho}_i\rangle$ at the early stage ($x_3 \simeq 0.2$) between results in which G_1, G_2, and G_3 eventually become dominant respectively

stage as the earliest generation at which $x_3(t)$ exceeds 0.2 in each trial. Figure 6 shows the difference of the average $\langle\overline{\rho}\rangle$, $\langle\overline{\rho}_i\rangle$, and x_i every α at the early stage between G_1, G_2 and G_3 becoming dominant at the stable generation. At some values of α in Figure 6(a) and (b), the values of x_3 are partially plotted less than 0.2, because in some trials x_3 never exceeded 0.2. In this case, we took the values at the stable generation for the average calculation. At some values of α where no data are plotted, there was no trial in which the corresponding language became dominant. Therefore, Figure 6 corresponds to the frequency distribution of dominance, shown in Figure 4.

In Figure 6, we can see that $\langle\overline{\rho}_3\rangle$ is lower than other densities at any value of α. Note that the average density of G_i denoted by $\langle\overline{\rho}_i\rangle$ is affected not only by a degree of forming colonies but also by its population[3]. Therefore, it is natural that only 20 percent of the population obtains the density smaller than others.

As was mentioned in Section 4.2, creolization takes place even at the large value of α ($0.8 \lesssim \alpha$), while G_2 dominates the community in the non-spatial model (Figure 5). We can see that the values of $\langle\overline{\rho}_3\rangle$ in Figure 6(c) are higher than that of (a) and (b). This tendency can be seen with other parameters of w

[3] Suppose there is only a colony of G_i forming a square in the space. If the size of the colony is 5-by-5, $\langle\overline{\rho}_i\rangle = 0.72$, while it is 0.855 for a 10-by-10 colony.

and S. Therefore, we consider that forming creole colonies at the early stage is important for full creolization.

On the contrary, at small values of α, the values of $\langle \overline{\rho}_3 \rangle$ are also small. Because the exposure ratio α determines a probability that a language learner communicates with its neighbors, forming a colony with neighbors is hardly effective in creolization at the small value of α. Rather, the creole becomes dominant due to an advantageous parameter set of w, S, and α. It is clear as evidenced by the result of the non-spatial model.

5 Conclusion

In this paper, we introduced a spatial structure to a mathematical framework of creolization. Observing this process, we discovered that forming colonies was an important factor. We showed that in the spatial language dynamics, creole could be dominant even in the high exposure ratio, different from the non-spatial model.

The quantitative analysis implies that there is a condition of creolization in terms of a combination between the ensemble average density $\langle \overline{\rho}_3 \rangle$ and the exposure ration α. Through the experiments, we can conclude as follows:

- Creole is easy to dominate the community in a parameter set where creolization takes place in the non-spatial model, regardless of the value of α.
- The value of $\langle \overline{\rho}_3 \rangle$ is probably useful for the prediction of creolization at the early stage at the large values of α.

We need to analyze the behavior through further experiments. Although we used a toroidal 50-by-50 square grid for a spatial structure, it can be expanded to more complicated social networks. There is yet room for improvement in some settings including the initial population distribution.

Acknowledgment

This work was partly supported by Grant-in-Aid for Young Scientists (B) (KAKENHI) No.20700239 from MEXT Japan.

References

1. Cangelosi, A., Parisi, D. (eds.): Simulating the Evolution of Language. Springer, London (2002)
2. Briscoe, E.J. (ed.): Linguistic Evolution through Language Acquisition: Formal and Computational Models. Cambridge University Press, Cambridge (2002)
3. Arends, J., Muysken, P., Smith, N. (eds.): Pidgins and Creoles. John Benjamins Publishing Co., Amsterdam (1994)
4. Bickerton, D.: Language and Species. University of Chicago Press (1990)
5. DeGraff, M. (ed.): Language Creation and Language Change. MIT Press, Cambridge (1999)

6. Nakamura, M., Hashimoto, T., Tojo, S.: Exposure dependent creolization in language dynamics equation. In: Sakurai, A., Hasida, K., Nitta, K. (eds.) JSAI 2003. LNCS (LNAI), vol. 3609, pp. 295–304. Springer, Heidelberg (2007)
7. Nowak, M.A., Komarova, N.L., Niyogi, P.: Evolution of universal grammar. Science 291, 114–118 (2001)
8. Castelló, X., Eguíluz, V.M., Miguel, M.S., Loureiro-Porto, L., Toivonen, R., Saramäki, J., Kaski, K.: Modelling language competition: bilingualism and complex social networks. In: Smith, A., Smith, K., Cancho, R. (eds.) The Evolution of Language: Proceedings of the 7th International Conference (EVOLANG7), p. 85. World Scientific Pub. Co. Inc., Singapore (2008)
9. Abrams, D.M., Strogatz, S.H.: Modelling the dynamics of language death. Nature 424, 900 (2003)
10. Nakamura, M., Hashimoto, T., Tojo, S.: Creole viewed from population dynamics. In: Proc. of the Workshop on Language Evolution and Computation in ESSLLI, Vienna, pp. 95–104 (2003)
11. Niyogi, P.: The Informational Complexity of Learning. Kluwer, Boston (1998)
12. Chomsky, N.: Lectures on Government and Binding. Foris, Dordrecht (1981)
13. Nakamura, M., Hashimoto, T., Tojo, S.: Simulation of common language acquisition by evolutionary dynamics. In: Proc. of IJCAI 2007 Workshop on Evolutionary Models of Collaboration, Hyderabad, pp. 21–26 (2007)
14. Miguel, M.S., Eguiluz, V.M., Toral, R., Klemm, K.: Binary and multivariate stochastic models of consensus formation. Computing in Science and Engg. 7, 67–73 (2005)

On the Average Size of Glushkov's Automata

Cyril Nicaud*

LIGM, UMR CNRS 8049, Université Paris Est, 77454 Marne-la-Vallée, France
nicaud@univ-mlv.fr

Abstract. Glushkov's algorithm builds an ε-free nondeterministic automaton from a given regular expression. In the worst case, its number of states is linear and its number of transitions is quadratic in the size of the expression. We show in this paper that in average, the number of transitions is linear.

1 Introduction

Kleene's Theorem states that regular expressions and automata describe the same objects, regular languages. They both are finite objects which encode infinite sets of words and they play a key role in formal language theory as well as in its many fields of application.

One can often take advantage of having this two different representations, the algorithmic problems of transforming one representation into another are therefore fundamental. They have been widely studied and are still improved nowadays, as one can see in a recent survey by J. Sakarovitch [1].

In this paper we focus on one particular such transformation, Glushkov's algorithm [2], an algorithm that builds an ε-free nondeterministic automaton from a given regular expression. Starting from a regular expression with n letters, it builds an automaton with $n+1$ states and $\mathcal{O}(n^2)$ transitions. In [3], J. Hromkovič, S. Seibert and T. Wilke proposed a variation on this algorithm where the produced automaton has $\mathcal{O}(n \log^2 n)$ states. They also proved a lower bound of $\Omega(n \log n)$ states for this general problem. Based on this work, C. Hagenah and A. Muscholl proposed in [4] an algorithm of time complexity $\mathcal{O}(n \log^2 n)$ to achieve the construction. See also [5,6] for some efficient related algorithms.

Our contribution is to give an average case analysis of the size of Glushkov's automata. Using the framework of analytic combinatorics, we prove that for the uniform distribution of regular expressions, the average number of transitions is linear.

The use of generating functions and complex analysis have proved to be useful in average case analysis of algorithms [7]. The methodology we shall use here can be summarized as follows:

1. Find an unambiguous specification of the objects, which can be recursive.
2. Transform the specification into a functional equation for the associated generating function.

* The author is supported by the ANR (GAMMA - project BLAN07-2_195422).

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 626–637, 2009.

3. Analyze the dominant singularities of the generating function to obtain the needed asymptotics.

The paper is organized as follows. In Section 2, we recall the basic definitions and present some analytic tools. In Section 3, we study the generating function associated to regular expressions. Section 4 is devoted to our main result. Finally, a short discussion about the distribution is presented in Section 5.

Note that due to the lack of space, we can not show all the details of the computations, but most of them are easily done with the help of a computer-algebra system.

2 Preliminaries

2.1 Automata and Regular Expressions

An *automaton* defined on a finite alphabet A is a tuple (A, Q, T, I, F) where Q is the finite *set of states*, $T \subset Q \times A \times Q$ is the set of *transitions*, $I \subset Q$ is the set of *initial states* and $F \subset Q$ is the set of *final states*. We refer the reader unfamiliar with basic notions of automata and regular languages to [8] for definitions and fundamental results.

The set of nonempty *regular expressions* $\mathcal{R}$ on a finite alphabet A is a set of words on the alphabet $\{\varepsilon, \cdot, *, \cup, (,)\} \cup A$ defined inductively by: $\varepsilon \in \mathcal{R}$, $A \subset \mathcal{R}$, $(R)* \in \mathcal{R}$ for all $R \in \mathcal{R}$, $(R_1 \cdot R_2)$ and $(R_1 \cup R_2)$ for every $R_1, R_2 \in \mathcal{R}$. A language defined on A is *denoted* by a regular expression of $\mathcal{R}$ when it is exactly the set of words obtained by interpreting each symbol $*$, $\cdot$ or $\cup$ as the corresponding regular operation on sets of words. Let $L(R)$ be the language denoted by $R \in \mathcal{R}$. For convenience, we shall freely remove parenthesis symbols that are not needed in an element of $\mathcal{R}$. It is also often useful to see regular expressions as trees, with this equivalent inductive definition:

$$\begin{cases} \varepsilon \in \mathcal{R} & \\ a \in \mathcal{R} & \forall a \in A \\ \overset{*}{|}_{R} \in \mathcal{R} & \forall R \in \mathcal{R} \\ \overset{\cup}{/\!\backslash}_{R_1\ R_2} \in \mathcal{R} & \forall R_1, R_2 \in \mathcal{R} \\ \overset{\bullet}{/\!\backslash}_{R_1\ R_2} \in \mathcal{R} & \forall R_1, R_2 \in \mathcal{R} \end{cases} \tag{1}$$

The *size* of an element of $\mathcal{R}$ is the number of nodes in its tree representation:

$$|\varepsilon| = |a| = 1; \qquad \left|\overset{*}{|}_{R}\right| = |R| + 1; \qquad \left|\overset{\cup}{/\!\backslash}_{R_1\ R_2}\right| = \left|\overset{\bullet}{/\!\backslash}_{R_1\ R_2}\right| = |R_1| + |R_2| + 1$$

Note that one usually add to $\mathcal{R}$ the symbol $\emptyset$ that denotes the empty language. For technical reasons it is slightly more convenient in this paper to work on nonempty regular languages.

2.2 Glushkov's Automaton

Let m be the number of letter symbols in R, for $R \in \mathcal{R}$. We consider the expression $\tilde{R}$ obtained from R by distinguishing the letters with subscripts in $\{1, \cdots, m\}$, marking them from left to right on its string representation, or equivalently using depth-first order on its tree representation. For instance $R = b^* \cdot (a \cup b \cdot b)^*$ is changed into $\tilde{R} = b_1^* \cdot (a_2 \cup b_3 \cdot b_4)^*$. We denote by $pos(R)$ the set of subscripted letters in $\tilde{R}$: $pos(R) = \{b_1, a_2, b_3, b_4\}$ in the example. We also denote by ν the function from $pos(R)$ to A that removes the subscripts, $\nu(a_2) = a$ for instance.

Let $First(R)$ and $Last(R)$ be the sets defined by

$$First(R) = \{\alpha \in pos(R) \mid \exists u \in L(\tilde{R}),\ u \text{ starts with the letter } \alpha\}$$
$$Last(R) = \{\alpha \in pos(R) \mid \exists u \in L(\tilde{R}),\ u \text{ ends with the letter } \alpha\}$$

And for any letter α in $pos(R)$, the set $follow(R, \alpha)$ is defined by

$$follow(R, \alpha) = \{\beta \in pos(R) \mid \exists u \in L(\tilde{R}),\ \alpha\beta \text{ is a factor of } u\}$$

The *Glushkov's automaton* of R, also called the *position automaton*, is the automaton $\mathcal{A}_R = (A, Q, T, \{i\}, F)$ with $Q = pos(R) \cup \{i\}$, $F = Last(R) \cup \{i\}$ if $\varepsilon \in L(R)$ and $F = Last(R)$ otherwise, and $T = \{(i, \nu(\alpha), \alpha) \mid \alpha \in First(R)\} \cup \{(\alpha, \nu(\beta), \beta) \mid \beta \in follow(R, \alpha)\}$. This classical construction provides an automaton that recognizes $L(R)$.

Let $Edges(R)$ be the set of pairs $(\alpha, \beta) \in pos(R)^2$ such that $\beta \in follow(\alpha)$. The number of transitions in $\mathcal{A}_R$ is thus $|First(R)| + |Edges(R)|$. The set $Edges(R)$ can also be defined inductively as follows.

$$\begin{cases} Edges(\varepsilon) = 0 \\ Edges(a) = 0 \\ Edges\left(\overset{*}{|}_{R}\right) = Edges(R) \cup Last(R) \times First(R) \\ Edges\left(\overset{\cup}{/\!\backslash}_{R_1\ R_2}\right) = Edges(R_1) \cup Edges(R_2) \\ Edges\left(\overset{\bullet}{/\!\backslash}_{R_1\ R_2}\right) = Edges(R_1) \cup Edges(R_2) \cup Last(R_1) \times First(R_2) \end{cases} \tag{2}$$

2.3 Generating Functions

A *combinatorial class* $\mathcal{C}$ is a set of objects with a size function $|\cdot|$ from $\mathcal{C}$ to $\mathbb{N}$ such that, for any $n \in \mathbb{N}$, the number c_n of objects of size n in $\mathcal{C}$ is finite. The generating function $C(z)$ of a combinatorial class $\mathcal{C}$ is the formal power series

$$C(z) = \sum_{C \in \mathcal{C}} z^{|C|} = \sum_{n \geq 0} c_n z^n$$

We also denote by $[z^n]C(z) = c_n$ the coefficient of z^n in $C(z)$.

Let $\mathcal{C}$ be a combinatorial class of generating function $C(z)$ and let $f : \mathcal{C} \to \mathbb{R}$ be a mapping from this class to $\mathbb{R}$. The *cost generating function* $F(z)$ of $\mathcal{C}$ associated to f is

$$F(z) = \sum_{C \in \mathcal{C}} f(C) z^{|C|} = \sum_{n \geq 0} f_n z^n, \text{ with } f_n = \sum_{\substack{C \in \mathcal{C} \\ |C|=n}} f(C)$$

For a given n, the average value of f for the uniform distribution on the elements of size n of $\mathcal{C}$ is therefore

$$\mu_n(\mathcal{C}, f) = \frac{[z^n]F(z)}{[z^n]C(z)}$$

The following lemma, though trivial, will be useful throughout this article.

Lemma 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be two combinatorial classes. Let $f : \mathcal{A} \to \mathbb{R}$ and $g : \mathcal{B} \to \mathbb{R}$ be two mappings from $\mathcal{A}$ and $\mathcal{B}$ to $\mathbb{R}$. The following property holds.*

$$\sum_{A \in \mathcal{A}} \sum_{B \in \mathcal{B}} (f(A) + g(B)) z^{|A|} z^{|B|} = F(z)B(z) + A(z)G(z)$$

where $A(z)$, $B(z)$, $F(z)$ and $G(z)$ respectively denote the generating functions of $\mathcal{A}$ and $\mathcal{B}$ and the cost generating functions associated to f and g.

2.4 Symbolic Methods and Transfer Theorem

The *symbolic methods* were introduced in [9]. When they can be applied, they directly and almost automatically build the generating functions associated to combinatorial classes. The idea is to avoid recurrence formulas on the number of objects by providing a dictionary that maps combinatorial constructions on classes into constructions on generating functions. This dictionary covers a lot of used constructions, and one can easily use it to describe combinatorial classes such as trees, permutations, set partitions, integer partitions, random mappings, etc. therefore obtaining directly their generating functions.

In this article we shall only use a small part of this dictionary, the most basic one. If $\mathcal{A} = \mathcal{B} \cup \mathcal{C}$ are combinatorial classes such that $\mathcal{B}$ and $\mathcal{C}$ are disjoint, it is direct to prove that the associated generating functions $A(z)$, $B(z)$ and $C(z)$ satisfy $A(z) = B(z) + C(z)$. Moreover if $\mathcal{A} = \mathcal{B} \times \mathcal{C}$, one can see that $A(z) = B(z)C(z)$. We shall only need this two constructions here, but in this framework, one can directly derive the generating functions of sequences of elements in $\mathcal{A}$, sets of elements in $\mathcal{A}$, and so on. We refer the reader to P. Flajolet and R. Sedgewick's book for more informations about this topic [7].

Once the generating function is known, either in close or implicit form, several theorems exist to compute asymptotic estimations of its coefficients. This theorems mainly use the theory of complex analysis, seeing generating functions as analytic functions from $\mathbb{C}$ to $\mathbb{C}$. The main idea is that the asymptotics of the coefficients of a generating function can be obtained by studying it around its dominant singularities (its singularities of smallest moduli). Informally, given a

generating function $A(z)$ of unique dominant singularity ρ, the transfer theorem states under some analytic conditions that if $A(z) \sim B(z)$ as $z \to \rho$, then $[z^n]A(z) \sim [z^n]B(z)$, for some useful functions $B(z)$ whose coefficient asymptotics are well-known.

We use the notations of [7] to give a formal description of the theorem. Let $R > 1$ and $0 < \phi < \pi/2$ be two real numbers, the domain $\Delta(\phi, R)$ is

$$\Delta(\phi, R) = \{z \in \mathbb{C} \mid |z| < R,\ z \neq 1 \text{ and } |Arg(z-1)| > \phi\}$$

A domain is a *Δ-domain at* 1 if it is a $\Delta(\phi, R)$ for some R and ϕ. For a given complex number $\zeta \neq 0$, a *Δ-domain at* ζ is the image by the mapping $z \mapsto \zeta z$ of a Δ-domain at 1. A function is *Δ-analytic* if it is analytic in some Δ-domain.

Theorem 1 (part of Transfer Theorem). *Let α and β be real number and let $f(z)$ be a function that is Δ-analytic that satisfies, on the intersection of a neighborhood of* 1 *and its Δ-domain, the condition*

$$f(z) = o\left((1-z)^{-\alpha}\left(\log\frac{1}{1-z}\right)^{\beta}\right)$$

then $[z^n]f(z) = o(n^{\alpha-1}(\log n)^{\beta})$.

Recall that Pringsheim's Theorem states that if $f(z)$ is representable at the origin by a series expansion with nonnegative coefficients, one of its dominant singularities, if any, is on $\mathbb{R}^+$. This theorem is useful as generating functions always have nonnegative coefficients.

2.5 Analytic Tools for This Paper

The generating functions we shall study in this paper always have a unique dominant singularity, which is therefore in $\mathbb{R}^+$ by Pringsheim's theorem. For any fixed alphabet size k, the dominant singularity of each generating function will always be the same ρ_k. Moreover, as they are all made of polynomials, quotients and square roots, their analysis have a lot of similarities. In particular, all generating functions satisfy one of the two conditions of the following proposition.

Proposition 1. *Let $f(z)$ be a function that is Δ-analytic at $\rho \in \mathcal{R}^+$.*

1. *If on the intersection of a neighborhood of ρ and its Δ-domain,*

$$f(z) = a - b\sqrt{1-z/\rho} + o(\sqrt{1-z/\rho}),\ \text{with } a, b \in \mathbb{R},\ b \neq 0$$

   then $[z^n]f(z) \sim \frac{b}{2\sqrt{\pi}}\rho^{-n}n^{-3/2}$.
2. *If on the intersection of a neighborhood of ρ and its Δ-domain,*

$$f(z) = \frac{a}{\sqrt{1-z/\rho}} + o\left(\frac{1}{\sqrt{1-z/\rho}}\right),\ \text{with } a \in \mathbb{R},\ a \neq 0$$

   then $[z^n]f(z) \sim \frac{a}{\sqrt{\pi}}\rho^{-n}n^{-1/2}$.

This proposition is a direct consequence of Theorem 1, using classical formulas for the asymptotic of the coefficients of $z \mapsto \sqrt{1-z}$ and $z \mapsto (1-z)^{-1/2}$. If $f(z)$ is a function which is Δ-analytic at $\rho \in \mathbb{R}^+$, we say that f *satisfies the property* Π_1 *or* Π_2 *at* ρ if it satisfies the first or second condition of Proposition 1 respectively.

The computations involved here can be technical, but can be done almost automatically (with the help of computer algebra software for the most complex ones). It is always straightforward to check that this functions satisfy the analytic conditions of Proposition 1. The value of a (and b) can also be computed, and will be some functions of the size of the alphabet. In the rest of the paper, due to the lack of space, most details of the computations will be omitted.

3 Generating Functions for Regular Expressions

In this section we apply the framework of analytic combinatorics to the classes of regular expressions we are studying. From now on, the alphabet considered is $A = \{a_1, \ldots, a_k\}$, where $k \geq 1$ is a positive integer.

3.1 General Regular Expressions

From Equation (1), one can use the symbolic method to obtain the following specification

$$\mathcal{R} = \varepsilon + a_1 + \cdots + a_k + \overset{*}{|}_{\mathcal{R}} + \overset{\cup}{/\!\backslash}_{\mathcal{R}\ \mathcal{R}} + \overset{\bullet}{/\!\backslash}_{\mathcal{R}\ \mathcal{R}}$$

And therefore, using the dictionary, one obtain the following equation for $R(z)$, the generating function associated to $\mathcal{R}$:

$$R(z) = (k+1)z + zR(z) + 2zR^2(z)$$

From which one can derive the following expression for $R(z)$, using the fact that its Taylor coefficients are nonnegative:

$$R(z) = \frac{1 - z - \sqrt{\Delta_k(z)}}{4z}, \text{ with } \Delta_k(z) = 1 - 2z - (7+8k)z^2$$

The unique dominant singularity of $R(z)$ is $\rho_k = \frac{2\sqrt{2k+2}-1}{7+8k}$, and around ρ_k, one has the following expansion of $R(z)$

$$R(z) = \frac{1-\rho_k}{4\rho_k} - \frac{1}{4\rho_k}\sqrt{\Delta_k(z)} + o\left(\left(1 - \frac{z}{\rho_k}\right)^{\frac{1}{2}}\right)$$

$$R(z) = \frac{\sqrt{2k+2}}{2} - \frac{\sqrt{2(1-\rho_k)}}{4\rho_k}\left(1 - \frac{z}{\rho_k}\right)^{\frac{1}{2}} + o\left(\left(1 - \frac{z}{\rho_k}\right)^{\frac{1}{2}}\right)$$

therefore, using Proposition 1, one can obtain an asymptotic equivalent to the number of nonempty regular expressions.

Lemma 2. *The number of elements of size* n *in* $\mathcal{R}$ *is asymptotically equivalent to* $C_k \rho_k^{-n} n^{-3/2}$, *with* $C_k = \frac{\sqrt{2(1-\rho_k)}}{8\rho_k\sqrt{\pi}}$.

Note that if some generating function $f(z)$ has a unique dominant singularity on ρ_k where we can apply Proposition 1, and satisfies as $z \to \rho_k$:

$$f(z) = a - \frac{b}{4\rho_k}\sqrt{\Delta_k(z)} + o(\sqrt{1 - z/\rho_k})$$

then $[z^n]f(z) \sim b\,[z^n]R(z)$. Therefore, for our computations, it is often more convenient to obtain developments in terms of $\frac{1}{4\rho_k}\sqrt{\Delta_k(z)}$ instead of $\sqrt{1 - z/\rho_k}$. The techniques are of course the same, since they only differ by a multiplicative constant as $z \to \rho_k$.

3.2 Regular Expressions of Languages Containing ε

Denote by $\mathcal{R}_\varepsilon$ and $\mathcal{R}_{\overline{\varepsilon}}$ the regular expressions whose associated languages respectively recognize and do not recognize the empty word. A specification of $\mathcal{R}_\varepsilon$ is the following:

$$\mathcal{R}_\varepsilon = \varepsilon + \overset{*}{|}_{\mathcal{R}} + \overset{\cup}{/\!\backslash}_{\mathcal{R}_\varepsilon\ \mathcal{R}} + \overset{\cup}{/\!\backslash}_{\mathcal{R}_{\overline{\varepsilon}}\ \mathcal{R}_\varepsilon} + \overset{\bullet}{/\!\backslash}_{\mathcal{R}_\varepsilon\ \mathcal{R}_\varepsilon} \tag{3}$$

From which one can obtain the following expression:

$$R_\varepsilon(z) = \frac{z + zR(z)}{1 - 2zR(z)}$$

The dominant singularity of $R_\varepsilon(z)$ is also ρ_k and it satisfies Π_1 at ρ_k:

$$R_\varepsilon(z) = \frac{2k + \sqrt{2k+2}}{4k+2} - \frac{1 - 2k + 4k\sqrt{2k+2}}{(2k+1)^2}\frac{\sqrt{\Delta_k(z)}}{4\rho_k} + o\left(\left(1 - \frac{z}{\rho_k}\right)^{\frac{1}{2}}\right)$$

Therefore, using Proposition 1 one can obtain an asymptotic equivalent to its coefficients:

$$[z^n]R_\varepsilon(z) \sim D_k[z^n]R(z), \text{ with } D_k = \frac{1 - 2k + 4k\sqrt{2k+2}}{(2k+1)^2} \tag{4}$$

Note that, as a consequence, the ratio of regular expressions whose denoted language contains the empty word is asymptotically D_k. For a two-letters alphabet, the value of D_k is approximatively $D_2 \approx 0.664$.

4 Main Result

This section is devoted to the proof of our main theorem:

Theorem 2. *The average number of transitions of the Glushkov's automaton associated to a regular expression of size n, for the uniform distribution, is in $\Theta(n)$.*

4.1 Lower Bound

First remark that if an expression $R \in \mathcal{R}$ contains m letters, then its Glushkov's automaton has $m + 1$ states. Moreover, since it is accessible by construction, it has at least m transitions. For $R \in \mathcal{R}$, let $\ell(R)$ be the number of letters in R.

Let $L(z)$ be the cost generating function of the number of letters in an element R of $\mathcal{R}$:

$$L(z) = \sum_{R \in \mathcal{R}} \ell(R) z^{|R|}$$

From Equation (1) and Lemma 1, we have

$$L(z) = kz + zL(z) + 2zL(z)R(z) + 2zL(z)R(z)$$

Hence, using the fact that $1 - z - 4zR(z) = \sqrt{\Delta_k(z)}$:

$$L(z) = \frac{kz}{\sqrt{\Delta_k(z)}}$$

Therefore, $L(z)$ has a unique dominant singularity at ρ_k and satisfy Π_2 at ρ_k. By Proposition 1,

$$[z^n]L(z) \sim \frac{k\rho_k}{\sqrt{2\pi(1-\rho_k)}} \rho_k^{-n} n^{-1/2}$$

and the average number of letters in an element of size n of $\mathcal{R}$ is equivalent to $\frac{4k\rho_k^2}{1-\rho_k} n$ as $n \to \infty$. For $k = 2$, an approximation of $\frac{4k\rho_k^2}{1-\rho_k}$ is 0.408. We conclude that in average, the number of transitions in the Glushkov's automaton of an element of size n of $\mathcal{R}$ is in $\Omega(n)$.

4.2 Outline of the Proof of the Upper Bound

In the following subsections we establish some results used to compute an upper bound for the average number of transitions in a Glushkov's automaton. As stated before, the size of the Glushkov's automaton associated to a nonempty regular expression $R \in \mathcal{R}$ is equal to the number of elements in $First(R)$ plus the number of elements in $Edges(R)$. In the next subsection we prove that in average, the size of $First(R)$ tends towards a constant. For the remaining part, we actually compute an upper bound of the size of $Edges(R)$, not its exact average cardinality. Define the function $e : \mathcal{R} \longrightarrow \mathbb{N}$ by:

$$\begin{cases} e(\varepsilon) = 0 & \\ e(a) = 0 & \forall a \in A \\ e\left(\overset{*}{|}_{R}\right) = e(R) + |Last(R)| \cdot |First(R)| & \forall R \in \mathcal{R} \\ e\left(\overset{\cup}{/_{R_1}\backslash_{R_2}}\right) = e(R_1) + e(R_2) & \forall R_1, R_2 \in \mathcal{R} \\ e\left(\overset{\bullet}{/_{R_1}\backslash_{R_2}}\right) = e(R_1) + e(R_2) + |Last(R_1)| \cdot |First(R_2)| & \forall R_1, R_2 \in \mathcal{R} \end{cases} \tag{5}$$

Clearly $e(R)$ is an upper bound of $|Edges(R)|$, since its definition is obtained from the one of $Edges$, Equation (2), and the inequality $|X \cup Y| \leq |X| + |Y|$.

Let $E(z)$ and $F(z)$ be the cost generating functions of e and $|First(\cdot)|$ respectively, and let $P(z)$ be the following generating function

$$P(z) = \sum_{R \in \mathcal{R}} |First(R)| \cdot |Last(R)| z^{|R|}$$

Note that, by symmetry, $F(z)$ is also the cost generating function of $\texttt{last}$, $F(z) = \sum_{R\in\mathcal{R}} \texttt{last}(R) z^{|R|}$.

Using Equation (5) and Lemma 1, one has

$$E(z) = zE(z) + zP(z) + 2zR(z)E(z) + 2zR(z)E(z) + zF(z)^2$$

hence

$$(1 - z - 4zR(z))E(z) = zP(z) + zF(z)^2$$

and as $1 - z - 4zR(z) = \sqrt{\Delta_k(z)}$, we finally obtain

$$E(z) = \frac{zP(z) + zF(z)^2}{\sqrt{\Delta_k(z)}} \tag{6}$$

At this point we need more informations about $P(z)$ and $F(z)$ in order to conclude that $[z^n]E(z) = n\,\mathcal{O}([z^n]\mathcal{R}(z))$. They will be obtained in the next subsections.

4.3 The Cost Generating Function $F(z)$

For an element $R \in \mathcal{R}$, let $\texttt{first}(R)$ and $\texttt{last}(R)$ denote respectively the cardinalities of the sets $First(R)$ and $Last(R)$. We analyze the average value of $\texttt{first}(R)$, for a regular expression $R \in \mathcal{R}$ of size n. $F(z)$ is the cost generating function of $\texttt{first}$, and we also consider its restrictions $F_\varepsilon(z)$ to $\mathcal{R}_\varepsilon$ and $F_{\overline{\varepsilon}}(z)$ to $\mathcal{R}_{\overline{\varepsilon}}$:

$$F_\varepsilon(z) = \sum_{R\in\mathcal{R}_\varepsilon} \texttt{first}(R) z^{|R|}; \quad F_{\overline{\varepsilon}}(z) = \sum_{R\in\mathcal{R}_{\overline{\varepsilon}}} \texttt{first}(R) z^{|R|}$$

For a given $R \in \mathcal{R}$, the value of $\texttt{first}(R)$ can be computed inductively as

$$\begin{cases} \texttt{first}(\varepsilon) = 0; & \\ \texttt{first}(a) = 1 & \forall a \in A \\ \texttt{first}\left(\overset{*}{\underset{R}{|}}\right) = \texttt{first}(R) & \forall R \in \mathcal{R} \\ \texttt{first}\left(\overset{\cup}{\underset{R_1}{/\!\backslash}}\,\underset{R_2}{}\right) = \texttt{first}(R_1) + \texttt{first}(R_2) & \forall R_1, R_2 \in \mathcal{R} \\ \texttt{first}\left(\overset{\bullet}{\underset{R_1}{/\!\backslash}}\,\underset{R_2}{}\right) = \texttt{first}(R_1) + \texttt{first}(R_2) & \forall R_1 \in \mathcal{R}_\varepsilon,\ \forall R_2 \in \mathcal{R} \\ \texttt{first}\left(\overset{\bullet}{\underset{R_1}{/\!\backslash}}\,\underset{R_2}{}\right) = \texttt{first}(R_1) & \forall R_1 \in \mathcal{R}_{\overline{\varepsilon}},\ \forall R_2 \in \mathcal{R} \end{cases} \tag{7}$$

We consider the following specification of $\mathcal{R}$

$$\mathcal{R} = \varepsilon + a_1 + \cdots + a_k + \overset{*}{\underset{\mathcal{R}}{|}} + \overset{\cup}{\underset{\mathcal{R}}{/\!\backslash}}\,\underset{\mathcal{R}}{} + \overset{\bullet}{\underset{\mathcal{R}_\varepsilon}{/\!\backslash}}\,\underset{\mathcal{R}}{} + \overset{\bullet}{\underset{\mathcal{R}_{\overline{\varepsilon}}}{/\!\backslash}}\,\underset{\mathcal{R}}{}$$

This specification can be transformed into a functional equation on cost generating functions using Lemma 1 and Equation (7). For instance, the construction $\overset{\bullet}{\underset{\mathcal{R}_\varepsilon}{/\!\backslash}}\,\underset{\mathcal{R}}{}$ produces the term:

$$z \sum_{R_1\in\mathcal{R}_\varepsilon} \sum_{R_2\in\mathcal{R}} (\texttt{first}(R_1) + \texttt{first}(R_2)) z^{|R_1|} z^{|R_2|} = zR_\varepsilon(z)F(z) + zF_\varepsilon(z)R(z)$$

All computations and simplifications done, we obtain:

$$F(z) = \frac{kz}{1 - z - 3zR(z) - zR_\varepsilon(z)}$$

As $R(z)$ and $R_\varepsilon(z)$ are known, it is straightforward to obtain the expansion of $F(z)$ near its dominant singularity ρ_k and to check that it satisfies Π_1 at ρ_k:

$$F(z) = E_k - F_k \frac{\sqrt{\Delta_k(z)}}{4\rho_k} + o\left(\left(1 - \frac{z}{\rho_k}\right)^{\frac{1}{2}}\right) \tag{8}$$

with $E_k = 1 + \sqrt{2k+2}$ and $F_k = \frac{12+6k+8\sqrt{2k+2}}{k}$. Hence, using Proposition 1, one can prove the following proposition.

Proposition 2. *For any fix integer $k \geq 1$, the average size of* `first` *tends toward a constant $F_k = \frac{12+6k+8\sqrt{2k+2}}{k}$, as n tends toward infinity: $[z^n]F(z) \sim F_k\,[z^n]R(z)$.*

For $k = 2$, an approximation of F_k is $F_2 \approx 21.8$.

4.4 The Cost Generating Function $F_\varepsilon(z)$

Similarly, one can use the specification of $\mathcal{R}_\varepsilon$ of Equation (3) to compute $F_\varepsilon(z)$. We obtain

$$F_\varepsilon(z) = \frac{zF(z) + 2zR_\varepsilon(z)F(z)}{1 - 2zR(z)}$$

Once again, $F_\varepsilon(z)$ satisfies Π_1 at ρ_k with

$$F_\varepsilon(z) = G_k - H_k \frac{\sqrt{\Delta_k(z)}}{4\rho_k} + o(\sqrt{\Delta_k(z)}), \text{ with } G_k = \frac{4k + 1 + \sqrt{2k+2}}{2k+1}$$

The expression of $H_k > 0$ in terms of k can also be computed, but is not needed in the following.

4.5 The Cost Generating Function of the Product `first` × `last`

We analyze the average value of the product of $\texttt{first}(R)$ and $\texttt{last}(R)$, for a regular expression $R \in \mathcal{R}$ of size n. We consider the cost generating function $P(z)$ of this product, and its restriction $P_\varepsilon(z)$ to $\mathcal{R}_\varepsilon$ and $P_{\overline{\varepsilon}}(z) = P(z) - P_\varepsilon(z)$ to $\mathcal{R}_{\overline{\varepsilon}}$:

$$P_\varepsilon(z) = \sum_{R \in \mathcal{R}_\varepsilon} \texttt{first}(R) \cdot \texttt{last}(R) z^{|R|}; \quad P_{\overline{\varepsilon}}(z) = \sum_{R \in \mathcal{R}_{\overline{\varepsilon}}} \texttt{first}(R) \cdot \texttt{last}(R) z^{|R|}$$

We use the following specification of $\mathcal{R}$:

$$\mathcal{R} = \varepsilon + a_1 + \cdots + a_k + \overset{*}{|}_{\mathcal{R}} + \overset{\cup}{\wedge}_{\mathcal{R}\,\mathcal{R}} + \overset{\bullet}{\wedge}_{\mathcal{R}_\varepsilon\,\mathcal{R}_\varepsilon} + \overset{\bullet}{\wedge}_{\mathcal{R}_\varepsilon\,\mathcal{R}_{\overline{\varepsilon}}} + \overset{\bullet}{\wedge}_{\mathcal{R}_{\overline{\varepsilon}}\,\mathcal{R}_\varepsilon} + \overset{\bullet}{\wedge}_{\mathcal{R}_{\overline{\varepsilon}}\,\mathcal{R}_{\overline{\varepsilon}}}$$

The cost generating function associated to each term of the specification is computed separately. For instance, one has the following cost function for $\overset{\bullet}{\wedge}_{\mathcal{R}_\varepsilon\ \mathcal{R}_{\overline{\varepsilon}}}$:

$$\begin{aligned}
&\sum_{R_1\in\mathcal{R}_\varepsilon}\sum_{R_2\in\mathcal{R}_{\overline{\varepsilon}}} \mathtt{first}\left(\overset{\bullet}{\wedge}_{R_1\ R_2}\right)\mathtt{last}\left(\overset{\bullet}{\wedge}_{R_1\ R_2}\right) z^{1+|R_1|+|R_2|}\\
&\qquad = z\sum_{R_1\in\mathcal{R}_\varepsilon}\sum_{R_2\in\mathcal{R}_{\overline{\varepsilon}}} (\mathtt{first}(R_1)+\mathtt{first}(R_2))\,\mathtt{last}(R_2) z^{|R_1|}z^{|R_2|}\\
&\qquad = z\sum_{R_1\in\mathcal{R}_\varepsilon}\sum_{R_2\in\mathcal{R}_{\overline{\varepsilon}}} \mathtt{first}(R_1)\mathtt{last}(R_2) z^{|R_1|}z^{|R_2|} + zR_\varepsilon(z)P_{\overline{\varepsilon}}(z)\\
&\qquad = zF_\varepsilon(z)F_{\overline{\varepsilon}}(z) + zR_\varepsilon(z)P_{\overline{\varepsilon}}(z)
\end{aligned}$$

Using the same techniques for the other parts of the expression, we obtain:

$$P(z) = \frac{kz + 3zF(z)^2 + zF_\varepsilon(z)^2}{1 - z - 2zR(z) - 2zR_\varepsilon(z)}$$

From which, using the previous results, we conclude that $P(z)$ satisfies Π_1 at ρ_k and that there exists some real numbers I_k and J_k such that, as $z \to \rho_k$,

$$P(z) = I_k - J_k\sqrt{\Delta_k(z)} + o(\sqrt{\Delta_k(z)})$$

with $I_k = \frac{38k^2+77k+28+(14k^2+45k+20)\sqrt{2k+2}}{2k(k+1)}$ and $J_k > 0$.

4.6 Concluding the Proof of Theorem 2

We can now analyze Equation (6). As both $F(z)$ and $P(z)$ satisfy Π_1, so does $zP(z) + zF(z)^2$, since the constant term of its development near ρ_k is not zero. Hence $E(z)$ satisfies Π_2 at ρ_k and Proposition 1 can be applied:

$$[z^n]E(z) \sim e_k n\,[z^n]R(z)$$

The value of e_k is not needed to prove the theorem, but is still an indication on a bound of the average number of transitions in a Glushkov's automaton:

$$e_k = \frac{32k^4 + 204k^3 + 406k^2 + 306k + 72 + (80k^3 + 230k^2 + 193k + 52)\sqrt{2k+2}}{2k(k+1)(2k+1)(8k+7)}$$

For $k = 2$, an approximation of e_k is $e_2 \approx 6.77$.

The quantity $[z^n]E(z)/[z^n]R(z)$ is an upper bound of the average cardinality of *Edges* and the average cardinality of *First* is bounded by Proposition 2, therefore, by linearity of the average, the average number of transitions in a Glushkov's automaton is in $\mathcal{O}(n)$. This concludes the proof of Theorem 2.

5 Remarks

In this paper we used the specification of regular expressions that directly follows the inductive definition. One could try to use a more precise specification to

prevent some useless patterns in regular expressions to appear, as was done for instance in [10] for enumeration purposes. Depending on the chosen such rules, the analysis we conduct here could still be doable, but probably much more complicated. Especially if the specification is not context-free (see [11] for a related analysis). However, we believe that the result would be the same in much cases. Indeed, the proofs of Theorem 2 and Proposition 2 are based on the fact that for a non-negligible proportion of regular expressions, the denoted language does not recognize the empty word, and that the average number of concatenation operators is non-negligible. This must be true for most natural specifications.

Also note that we could have used bivariate generating functions instead of cost generating functions in this paper. The computations would have been the same here, but this approach should be useful if one want to go one step further, and try to obtain some informations not only about the average value, but also about the limit distribution.

References

1. Sakarovitch, J.: The language, the expression, and the (Small) automaton. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845, pp. 15–30. Springer, Heidelberg (2006)
2. Glushkov, V.M.: The abstract theory of automata. Russian Math. Surveys 16, 1–53 (1961)
3. Hromkovic, J., Seibert, S., Wilke, T.: Translating regular expressions into small epsilon-free nondeterministic finite automata. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 55–66. Springer, Heidelberg (1997)
4. Hagenah, C., Muscholl, A.: Computing epsilon-free NFA from regular expressions in $O(n \log^2(n))$ time. ITA 34(4), 257–278 (2000)
5. Ilie, L., Yu, S.: Constructing NFAs by optimal use of positions in regular expressions. In: Apostolico, A., Takeda, M. (eds.) CPM 2002. LNCS, vol. 2373, pp. 279–288. Springer, Heidelberg (2002)
6. Champarnaud, J.M., Nicart, F., Ziadi, D.: Computing the follow automaton of an expression. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 90–101. Springer, Heidelberg (2005)
7. Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Cambridge University Press, Cambridge (2008)
8. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
9. Flajolet, P., Odlyzko, A.M.: The average height of binary trees and other simple trees. J. Comput. Syst. Sci. 25(2), 171–213 (1982)
10. Lee, J., Shallit, J.: Enumerating regular expressions and their languages. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 2–22. Springer, Heidelberg (2005)
11. Fernández-Camacho, M.I., Steyaert, J.M.: Algebraic simplification in computer algebra: An analysis of bottom-up algorithms. TCS 74(3), 273–298 (1990)

Tiling the Plane with a Fixed Number of Polyominoes

Nicolas Ollinger

Laboratoire d'informatique fondamentale de Marseille (LIF),
Aix-Marseille Université, CNRS,
39 rue Joliot-Curie, 13 013 Marseille, France
Nicolas.Ollinger@lif.univ-mrs.fr

Abstract. Deciding whether a finite set of polyominoes tiles the plane is undecidable by reduction from the Domino problem. In this paper, we prove that the problem remains undecidable if the set of instances is restricted to sets of 5 polyominoes. In the case of tiling by translations only, we prove that the problem is undecidable for sets of 11 polyominoes.

Introduction

Tiling the plane given a finite set of tiles is an old and fascinating problem. For an survey on tilings, the reader is invited to consult Grünbaum and Shephard [1]. A celebrated computability result by Berger [2] is the undecidability of the Domino problem: given a finite set of Wang tiles, unit squares with colored edges, decide if the Wang tiles can tile the whole plane so that matching edges share a same color. A polyomino is a simple kind of tile: it consists of rookwise connected unit squares. Golomb [3] studied tiling by polyominoes and proved in [4] that the Domino problem can be reduced to deciding if a finite set of polyominoes tiles the plane. The reduction can be achieved by a classical encoding of Wang tiles by polyominoes that preserves tilings. In this reduction, the number of polyominoes is equal to the initial number of Wang tiles. A natural question arises: what happens if we consider the tiling problem for a fixed number of polyominoes? From this previous result, two cases might happen: (1) the problem is undecidable starting from a certain fixed number of polyominoes (2) the problem is decidable for every fixed number of polyominoes but the family of decision procedures is not itself recursive. As case (1) is more likely to happen, the question is to find the frontier between decidability and undecidability. Such a study of decidability questions with respect to a parameter appears for example in the study of semi-Thue systems or for Post correspondence problem (PCP) where it is shown that $PCP(2)$ is decidable and $PCP(7)$ is undecidable [5,6,7].

Motivated by parallel computing, Wijshoff and van Leeuwen [8] proved that the tilability of the plane by translation of a unique polyomino is decidable. That result was further studied and understood by Beauquier and Nivat [9] who described precisely the tilings by translation generated by a unique polyomino.

This paper is organized as follows. In section 1, we introduce Wang tiles, polyominoes and dented polyominoes, a special variation of polyominoes used in

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 638–647, 2009.

the constructions. In section 2, we prove that it is undecidable whether a set of 5 polyominoes tiles the plane. In section 3, we deduce from previous section that it is undecidable whether a set of 11 polyominoes tiles the plane by translation. In section 4, we discuss the case of smaller sets of tiles.

1 Definitions

Polyominoes. A *polyomino* is a simply connected tile obtained by gluing together rookwise connected unit squares. A *tiling*, of the Euclidian plane, by a set of polyominoes is a partition of the plane such that each element of the partition is the image by an isometry of a polyomino of the set. A *tiling by translation* is a tiling where isometries are restricted to translations. A tiling is *periodic* if it is invariant by translation, *biperiodic* if it is invariant by two non-colinear translations, *aperiodic* if it is not periodic. A set of polyominoes is *aperiodic* if it admits a tiling and all its tilings are aperiodic.

A tiling is *discrete* if all the vertices of the unit squares composing the polyominoes are aligned on the grid $\mathbb{Z}^2$. If a tiling is not discrete, the tiling can be split into two tilings of a half-plane along a line going through an edge along which two unit squares are not aligned. By shifting one half-plane to align the tiles, and iterating this process, one obtains the following lemma.

Lemma 1. *A set of polyominoes admits a tiling if and only if it admits a discrete tiling.*

In this paper where we deal with tilability, we only consider discrete tilings, thanks to this lemma. Thus, a polyomino can be considered as a finite, simply connected, subset of $\mathbb{Z}^2$ and a tiling by a set of polyominoes is a partition of $\mathbb{Z}^2$ where each element is the image by an isometry of an element of the set. Each such isometry can be decomposed into a translation and one out of 8 elementary transformations obtained by composing right angle rotations and mirroring. A sample polyomino and its 8 transformations are represented in Fig. 1.

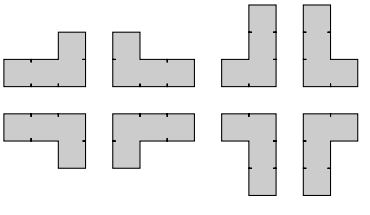
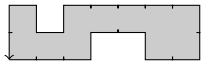


Fig. 1. A polyomino and its 8 transformations

Polyominoes are compactly described by their contour words. A *contour word* of a polyomino is a (finite) word on the alphabet $\{e, w, n, s\}$ describing a walk along the outline of the polyomino starting from and ending to a vertex of the boundary of the polyomino where e is an east move $(1, 0)$, w is a west move

$(-1, 0)$, n is a north move $(0, 1)$ and s is a south move $(0, -1)$. A word is a contour word if and only if the associated path does not cross itself. A polyomino with a pointed contour word is represented in Fig. 2



$e^3ne^2se^2n^2w^5swnws^2$

Fig. 2. A polyomino with a pointed contour word

Wang Tiles. A *Wang tile* is a unit square with colored edges. A tiling by a set of Wang tiles is a discrete tiling by tiles of the set such that along each edge the colors match on both sides. The *Domino problem* is the following decision problem: given a finite set of Wang tiles, decide whether it admits a tiling.

Theorem 1 (Berger [2]). *The Domino problem is undecidable.*

The *Polyomino problem* is the following decision problem: given a finite set of polyominoes, decide whether it admits a tiling. By a reduction from the Domino problem to the Polyomino problem, Golomb [4] proved the undecidability of the Polyomino problem.

Theorem 2 (Golomb [4]). *The Polyomino problem is undecidable.*

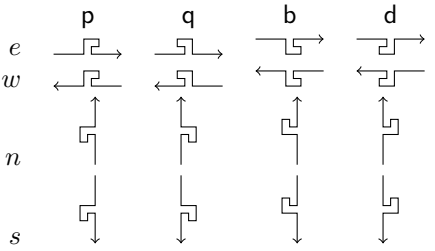
The reduction proceeds as follows. Given a finite set of Wang tiles, Golomb encodes each tile into a big squarish polyomino. Special bumps and dents are added to the corners of the tiles to force both alignment and orientation of the tiles: if one of the encoding polyominoes appears with an orientation, all the other tiles of the tiling have to use the same orientation. Special bumps and dents are used along the sides of the big polyominoes to encode the colors of the Wang tiles. Quotienting the set of tilings of the set of encoding polyominoes by isometries, it is in bijection with the set of tilings of the given set of Wang tiles.

Dented Polyominoes. A *dented polyomino* is a polyomino with edges labeled by a shape and an orientation. The four possible orientations $\{\mathsf{p}, \mathsf{q}, \mathsf{b}, \mathsf{d}\}$ and their interpretation depending on the direction of the edge are depicted on Table 1 for a sample shape. On a contour word, inside shapes define bumps and outside shapes define dents. A tiling by a set of dented polyominoes is a tiling by the corresponding set of polyominoes where bumps and dents match along edges.

Dented polyominoes provide a convenient tool to construct complicated sets of polyominoes. These polyominoes with puzzle bumps and dents can be easily converted into polyominoes.

Lemma 2. *Every finite set of dented polyominoes can be encoded as a finite set of polyominoes such that their sets of tilings are in one-to-one correspondence.*

Table 1. Encoding of bumps and dents orientation



Proof. In order to guarantee that bumps and dents do not interfere with the matching conditions of polyominoes, the idea is to rescale the polyominoes. For all $k \in \mathbb{Z}^+$, a k-rescaling of a set of polyominoes consists into scaling the polyominoes by a factor k, *i.e.*, replacing each unit square by a square of k by k unit squares. Tilings are preserved by rescaling: the set of tilings of a set of polyominoes is in one-to-one correspondence with the set of tilings of its k-rescaling.

To encode a finite set of dented polyominoes into a finite set of polyominoes: first, rescale the set of polyominoes by a factor far bigger than the size of any shape of its bumps and dents; then, add bumps and dents in the middle of each rescaled edge. □

2 Tiling with a Fixed Number of Polyominoes

The k-Polyomino problem is the following decision problem: given a set of k polyominoes, decide whether it admits a tiling. This section is dedicated to the proof of the following theorem.

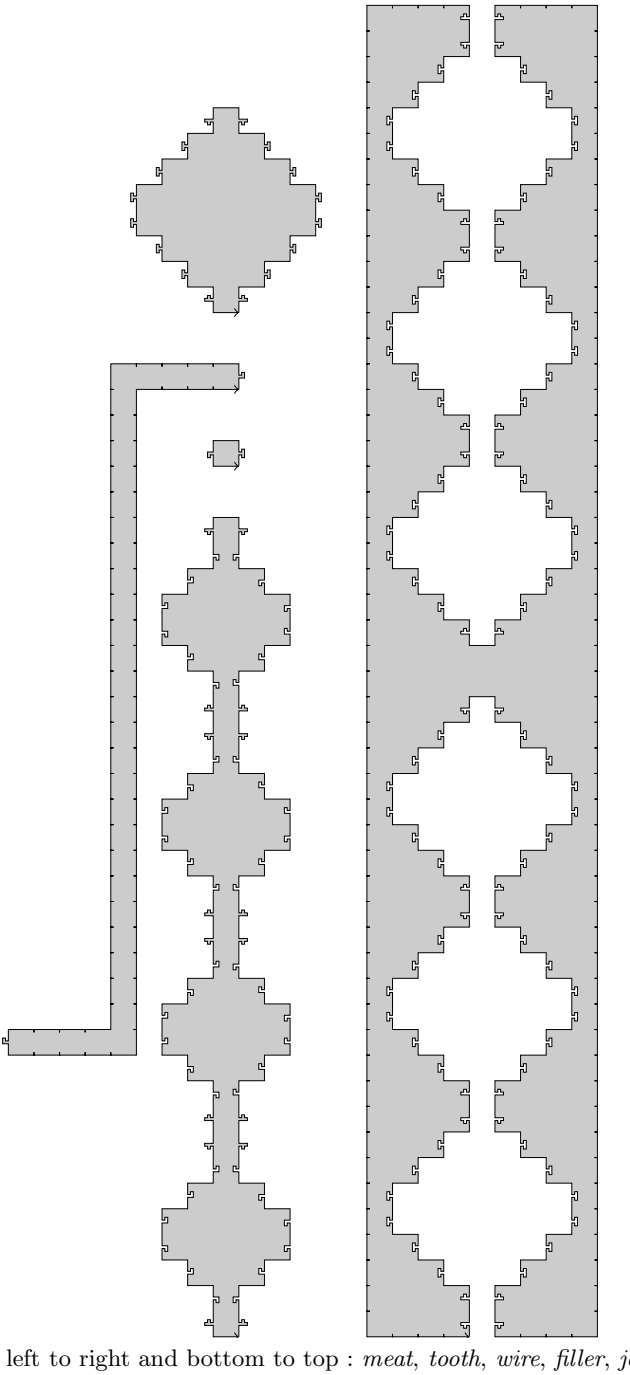
Theorem 3. *The* 5*-Polyomino problem is undecidable.*

We will proceed by reduction of the Domino problem. Given a finite set τ of Wang tiles, we construct a set of 5 dented polyominoes $P(\tau)$ such that, up to isometry, the set of tilings of τ is in one-to-one correspondence with the set of tilings of $P(\tau)$. The proof goes as follows. First, we describe the construction of $P(\tau)$. Then, we explain how to encode any tiling of τ by a tiling of $P(\tau)$. Finally, we show that any tiling of $P(\tau)$ encodes a tiling of τ.

2.1 Encoding a Set of Wang Tiles

Let τ be a set of N Wang tiles. The set of dented polyominoes $P(\tau)$ consists of the following 5 tiles, represented in Fig. 3:

meat encodes all tiles of the set τ sequentially;
jaw acts as a selector to select exactly one tile of the *meat*;
filler is used for padding the blank leaved by the *meat* inside the *jaw*;
tooth erases the bits on the *meat* so that it fits inside the *jaw*;
wire links *meat* pieces together to verify tiling constraints.



From left to right and bottom to top : *meat*, *tooth*, *wire*, *filler*, *jaw*
Notice that here $N = 4$ and $k = 3$ to fit the page (in the text $k > 4$).

Fig. 3. Tiles *(rotated to fit in page)*

More formally, the dented polyominoes use 4 different shapes for bumps and dents, detailed on Table 2.

Table 2. Types of bumps and dents

	blank	*bit*	*marker*	*inside*
shape	———			
notation		b	m	i
order	1	4	4	2
bump		*wire*, *tooth*	*meat*, *filler*	*tooth*, *filler*
dent		*meat*	*jaw*	*jaw*

Let k be a large enough integer and choose an encoding on $k-4$ bits of the colors of the set of Wang tiles (horizontal and vertical colors can use different encodings). Let (a_j^i), (b_j^i), (c_j^i), (d_j^i) be respectively the north, east, south, and west binary encoding of the tiles where i is the tile index from 1 to N and j is the bit index from 1 to $k-4$. Let $(\mathfrak{a}_j^i)$ be the encoding of the k bits, by adding prefix 00 and suffix 01, of (a_j^i) on the alphabet $\{\mathfrak{b},\mathfrak{d}\}$. Let $(\mathfrak{b}_j^i)$ be the encoding of the k bits, by adding prefix 10 and suffix 11, of (b_j^i) on the alphabet $\{\mathfrak{p},\mathfrak{q}\}$. Let $(\mathfrak{c}_j^i)$ be the encoding of the k bits, by adding prefix 00 and suffix 01, of (c_j^i) on the alphabet $\{\mathfrak{b},\mathfrak{d}\}$. Let $(\mathfrak{d}_j^i)$ be the encoding of the k bits, by adding prefix 10 and suffix 11, of (d_j^i) on the alphabet $\{\mathfrak{p},\mathfrak{q}\}$. The dented polyominoes are given by their contour words on Table 3.

Table 3. Contour encoding of the tiles

tooth: $e_i^{\mathfrak{b}} n w_b^{\mathfrak{q}} s$

wire: $e_b^{\mathfrak{b}} n^5 w^{2N(k+1)+1} n^4 w_b^{\mathfrak{p}} s^5 e^{2N(k+1)+1} s^4$

filler: $e_m^{\mathfrak{b}} \left(s e_i^{\mathfrak{b}}\right)^k \left(e_i^{\mathfrak{b}} n\right)^k e_m^{\mathfrak{d}} n w_m^{\mathfrak{q}} \left(n w_i^{\mathfrak{p}}\right)^k \left(w_i^{\mathfrak{p}} s\right)^k w_m^{\mathfrak{p}} s$

jaw: $e_m^{\mathfrak{q}} \left(e_m^{\mathfrak{p}} \left(n e_i^{\mathfrak{p}}\right)^k \left(e_i^{\mathfrak{p}} s\right)^k e_m^{\mathfrak{q}}\right)^{N-1} s \left(w_m^{\mathfrak{d}} \left(s w_i^{\mathfrak{d}}\right)^k \left(w_i^{\mathfrak{d}} n\right)^k w_m^{\mathfrak{b}}\right)^{N-1} w_m^{\mathfrak{d}} s^4 e^{2(N-1)(2k+2)+4} n^4$
$w_m^{\mathfrak{b}} \left(w_m^{\mathfrak{d}} \left(s w_i^{\mathfrak{d}}\right)^k \left(w_i^{\mathfrak{d}} n\right)^k w_m^{\mathfrak{b}}\right)^{N-1} n \left(e_m^{\mathfrak{p}} \left(n e_i^{\mathfrak{p}}\right)^k \left(e_i^{\mathfrak{p}} s\right)^k e_m^{\mathfrak{q}}\right)^{N-1} e_m^{\mathfrak{p}} n^4 w^{2(N-1)(2k+2)+4} s^4$

meat: $\prod_{i=1}^{N} \left(e_m^{\mathfrak{b}} \prod_{j=1}^{k-1} \left(e_b^{\mathfrak{a}_j^i} s\right) e_b^{\mathfrak{a}_k^i} \prod_{j=1}^{k-1} \left(e_b^{\mathfrak{b}_j^i} n\right) e_b^{\mathfrak{b}_k^i} e_m^{\mathfrak{d}}\right) n$
$\prod_{i=N}^{1} \left(w_m^{\mathfrak{q}} \prod_{j=k}^{2} \left(w_b^{\mathfrak{c}_j^i} n\right) w_b^{\mathfrak{c}_1^i} \prod_{j=k}^{2} \left(w_b^{\mathfrak{d}_j^i} s\right) w_b^{\mathfrak{d}_1^i} w_m^{\mathfrak{p}}\right) s$

Notice the following important properties of these tiles. The *meat* consists of a sequence of N diamonds decorated with *bit* shapes with a prefix and a suffix *marker* shape pointing to the diamond. If one connects a *tooth* in each *bit* dent of a diamond of the *meat*, the diamond becomes similar to the *filler*. Moreover, both inside parts of a *jaw* consists of $N-1$ places to put a *filler* plus a *marker* shape at the entry of the *jaw*, pointing outside.

2.2 Encoding Tilings by Wang Tiles

Let us first prove the following lemma.

Lemma 3. *Every tiling by τ can be encoded as a tiling by $P(\tau)$.*

The set of dented polyominoes is designed to encode a Wang tile by selecting one diamond of a *meat*, hiding the other diamonds using two *jaws* on the left and the right, padding inside the *jaws* with *teeth* and *fillers*, as represented in Fig. 4. The colors of the tile are propagated to the four neighbor tiles using *wires*.

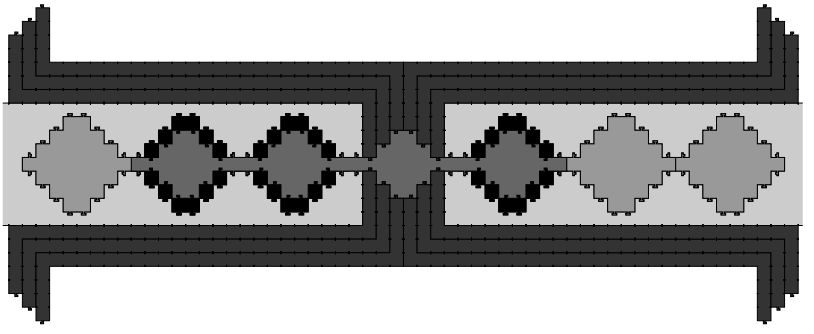


Fig. 4. Encoding of a Wang tile including inter-tiles wires

Let $T \in \tau^{\mathbb{Z}^2}$ be a tiling by τ. Each north-west diagonal ($x + y = i$ for the ith diagonal) is encoded as a line of Wang tile encodings where a *jaw* connects tile $T(x, i - x)$ to tile $T(x + 1, i - x - 1)$. These lines of encoding are put on top of each other with a slight translation so that tile $T(x, i - x)$ is connected by *wires* to $T(x + 1, i - x)$, $T(x, i - x + 1)$, $T(x - 1, i - x)$ and $T(x, i - x - 1)$, as represented in Fig. 5. Notice that the choices made for $\mathfrak{a}$, $\mathfrak{b}$, $\mathfrak{c}$, and $\mathfrak{d}$ permits such a connection only if the Wang tiling is valid. Thus, one obtains a tiling of the dented polyominoes.

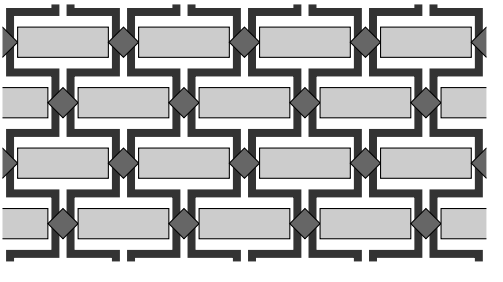


Fig. 5. Wiring of Wang tiles

2.3 Every Tiling Encodes a Tiling

Now, we prove the following lemma.

Lemma 4. *Every tiling by $P(\tau)$ encodes a tiling by τ.*

Consider a tiling by the dented polyominoes. We first show that it must contain a *jaw*. Consider any tile of the tiling. If it is a *jaw*, we are done. Examine Table 2. If it is a *meat* or a *filler*, it has a *marker* bump that should be linked to a dent only found on a *jaw*. If it is a *tooth*, it has an *inside* bump that should be linked to a dent only found on a *jaw*. Finally, if it is a *wire*, it has *bit* bump that should be linked to a *meat*, itself connected to a *jaw*.

Consider a *jaw* tile of the tiling. To fill all the *marker* bumps, only *filler* and *meat* tiles can be used. As *fillers* have *inside* bumps, they can only be used completely inside the *jaw*. Thus, the *markers* on the extremities of the *jaw* have to be filled by the dents of a *meat*. Consider the *meat* that fills a *marker* at the extremity of the *jaw*. As *marker* bumps only appear inside *jaws*, the *marker* dent on the other side of the diamond of the *meat* next to the *jaw* has to be at the extremity of a next *jaw*. The only possibility to fill the gap in between the *jaw* and the *meat* locking its extremity is to use *fillers* and *teeth*. By now, we have proved that each *jaw* of the tiling appears in a biinfinite line (or column if rotated) of alternating *jaws* and *meats* where each *meat* has exactly one selected diamond outside the *jaws*.

Consider now the diamond of each *meat* appearing outside the *jaws*: it has *bit* bumps. A *bit* dent is found only on *teeth* and *wires*. As a *tooth* cannot appear outside a *jaw* (it as an *inside* dent), only *wires* can be connected to these bumps. Consider the two *bit* bumps side by side at the top of the diamond hill. The only possibility for two *wires* to appear side by side is to have the left one point to the left and the right one point to the right. This enforces all the *wires* in between the *jaw* and the top hill *wire* to point in the same direction: left ones to the left and right ones to the right. Thus, every tiling by dented polyominoes consists of biinfinite lines (or columns) of selected *meat* diamonds connected by *wires* in a lattice way as on Fig. 5.

Due to isometries, it remains to prove that all the selected diamonds have the same orientation. This part is enforced by the prefix/suffix trick in the $\mathfrak{a}$, $\mathfrak{b}$, $\mathfrak{c}$, and $\mathfrak{d}$ encoding: the only possibility for a diamond to be connected to another diamond is that prefix and suffix match, thus both should be oriented in the same way. Thus, the tiling is the image by an isometry of a tiling by Wang tiles. We have proved that every tiling by $P(\tau)$ encodes a tiling by τ, achieving the reduction.

3 Tiling by Translation

The k-Polyomino translation problem is the following decision problem: given a set of k polyominoes, decide if it admits a tiling by translation. Using the result of previous section, one obtains the following.

Theorem 4. *The* 11*-Polyomino translation problem is undecidable.*

To prove this theorem, for any finite set of Wang tiles τ, construct a set of 11 polyominoes as follows. Consider the set of 5 dented polyominoes $P(\tau)$. To encode any tiling by τ into a tiling by $P(\tau)$ as done in the previous section, we use exactly:

1 transformation of *meat*;
1 transformation of *jaw*;
1 transformation of *filler*;
4 transformations of *wire*;
4 transformations of *tooth*.

The set of 11 polyominoes tiling by translation consists exactly of these tile transformations. These dented polyominoes admit a tiling if and only if the set of Wang tiles admits a tiling.

4 Going Further

What can be said about tilability for sets of less that 5 polyominoes? or less than 11 polyominoes for tilings by translation? In the case of 1 polyomino, it is decidable for tiling by translation and still open for tiling with isometries.

Theorem 5 (Wijshoff and van Leeuwen [8], Gambini and Vuillon [10]). *The* 1*-Polyomino translation problem is decidable in time quadratic in the size of the contour word.*

Open Problem 1. *Is the* 1*-Polyomino problem decidable?*

To prove the undecidability of the Polyomino problem, one has to be able to construct aperiodic sets of polyominoes. Ammann *et al* provide a set of 2 polygonal tiles with colors and 3 polygonal tiles with bumps and dents that admits only aperiodic tilings. This set is convertible into polyominoes.

Theorem 6 (Ammann *et al.* [11]). *There exists an aperiodic set of* 3 *polyominoes.*

Theorem 7 (Ammann *et al.* [11]). *There exists an aperiodic set of* 8 *polyominoes for tiling by translation.*

Open Problem 2. *Is the* k*-Polyomino problem decidable for* $3 \leqslant k < 5$*?*

Open Problem 3. *For* $8 \leqslant k < 11$*, is the* k*-Polyomino translation problem decidable?*

Acknowledgement

The author thanks Bruno Durand for challenging him with the decision problem of tiling the plane with a fixed number of polyominoes.

References

1. Grünbaum, B., Shephard, G.C.: Tilings and patterns. A Series of Books in the Mathematical Sciences. W. H. Freeman and Company, New York (1989)
2. Berger, R.: The undecidability of the domino problem. Memoirs American Mathematical Society 66 (1966)
3. Golomb, S.W.: Tiling with polyominoes. Journal of Combinatorial Theory 1(2), 280–296 (1966)
4. Golomb, S.W.: Tiling with sets of polyominoes. Journal of Combinatorial Theory 9(1), 60–71 (1970)
5. Post, E.L.: A variant of a recursively unsolvable problem. Bulletin of the American Mathematical Society 52, 264–268 (1946)
6. Claus, V.: Some remarks on PCP(k) and related problems. Bulletin of the EATCS 12, 54–61 (1980)
7. Matiyasevich, Y., Sénizergues, G.: Decision problems for semi-thue systems with a few rules. Theoretical Computer Science 330(1), 145–169 (2005)
8. Wijshoff, H.A.G., van Leeuwen, J.: Arbitrary versus periodic storage schemes and tessellations of the plane using one type of polyomino. Information and Control 62(1), 1–25 (1984)
9. Beauquier, D., Nivat, M.: On translating one polyomino to tile the plane. Discrete and Computational Geometry 6(1), 575–592 (1991)
10. Gambini, I., Vuillon, L.: An algorithm for deciding if a polyomino tiles the plane. Theoretical Informatics and Applications 41(2), 147–155 (2007)
11. Ammann, R., Grünbaum, B., Shephard, G.: Aperiodic tiles. Discrete and Computational Geometry 8(1), 1–25 (1992)

New Morphic Characterizations of Languages in Chomsky Hierarchy Using Insertion and Locality

Kaoru Onodera

Department of Information Sciences, School of Science and Engineering, Tokyo Denki University, Ishizaka, Hatoyama-machi, Hiki-gun, Saitama 350-0394, Japan
kaoru@j.dendai.ac.jp

Abstract. In this paper, we obtain some characterizations and representation theorems of languages in Chomsky hierarchy by using insertion systems, strictly locally testable languages, and morphisms. For instance, each recursively enumerable language L can be represented in the form $L = h(L(\gamma) \cap R)$, where γ is an insertion system of weight $(3, 3)$, R is a strictly 2-testable language, and h is a projection. A similar representation can be obtained for context-free languages, using insertion systems of weight $(3, 0)$ and strictly 4-testable languages, as well as for regular languages, using insertion systems of weight $(1, 0)$ and strictly 2-testable languages.

1 Introduction

DNA computing theory involves the use of *insertion* and *deletion operations*. It has been shown that by using insertion and deletion operations, any recursively enumerable language can be characterized in [1], [2].

Insertion systems in which we can use only insertion operations are somewhat intermediate between Chomsky context-sensitive grammars (nonterminal symbols are rewritten according to specified contexts) and Marcus contextual grammars (contexts are adjoined to specified strings associated with contexts). From the definition of insertion operations, one would easily imagine that by using only insertion operations, we generate only context-sensitive languages.

On the other hand, the class of strictly locally testable languages is known as a proper subclass of regular language classes [3]. The equivalence relation between a certain type of splicing languages and strictly locally testable languages is known [4].

We focus on characterizing recursively enumerable languages by using insertion systems together with some "additional mechanisms". It has been shown that using insertion systems together with some morphisms, characterizing recursively enumerable languages is accomplished in [2], [5], [6]. Each recursively enumerable language L can be represented in a similar way to the well-known Chomsky-Schützenberger representation of context-free languages, $L = h(L(\gamma) \cap D)$, where γ is an insertion system, h is a projection, and D is a Dyck language [7]. In this paper, we use strictly locally testable languages and morphisms as the additional mechanisms for characterizing recursively enumerable languages.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 648–659, 2009.

In insertion systems, a pair of the maximum length of inserted strings and the one of context-checking strings, called *weight* is an important parameter for generative powers. As for strictly locally testable languages, the length of local testability-checking is considered.

We prove that each recursively enumerable language can be represented in the form $h(L(\gamma) \cap R)$, where γ is an insertion system of weight $(3, 3)$, h is a morphism, and R is a strictly 2-testable language. The similar characterizations are shown for context-free and regular languages. The optimality of these results, in terms of weight in insertion operations and the parameter of strictly locally testable languages remains to be checked.

2 Preliminaries

In this section, we introduce necessary notation and basic definitions needed in this paper. We assume the reader to be familiar with the rudiments on basic notions in formal language theory (see, e.g., [2,8]).

2.1 Basic Definitions

For an alphabet V, V^* is the set of all strings of symbols from V which includes the empty string λ. For a string $x \in V^*$, $|x|$ denotes the length of x. For $0 \leq k \leq |x|$, let $Pre_k(x)$ and $Suf_k(x)$ be the prefix and the suffix of x of length k, respectively. For $0 \leq k \leq |x|$, let $Int_k(x)$ be the set of *proper* interior substrings of x of length k, while if $|x| = k$ then $Int_k(x) = \emptyset$.

2.2 Normal Forms of Grammars

A *phrase structure grammar* is a quadruple $G = (N, T, P, S)$, where N is a set of *nonterminal symbols*, T is a set of *terminal symbols*, P is a set of *production rules*, and S in N is the *initial symbol*. A rule in P is of the form $r : \alpha \to \beta$, where $\alpha \in (N \cup T)^* N (N \cup T)^*$, $\beta \in (N \cup T)^*$, and r is a label from a given set $Lab(P)$ such that there are no production rules with the same label. For any x and y in $(N \cup T)^*$, if $x = u\alpha v$, $y = u\beta v$, and $r : \alpha \to \beta \in P$, then we write $x \overset{r}{\Longrightarrow}_G y$. We say that x *directly derives* y with respect to G. If there is no confusion, we write $x \Longrightarrow y$. The reflexive and transitive closure of $\Longrightarrow$ is denoted by $\Longrightarrow^*$.

We define a *language* $L(G)$ generated by a grammar G as follows:

$$L(G) = \{w \in T^* \mid S \Longrightarrow_G^* w\}.$$

It is well known that the class of languages generated by the phrase structure grammars is equal to the class of *recursively enumerable languages RE* [8].

A grammar $G = (N, T, P, S)$ is *context-free* if P is a finite set of *context-free rules* of the form $A \to \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$. A language L is a *context-free language* if there is a context-free grammar G such that $L = L(G)$. Let CF be the class of context-free languages.

A context-free grammar $G = (N, T, P, S)$ is in *Chomsky normal form* if each production rule in P is of one of the following forms:

1. $X \to YZ$, where $X, Y, Z \in N$.
2. $X \to a$, where $X \in N$, $a \in T$.
3. $S \to \lambda$ (only if S does not appear in right-hand sides of production rules).

It is well known that, for each context-free language L, there is a context-free grammar in Chomsky normal form generating L [8].

A grammar $G = (N, T, P, S)$ is *regular* if P is a finite set of rules of the form $X \to \alpha$, where $X \in N$ and $\alpha \in TN \cup T \cup \{\lambda\}$. A language L is a *regular language* if there is a regular grammar G such that $L = L(G)$. Let REG be the class of regular languages.

We are going to define a strictly locally testable language, which is one of the main objectives of the present work.

Let k be a positive integer. A language L over T is strictly k-testable if there is a triplet $S_k = (A, B, C)$ with $A, B, C \subseteq T^k$ such that, for any w with $|w| \geq k$, w is in L iff $Pre_k(w) \in A$, $Suf_k(w) \in B$, $Int_k(w) \subseteq C$.

Note that if L is strictly k-testable, then L is strictly k'-testable for all $k' > k$. Further, the definition of strictly k-testable says nothing about the strings of "length $k-1$ or less".

A language L is *strictly locally testable* iff there exists an integer $k \geq 1$ such that L is strictly k-testable. Let $LOC(k)$ be the class of strictly k-testable languages. Then one can prove the following theorem.

Theorem 1. [9] $LOC(1) \subset LOC(2) \subset \cdots \subset LOC(k) \subset \cdots \subset REG$.

We are now going to define an insertion system. An *insertion system* (*ins system*, for short) is a triple $\gamma = (T, P, A_X)$, where T is an alphabet, P is a finite set of *insertion rules* of the form (u, x, v) with $u, x, v \in T^*$, and A_X is a finite set of strings over T called axioms.

We write $\alpha \overset{r}{\Longrightarrow}_\gamma \beta$ if $\alpha = \alpha_1 u v \alpha_2$ and $\beta = \alpha_1 u x v \alpha_2$ for some insertion rule $r : (u, x, v) \in P$ with $\alpha_1, \alpha_2 \in T^*$. If there is no confusion, we write $\alpha \Longrightarrow \beta$. The reflexive and transitive closure of $\Longrightarrow$ is defined by $\Longrightarrow^*$.

A language generated by γ is defined by

$$L(\gamma) = \{w \in T^* \mid \ s \Longrightarrow^*_\gamma w, \text{ for some } s \in A_X\}.$$

An insertion system $\gamma = (T, P, A_X)$ is said to be of *weight* (m, n) if

$$\begin{aligned} m &= max\{\ |x| \mid (u, x, v) \in P\}, \\ n &= max\{\ |u| \mid (u, x, v) \in P \text{ or } (v, x, u) \in P\}. \end{aligned}$$

For $m, n \geq 0$, INS^n_m denotes the class of all languages generated by ins systems of weight (m', n') with $m' \leq m$ and $n' \leq n$. When the parameter is not bounded, we replace m or n with $*$.

For insertion systems, there exist the following results.

Theorem 2. [2]

1. $FIN \subset INS^0_* \subset INS^1_* \cdots \subset INS^*_* \subset CS$.
2. $REG \subset INS^*_*$.

3. $INS_*^1 \subset CF$.
4. *CF is incomparable with all* INS_*^n $(n \geq 2)$, *and* INS_*^*.
5. INS_2^2 *contains non-semilinear languages.*

From the definition of insertion systems, we can easily prove the following lemma.

Lemma 1. $INS_1^0 \subset REG$.

A mapping $h : V^* \to T^*$ is called *morphism* if $h(\lambda) = \lambda$ and $h(xy) = h(x)h(y)$ for any $x, y \in V^*$. For languages L_1, L_2, and morphism h, we use the following notation: $h(L_1 \cap L_2) = \{h(w) \mid w \in L_1 \cap L_2\}$. For language classes $\mathcal{L}_1$ and $\mathcal{L}_2$, we introduce the following class of languages:

$$H(\mathcal{L}_1 \cap \mathcal{L}_2) = \{h(L_1 \cap L_2) \mid h \text{ is a morphism}, L_i \in \mathcal{L}_i \ (i = 1, 2)\}.$$

3 Characterizations of Regular Languages

In this section, we will characterize regular languages in terms of insertion languages and strictly locally testable languages both of which form proper subclasses of regular languages.

Lemma 2. $REG \subseteq H(INS_1^0 \cap LOC(2))$.

Proof. For a regular language L, let $G = (N, T, P, S)$ be a regular grammar such that $L = L(G)$. Using the new symbol F, we construct the insertion system $\gamma = (V, P', \{\lambda\})$ of weight $(1, 0)$, where

$$\begin{aligned} V &= \{X_r \mid r : X \to \alpha \in P\} \cup \{F\}, \\ P' &= \{(\lambda, X, \lambda) \mid X \in V\}. \end{aligned}$$

Then, $L(\gamma) = V^*$.

Further, we define the morphism $h : V^* \to T^*$ by

$$\begin{aligned} h(X_r) &= a \quad \text{if } r : X \to aY \in P \text{ or } r : X \to a \in P, \\ h(X_r) &= \lambda \quad \text{if } r : X \to \lambda \in P, \\ h(F) &= \lambda. \end{aligned}$$

Finally, consider $R = AV^* \cap V^*B - V^+C'V^+$ with $C' = V^2 - C$, where

$$\begin{aligned} A = {} & \{S_rX_{r_1} \mid r : S \to aX \in P,\ r_1 : X \to \alpha \in P,\ \alpha \in T \cup TN \cup \{\lambda\}\} \cup \\ & \{S_rF \mid r : S \to \alpha \in P,\ \alpha \in T \cup \{\lambda\}\}, \\ B = {} & \{X_rF \mid r : X \to a \in P \text{ or } r : X \to \lambda \in P\}, \\ C = {} & \{X_rY_{r_1} \mid r : X \to aY \in P,\ r_1 : Y \to \alpha \in P,\ \alpha \in T \cup TN \cup \{\lambda\}\}. \end{aligned}$$

Then R is a strictly 2-testable language prescribed by $S_2 = (A, B, C)$.

We can prove that, for any $X \in N$, $X \overset{r_1}{\Longrightarrow}_G \cdots \overset{r_{n-1}}{\Longrightarrow}_G w'Y \overset{r_n}{\Longrightarrow}_G w'y = w \in T^*$ iff $X_{r_1} \cdots Y_{r_n}F \in V^*B - V^*C'V^+$ with $h(X_{r_1} \cdots Y_{r_n}F) = w$ by the induction on n. We omit the proof here.

Considering the special case $X = S$, then a string w is in $L(G)$ iff w is in $h(L(\gamma) \cap R)$. □

Lemma 3. $H(INS_1^0 \cap LOC(2)) \subseteq REG$.

Proof. Since the class of regular languages is closed under intersection with regular languages and morphisms, the result follows from the facts that $INS_1^0 \subset REG$ in Lemma 1 and $LOC(2) \subset REG$ in Theorem 1. □

From Lemma 2 and Lemma 3, we have the following theorem.

Theorem 3. $REG = H(INS_1^0 \cap LOC(2))$.

Since for arbitrary k with $k \geq 2$, the class of regular languages includes the class of strictly k-testable languages, the next result follows from Theorem 3 and Theorem 1.

Corollary 1. $REG = H(INS_1^0 \cap LOC(k))$ ($k \geq 2$).

The value of parameter $k = 2$ in the strictly k-testable languages in Theorem 3 is necessary for expressing regular languages in the following sense.

Lemma 4. *There exists a regular language which cannot be written in the form* $h(L(\gamma) \cap R)$*, for any insertion system* γ *of weight* $(i, 0)$ $(\forall i \geq 1)$*, strictly* 1*-testable language* R*, and morphism* h*.*

Proof. Consider the regular language $L = \{a^l \mid l \geq 0\} \cup \{b^l \mid l \geq 0\}$. Suppose that there is an insertion system $\gamma = (V, P, A_X)$ of weight $(i, 0)$ with $i \geq 1$, a strictly 1-testable language R prescribed by $S_1 = (A, B, C)$, and a morphism h such that $L = h(L(\gamma) \cap R)$.

Then, for any $l \geq 0$, there exists the set of strings $D_l = \{x \mid h(x) = a^l\} \cup \{y \mid h(y) = b^l\}$ such that $D_l \subset L(\gamma) \cap R$. Let $D = \bigcup_{l \geq 0} D_l$, then D is an infinite set. Since $D \subset L(\gamma) \cap R$ holds, $L(\gamma) \cap R$ is also an infinite set. Then P includes both (λ, u_a, λ) and (λ, u_b, λ), where $u_a, u_b \in C^i$, $h(u_a) = a^{i_a}$, $h(u_b) = b^{i_b}$ for some $i_a, i_b > 0$. Let $t_1 x t_2$ and $t_3 y t_4$ be in $L(\gamma) \cap R$ with $t_1, t_3 \in A$, $t_2, t_4 \in B$, $u_a \in Int_i(t_1 x t_2)$, $u_b \in Int_i(t_3 y t_4)$.

Then, the string $t_1 u_b x t_2$ is in $L(\gamma) \cap R$ satisfying $|h(t_1 u_b x t_2)|_a \geq i_a > 0$ and $|h(t_1 u_b x t_2)|_b \geq i_b > 0$, which contradicts to the fact that $L = \{a^l \mid l \geq 0\} \cup \{b^l \mid l \geq 0\}$. □

From Lemma 4, Theorem 1, and Theorem 3, we have the following theorem.

Theorem 4. $H(INS_1^0 \cap LOC(1)) \subset REG$.

The value of weight $(1, 0)$ in insertion systems in Theorem 3 is optimal for expressing regular languages in the following sense.

Lemma 5. *There exist an insertion system* γ *of weight* $(2, 0)$*, a strictly* 1*-testable language* R*, and a morphism* h *such that* $h(L(\gamma) \cap R)$ *is non-regular.*

Proof. Consider an insertion system $\gamma = (T, \{\lambda\}, \{(\lambda, ab, \lambda)\})$ with $T = \{a, b\}$. Then, for any w in $L(\gamma)$, $|w|_a = |w|_b$ holds.

Consider $R = AT^* \cap T^*B - T^+C'T^+$ with $C' = T - C$, where $A = B = C = T$. Then $R = T^+$ is a strictly 1-testable language prescribed by $S_1 = (T, T, T)$. Further, we define a morphism $h : T^* \to T^*$ by $h(c) = c$ for any $c \in T$. Then, we have $L(\gamma) \cap R = h(L(\gamma) \cap R) = \{w \mid w \in L(\gamma),\ w \neq \lambda\}$.

For a regular language $R_* = \{a^i b^j \mid i, j \geq 1\}$, $h(L(\gamma) \cap R) \cap R_* = \{a^i b^i \mid i \geq 1\}$, which is not regular. From the fact that the class of regular languages is closed under intersection with regular languages, $h(L(\gamma) \cap R)$ is not regular. □

From Lemma 4, Lemma 5, and the fact that $INS_i^0 \subseteq INS_{i+1}^0$ with $i \geq 1$, we have the following corollary.

Corollary 2. *REG and* $H(INS_i^0 \cap LOC(1))$ *are incomparable* ($i \geq 2$).

From Lemma 5, Theorem 2, and Theorem 1, we have the following corollary.

Corollary 3. $REG \subset H(INS_i^0 \cap LOC(k))$ ($i \geq 2$, $k \geq 2$).

4 Characterizations of Context-Free Languages

We will show how context-free languages can be characterized by insertion systems and strictly locally testable languages. In some respect the proof technique from [7] might be helpful to follow the main proof of this section.

Lemma 6. $CF \subseteq H(INS_3^0 \cap LOC(4))$.

Proof. Consider a context-free grammar $G = (N, T, P, S)$ in Chomsky normal form. We construct an insertion system $\gamma = (\Sigma, P_\gamma, \{S\})$, where

$$\begin{aligned}
\Sigma &= V \cup \bar{V} \cup T \\
V &= N \cup Lab(P), \\
P_\gamma &= \{(\lambda, YZr, \lambda), (\lambda, \bar{X}\bar{r}, \lambda) \mid r : X \to YZ \in P\} \cup \\
&\quad \{(\lambda, ar, \lambda), (\lambda, \bar{X}\bar{r}, \lambda) \mid r : X \to a \in P\} \cup \\
&\quad \{(\lambda, r, \lambda), (\lambda, \bar{S}\bar{r}, \lambda) \mid r : S \to \lambda \in P\}.
\end{aligned}$$

For the rule $r : X \to \alpha$ in P, we say that the two insertion rules $(\lambda, \alpha r, \lambda)$ and $(\lambda, \bar{X}\bar{r}, \lambda)$ in P_γ are *r-pair*.

We define the projection $h : \Sigma^* \to T^*$ by

$$\begin{aligned}
h(a) &= a \quad \text{for all } a \in T, \\
h(a) &= \lambda \quad \text{otherwise.}
\end{aligned}$$

Consider $R = A\Sigma^* \cap \Sigma^* B - \Sigma^+ C' \Sigma^+$ with $C' = \Sigma^4 - C$, where

$$\begin{aligned}
A &= \{arX\bar{X} \mid r : X \to a \in P\} \cup \{rS\bar{S}\bar{r} \mid r : S \to \lambda \in P\}, \\
B &= \{rS\bar{S}\bar{r} \mid r : S \to \alpha \in P,\ \alpha \in (N \cup T)^*\}, \\
C &= \{rX\bar{X}\bar{r},\ X\bar{X}\bar{r}a,\ X\bar{X}\bar{r}r_1,\ \bar{X}\bar{r}ar_1,\ \bar{X}\bar{r}r_1Y,\ \bar{r_1}arX,\ \bar{r_1}rX\bar{X},\ arX\bar{X} \mid \\
&\quad r : X \to \alpha \in P,\ r_1 : Y \to \alpha_1 \in P,\ a \in T,\ \alpha, \alpha_1 \in (N \cup T)^*\}.
\end{aligned}$$

Then R is a strictly 4-testable language prescribed by $S_4 = (A, B, C)$. The language R can be characterized by using

$$\Omega = \{rX\bar{X}\bar{r} \mid r : X \to \alpha \in P,\ \alpha \in (N \cup T)^*\}$$

such that $R \subset (\Omega \cup T\Omega)^*(B \cup TB)$. A nonterminal symbol X in $rX\bar{X}\bar{r} \in \Omega$ is said to be *Ω-blocked*. A symbol in $N \cup T$ which is not Ω-blocked is said to be *unblocked*. Intuitively, an Ω-blocked nonterminal symbol X in $rX\bar{X}\bar{r}$ means that X has been used for the rule r.

Further, based on γ and R, we define the followings: for each $X \in N$, let

$$\gamma_X = (\Sigma, P_\gamma, \{X\})$$

be an insertion grammar, and let

$$R_X = A\Sigma^* \cap \Sigma^* B_X - \Sigma^+ C' \Sigma^*$$

be a strictly 4-testable language, where $B_X = \{rX\bar{X}\bar{r} \mid r : X \to \alpha \in P,\ \alpha \in (N \cup T)^*\}$. There is a slight note on the form of $\Sigma^+C'\Sigma^*$ in R_X. Then R_X can be characterized by $R_X \subset (\Omega \cup T\Omega)^*$. For the case $X = S$, $\gamma_S = \gamma$ and $B_S = B$ hold.

We can prove that, for any X in N, if there is a derivation $X \overset{r_1 \cdots r_n}{\Longrightarrow}_G a_1 \cdots a_l$ with $a_i \in T$ $(1 \le i \le l)$ then there is a string

- $w = a_1 u_1 \cdots a_l u_l$ in $L(\gamma_X) \cap R_X$,
  where $l \ge 2$, $u_i \in \Omega^+$ $(1 \le i \le l-1)$, and $u_l \in \Omega^*\{r_1 X \bar{X} \bar{r}_1\}$, or
- $w = a_1 u_1$ in $L(\gamma_X) \cap R_X$,
  where $u_1 \in \Omega^*\{r_1 X \bar{X} \bar{r}_1\}$

by induction on the length n of derivations in G. We omit the proof here.

Conversely, we will show that, for nonempty string w in $L(\gamma) \cap R$, $h(w)$ is in $L(G)$. We start by showing that if a string w is in $L(\gamma_X) \cap R_X$, then there is a derivation $X \Longrightarrow_G^* h(w)$.

Suppose that w is in $L(\gamma_X) \cap R_X$. In order to derive the string w in $R_X \subset (\Omega \cup T\Omega)^*$, without loss of generality, we may assume here that, for each r in P, the first two steps in γ_X are performed by r-pair insertion rules. For the derivation $X \Longrightarrow_{\gamma_X}^{2n} w$, we can prove the claim by induction on n. We omit the proof here. For the case $X = S$, if there is a nonempty string $w \in L(\gamma) \cap R$, then a string $h(w)$ is in $L(G)$.

Finally let us consider the case that λ is in $L(G)$. Since G is in Chomsky normal form, λ is in $L(G)$ if and only if for λ there is a derivation $S \overset{r}{\Longrightarrow} \lambda$. By the construction of P_γ and R, the string λ is in $L(G)$ if and only if (λ, r, λ), $(\lambda, \bar{S}\bar{r}, \lambda) \in P_\gamma$ and $rS\bar{S}\bar{r} \in A \cap B$. Then there is a derivation $S \Longrightarrow_\gamma S\bar{S}\bar{r} \Longrightarrow_\gamma rS\bar{S}\bar{r} \in A \cap B$. From the definition of h, $h(rS\bar{S}\bar{r}) = \lambda$. Therefore, λ is in $L(G)$ if and only if λ is in $h(L(\gamma) \cap R)$. □

It is known that the class of context-free languages includes the class of insertion languages of weight $(3, 0)$ [10]. Together with the fact that the class of context-free languages is closed under intersection with regular languages and morphisms, we have the following lemma.

Lemma 7. $H(INS_3^0 \cap LOC(4)) \subseteq CF$.

From Lemma 6 and Lemma 7, we have the following theorem.

Theorem 5. $CF = H(INS_3^0 \cap LOC(4))$.

Furthermore, from the fact that for arbitrary k with $k \geq 1$, the class of regular languages includes the class of strictly k-testable languages, we have the following corollary.

Corollary 4. $CF = H(INS_3^0 \cap LOC(k))$ ($k \geq 4$).

5 Characterizations of RE Languages

In this section, we will show that any recursively enumerable language can be represented by using insertion systems and strictly locally testable languages in the similar way to context-free and regular languages.

Theorem 6. $RE = H(INS_3^3 \cap LOC(2))$.

Construction of an insertion system γ: Let $G = (N, T, P, S)$ be a type-0 grammar in Penttonen normal form. In this normal form, the rules in P are of the following types:

$$\begin{array}{l} \text{Type 1}: \ X \to \alpha \in P, \text{ where } X \in N, \ \alpha \in (N \cup T)^*, \ |\alpha| \leq 2. \\ \text{Type 2}: \ XY \to XZ \in P, \text{ where } X, Y, Z \in N. \end{array}$$

By introducing new symbols # and c, we construct the insertion system $\gamma = (\Sigma, P_\gamma, \{Scc\})$, where $\Sigma = N \cup T \cup \{\#, c\}$ and P_γ contains the following insertion rules:

- Group 1: For each rule $r : X \to YZ \in P$ of Type 1, with $X \in N$ and $Y, Z \in N \cup T \cup \{\lambda\}$, we construct the following insertion rules

$$\text{form-}(r_1) \quad (X, \#YZ, \alpha_1\alpha_2) \text{ in } P_\gamma, \text{ where } \alpha_1\alpha_2 \in (N \cup T \cup \{c\})^2.$$

- Group 2: For each rule $r : XY \to XZ \in P$ of Type 2, with $X, Y, Z \in N$, we construct the following insertion rules

$$\text{form-}(r_2) \quad (XY, \#Z, \alpha_1\alpha_2) \text{ in } P_\gamma, \text{ where } \alpha_1\alpha_2 \in (N \cup T \cup \{c\})^2.$$

- Group 3 (Relocation task for X): For each $X, Y \in N$, we construct the following insertion rules

$$\begin{array}{ll} \text{form-}(r_3) & (XY\#, \#X, \alpha), \text{ where } \alpha \in (N \cup T \cup \{c\}), \\ \text{form-}(r_4) & (X, \#, Y\#\#), \\ \text{form-}(r_5) & (\#Y\#, Y, \#X). \end{array}$$

We define a projection $h : \Sigma^* \to T^*$ by

$$\begin{array}{ll} h(a) = a & \text{for all } a \in T, \\ h(a) = \lambda & \text{otherwise.} \end{array}$$

Finally, let $R = A\Sigma^* \cap \Sigma^* B - \Sigma^+ C' \Sigma^+$ with $C' = \Sigma^2 - C$,

$$\begin{array}{l} A = \{X\# \mid X \in N\}, \\ B = \{cc\}, \\ C = \{X\# \mid X \in N\} \cup \{\#X \mid X \in N\} \cup \{aX \mid X \in N\} \cup \\ \quad \{ab \mid a, b \in T\} \cup \{ac \mid a \in T\} \cup \{\#a \mid a \in T\} \cup \{\#c\}. \end{array}$$

Then R is a strictly 2-testable language prescribed by $S_2 = (A, B, C)$. The language R can be represented by $R = N\{\#\}(T \cup N\{\#\})^*\{cc\}$.

Then we obtain $L(G) = h(L(\gamma) \cap R)$, which will be proven in the sequel. We start by introducing some useful notions.

We call the symbol # a *marker*. A symbol in N followed by # is said to be *#-marked* (briefly *marked*). A symbol in $N \cup T$ which is not marked is said to be *unmarked*. We call a string in $N\{\#\}$ a *wreck* and a string in $(N\{\#\})^+$ a *wrecks*. Since the symbols c and # are special symbols, they are neither marked nor unmarked. A string xcc, where x is in $(N\{\#\} \cup N \cup T)^*$, is a *legal* string.

An intuitive explanation of marked symbols and unmarked symbols is the followings:

Note 1. A marked symbol means that the symbol has been used (i.e. consumed) for some derivation in γ.

Note 2. In γ at each step a string consisting of unmarked symbols of a legal string indicates a sentential form of G.

By the construction of R, making $L(\gamma) \cap R$ leads to only legal strings. Then if we erase the "wrecks" and the symbol c, we get the legal strings of unmarked symbols which are exactly sentential forms of G.

By using the rules of Group 1 and Group 2, we can simulate the rules of Type 1 and Type 2 respectively. By using the rules of Group 3, we move an unmarked symbol to the right across a block $M\#$, where $M \in N$. Thus the nonterminal pairs XY can be ready for simulating the rules $XY \to YZ$ of Type 2.

In order to prove the equality $L(G) = h(L(\gamma) \cap R)$, we first prove the inclusion $L(G) \subseteq h(L(\gamma) \cap R)$.

Fact 1. *Applying a form-(r_1) rule $: (X, \#YZ, \alpha_1\alpha_2)$ to an occurrence of a string $X\alpha_1\alpha_2$ with $\alpha_1\alpha_2 \in (N \cup T \cup \{c\})^2$ makes a new occurrence of the string $X\#YZ\alpha_1\alpha_2$. Note that the unmarked symbol X becomes marked, while the symbols Y, Z are newly created unmarked symbols.*

Fact 2. *Applying a form-(r_2) rule $: (XY, \#Z, \alpha_1\alpha_2)$ to an occurrence of a string $XY\alpha_1\alpha_2$ with $\alpha_1\alpha_2 \in (N \cup T \cup \{c\})^2$ makes a new occurrence of the string $XY\#Z\alpha_1\alpha_2$. Note that the symbol X is preserved in just the unmarked state, the unmarked symbol Y becomes marked, while the symbol Z is newly created unmarked symbol.*

Lemma 8. *The rules in Group 3 can replace a substring* $XY\#\alpha$ ($\alpha \in N \cup T \cup \{c\}$) *by a substring consisting of the strings in* $N\{\#\}$ *and ending with* $X\alpha$. *The symbol* X *is unmarked before and after the derivations.*

Proof. A form-(r_3) rule $(XY\#, \#X, \alpha)$ can be applied to a string $XY\#\alpha$, where $X, Y \in N$, $\alpha \in N \cup T \cup \{c\}$. After applying the form-(r_3) rule, we have $XY\#\#X\alpha$. Then the form-(r_4) rule $(X, \#, Y\#\#)$ can be applied for the substring $XY\#\#$, and we have $X\#Y\#\#X\alpha$. Now we apply the form-(r_5) rule $(\#Y\#, Y, \#X)$ for the substring $\#Y\#\#X$, and the substring is replaced by $\#Y\#Y\#X$.

Therefore, the substring $XY\#\alpha$ is replaced by $X\#Y\#Y\#X\alpha$, which has the unmarked symbol X on the rightmost position. □

Thus the insertion rules in γ simulate the rules in G, and generate legal strings from the legal string *Scc*.

We will give separate consideration to the case of using the rules in Group 3.

Lemma 9. *Once the form-*(r_3) *rule* : $(XY\#, \#X, \alpha)$ *is applied to obtain a substring of a legal string, then the form-*(r_4) *rule and form-*(r_5) *rule are used in this order.*

Proof. We may consider a substring $XY\#\alpha$, where $X, Y \in N$, $\alpha \in N \cup T \cup \{c\}$. After using rule in form-(r_3), we obtain $XY\#\#X\alpha$. Because of the symbols $\#\#$, rules in form-(r_1) or (r_2) or (r_3) cannot be applied for the substring $XY\#\#$. In view of the construction of form-(r_5) rule, we cannot apply a form-(r_5) rule for $XY\#\#$. Hence, the only applicable rule for $XY\#\#$ is form-(r_4) rule.

After using form-(r_4) rule $(X, \#, Y\#\#)$ for $XY\#\#X\alpha$, we obtain the substring $X\#Y\#\#X\alpha$. For the symbol X following $\#\#$, we have a chance to apply one of the rules in form-(r_1), (r_2), (r_3), (r_4). If we apply form-(r_1) or form-(r_2) rule, we may take it as the first step of simulation for Type 1 or Type 2 respectively. Note that, during these simulations, X remains at the immediately to the right of $\#\#$. If we apply form-(r_3) or form-(r_4) rule, we may take it independently a new relocation task. Note that, after application of form-(r_3) or form-(r_4) rule, X remains immediately to the right of $\#\#$. Therefore, in all cases the symbol $\#\#$ is followed by X. Further, since the symbol X was originally unmarked in $XY\#\alpha$, X provides the possibility of applying one of the rules in form-(r_1), (r_2), (r_3), (r_4). Hence this application causes no trouble with the current relocation task.

After using form-(r_4) rule for $XY\#\#$, we obtain $X\#Y\#\#$. From the above notation, since X always follows the symbols $\#\#$, after applying form-(r_4) rule, we obtain $X\#Y\#\#X$. In the substring $X\#Y\#\#$, both of the symbols X and Y are already marked, and in view of the form of the rules, none of form-(r_1), (r_2), (r_3), (r_4) rule can be used for this substring. Hence, the only applicable rule for $X\#Y\#\#X$ is form-(r_5) rule. After applying this rule, $(\#Y\#, Y, \#X)$, we have $X\#Y\#X\#X$, which is the substring of a legal string.

Hence to obtain a substring of a legal string, whenever we use the form-(r_3) rule, we have to use form-(r_4) rule and (r_5) rule in this order. □

From Lemma 9, for any derivation in γ, $x \overset{\pi}{\Longrightarrow}_{\gamma} y$, there is a *standard derivation* which satisfies that form-(r_4) rule and form-(r_5) rule are applied in this order immediately after applying form-(R_3) rule.

Denote by $umk(x)$ a string consisting of unmarked symbols in a legal string x generated by γ. Note that since c is the special symbol, neither marked nor unmarked, $umk(x)$ does not contain a suffix cc. We thus have the next lemma.

Lemma 10. *The nonterminal symbol S derives x in G if and only if there is a derivation $Scc \Longrightarrow_{\gamma}^{*} x'$ in γ such that $umk(x') = x$.*

Proof. We will show that if there is a derivation $S \Longrightarrow_{G}^{n} x$ then there is a derivation $Scc \Longrightarrow_{\gamma}^{*} x'$ such that $umk(x') = x$ by induction on n.

Base step: If $n = 0$, then for the axiom Scc in γ, $umk(Scc) = S$ holds. Thus obviously the claim holds.

Induction step: We suppose that the claim holds for any $k \leq n$. Now consider a derivation $S \Longrightarrow_{G}^{n} x \Longrightarrow_{G} y$. From the induction hypothesis, there is a derivation $Scc \Longrightarrow_{\gamma}^{*} x'$, where $umk(x') = x$. If the rule applied for x is of Type 1 (Type 2, resp.) then we use the corresponding insertion rule in Group 1 (Group 2, resp.) for the string x'.

However, in the latter case (i.e. Group 2), if the insertion rule in Group 2 cannot be immediately applied for x', we need to apply some rules in Group 3. From Lemma 8, after application of the rules in Group 3, unmarked symbols of a legal string x' remain unchanged. We denote this process of derivations by $x' \Longrightarrow_{\gamma}^{*} x'' \Longrightarrow_{\gamma} y'$, where x'', which is a string ready for applying a rule in Group 2, is derived by using only rules in form-(r_3), (r_4), (r_5) in Group 3 and y' is derived by using only a rule in Group 2. Note that $umk(x') = umk(x'')$.

Then, in either case, from Fact 1 and Fact 2 we eventually have $umk(y') = y$. Therefore the claim holds for $n + 1$.

Conversely, we can prove that if there is a standard derivation $Scc \overset{\pi}{\Longrightarrow}_{\gamma} x'$ then there is a derivation $S \Longrightarrow_{G}^{*} x$ such that $umk(x') = x$ by induction on the number n of legal strings in the derivation π. We omit the proof here. □

In view of the manner of constructing the strictly 2-testable language R and the projection h, we have the following fact.

Fact 3. *For any $y \in L(\gamma)$, if y is in R and $umk(y) \in T^{*}$, then $umk(y) = h(y)$.*

From Lemma 10 and Fact 3, we obtain the inclusion $L(G) \subseteq h(L(\gamma) \cap R)$. Next we prove the inverse inclusion which completes the proof of Theorem 6.

Fact 4. *As far as unmarked symbols are concerned, the rules in Group 1 and Group 2 can only simulate the rules of Type 1 and Type 2 respectively in G.*

Proof of Theorem 6. From Fact 4, Lemma 9 and Lemma 10, every string of a form $umk(xcc)$ is generated by the grammar G, where xcc is a legal string generated by γ.

Therefore, if for any $y \in L(\gamma)$, y is in R, then there is a string $h(y)$ such that $S \Longrightarrow_{G}^{*} h(y)$. This means that the inclusion $h(L(\gamma) \cap R) \subseteq L(G)$ holds. Together with the fact that $L(G) \subseteq h(L(\gamma) \cap R)$, we complete the proof of Theorem 1. □

6 Conclusion

In this paper, we have contributed to the study of insertion systems with new characterizations of recursively enumerable, context-free, and regular languages. Specifically, we have shown that

$REG = H(INS_1^0 \cap LOC(k))$ with $k \geq 2$.
$H(INS_1^0 \cap LOC(1)) \subset REG \subset H(INS_i^0 \cap LOC(k))$ with $i, k \geq 2$.
$CF = H(INS_3^0 \cap LOC(k))$ with $k \geq 4$.
$RE = H(INS_3^3 \cap LOC(k))$ with $k \geq 2$.

The followings are open problems:

Can CF be represented as $CF = H(INS_3^0 \cap LOC(k))$ for some $k < 4$?
Can RE be represented as $RE = H(INS_i^j \cap LOC(2))$ for some $i < 3$ or $j < 3$?

Acknowledgements

The author is deeply indebted to T.Yokomori for his helpful discussions. This work is supported in part by Grant-in-Aid for the Research Institute for Science and Technology of Tokyo Denki University with no.Q07J-05.

References

1. Margenstern, M., Păun, G., Rogozhin, Y., Verlan, S.: Context-free insertion-deletion systems. Theor. Comput. Sci. 330(2), 339–348 (2005)
2. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing. New Computing Paradigms. Springer, Heidelberg (1998)
3. Yokomori, T., Kobayashi, S.: Learning local languages and their application to DNA sequence analysis. IEEE Trans. Pattern Anal. Mach. Intell. 20(10), 1067–1079 (1998)
4. Head, T.: Splicing representations of strictly locally testable languages. Discrete Applied Math. 87, 87–139 (1998)
5. Martin-Vide, C., Păun, G., Salomaa, A.: Characterizations of recursively enumerable languages by means of insertion grammars. Theor. Comput. Sci. 205(1-2), 195–205 (1998)
6. Onodera, K.: A note on homomorphic representation of recursively enumerable languages with insertion grammars. IPSJ Journal 44(5), 1424–1427 (2003)
7. Păun, G., Pérez-Jiménez, M.J., Yokomori, T.: Representations and characterizations of languages in Chomsky hierarchy by means of insertion-deletion systems. Int. J. Found. Comput. Sci. 19(4), 859–871 (2008)
8. Rozenberg, G., Salomaa, A. (eds.): Handbook of formal languages. Springer, New York (1997)
9. McNaughton, R., Papert, S.A.: Counter-Free Automata. M.I.T. research monograph, vol. 65. MIT Press, Cambridge (1971)
10. Verlan, S.: On minimal context-free insertion-deletion systems. J. Autom. Lang. Comb. 12(1), 317–328 (2007)

On Parallel Communicating Grammar Systems and Correctness Preserving Restarting Automata

Dana Pardubská[1,⋆], Martin Plátek[2,⋆⋆], and Friedrich Otto[3]

[1] Dept. of Computer Science, Comenius University, Bratislava
pardubska@dcs.fmph.uniba.sk
[2] Dept. of Computer Science, Charles University, Prague
Martin.Platek@mff.cuni.cz
[3] Fachbereich Elektrotechnik/Informatik, Universität Kassel, Kassel
otto@theory.informatik.uni-kassel.de

Abstract. This paper contributes to the study of Freely Rewriting Restarting Automata (FRR-automata) and Parallel Communicating Grammar Systems (PCGS) as formalizations of the linguistic method of *analysis by reduction*. For PCGS we study two complexity measures called *generation complexity* and *distribution complexity*, and we prove that a PCGS Π, for which both these complexity measures are bounded by constants, can be simulated by a freely rewriting restarting automaton of a very restricted form. From this characterization it follows that the language $L(\Pi)$ is semi-linear, that its characteristic analysis is of polynomial size, and that this analysis can be computed in polynomial time.

1 Introduction

This paper contributes to the analysis of Freely Rewriting Restarting Automata (FRR-automata) and Parallel Communicating Grammar Systems (PCGS), see [1,2]. Here the main goal is the quest for constraints for FRRs and PCGSs, under which the corresponding classes of languages and their *analysis by reduction* are of interest from the point of view of computational linguistics. For example, this is the case if the languages obtained are semi-linear, and if their so-called *characteristic analysis* can be computed in polynomial time.

Freely rewriting restarting automata create a suitable tool for modelling the so-called *analysis by reduction*. In general, analysis by reduction explains basic types of so-called dependencies in sentences of natural languages. The Functional Generative Description for the Czech language developed in Prague (see, e.g., [3]) is based on this method.

In order to model analysis by reduction, FRR-automata work on so-called *characteristic languages*, that is, on languages with auxiliary symbols (categories) included in addition to the input symbols. The *proper language* is obtained from

⋆ Partially supported by the Slovak Grant Agency for Science (VEGA) under contract "1/0726/09 - Algorithmic and complexity issues in information processing".
⋆⋆ Partially supported by the Grant Agency of the Czech Republic under Grant No. 405/08/0681 and by the program Information Society under project 1ET100300517.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 660–671, 2009.

a characteristic language by removing all auxiliary symbols from its sentences. We focus on restarting automata that ensure the *correctness preserving property* for the analysis, that is, after any restart within a computation starting with a word from the characteristic language, the content of the tape is again from that language. This property is required in order to introduce the notion of *characteristic analysis* in a linguistically adequate way. To achieve our goal we use a technique that is based on the notion of *skeletal set*, which is particularly useful for error recovery during a robust parsing or during a grammar-checking procedure.

We study two complexity measures for returning PCGSs with regular components: the *generation complexity*[1], which bounds the number of generative sections in a word generated by a PCGS, and the *distribution complexity*[2], which bounds the distribution of concurrently generated segments over the word generated. Our technical main result states the following. If Π is a PCGS, for which the generation complexity is bounded by a constant g and the distribution complexity is bounded by a constant d, then the language $L(\Pi)$ generated by Π is the proper language of a freely rewriting restarting automaton M of a very restricted form: M is correctness preserving, and it only performs rewrite operations of a very restricted type. In addition, the number of rewrites per cycle and the number of auxiliary symbols that occur in any word from the characteristic language of M are both bounded by constants that depend on the bounds g and d above. In fact, M even has a skeletal set of type (g, d). Based on these restrictions of M we obtain the following important results on the language $L(\Pi)$: it is semi-linear, its characteristic analysis is of polynomial size, and it can be computed in polynomial time, where the degree of the polynomial time-bound also depends on the constants g and d. The latter two results answer questions that were left open in [1,2].

The structure of the paper is as follows. In Section 2 we give the (informal) definitions of FRR-automata, AuxRR-automata, and PCGS and present some basic facts about them. In Section 3, which constitutes the technical main part of the paper, we present our simulation result described above, and we then introduce the notion of skeletal set. Using this notion we derive the main results of the paper from the simulation given in the first part of this section. This section ends with some concluding remarks.

2 Basic Notions

Restarting automata. A *freely rewriting restarting automaton* (FRR-automaton, for short) is a restarting automaton without rewriting constraints, that is, it is a nondeterministic machine with a flexible tape, a read/write window of a fixed size $k \geq 1$ that can move along this tape, and a finite-state control. Formally, it is described by an 8-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$. Here Q denotes a finite set of (internal) states that contains the initial state q_0, Σ is a finite input alphabet,

[1] The generation complexity corresponds to the degree of (linguistic) independence.
[2] The distribution complexity models the degree of (linguistic) dependence (valence).

and Γ is a finite tape alphabet that contains Σ. The elements of $\Gamma \setminus \Sigma$ are called *auxiliary symbols*. The additional symbols $\mathbb{c}, \$ \notin \Gamma$ are used as markers for the left and right end of the workspace, respectively. They cannot be removed from the tape. The behavior of M is described by a transition function δ that associates a finite set of transition steps to each pair of the form (q, u), where q is a state and u is a possible content of the read/write window. There are four types of transition steps: *move-right steps*, *rewrite steps*, *restart steps*, and *accept steps*. A *move-right step* simply shifts the read/write window one position to the right and changes the internal state. A *rewrite step* causes M to replace a non-empty prefix u of the content of the read/write window by a word v satisfying $|v| \leq |u|$, and to change the state. Further, the read/write window is placed immediately to the right of the string v. A *restart step* causes M to place its read/write window over the left end of the tape, so that the first symbol it sees is the left sentinel $\mathbb{c}$, and to reenter the initial state q_0. Finally, an *accept step* simply causes M to halt and accept.

A *configuration* of M is described by a string $\alpha q \beta$, where $q \in Q$, and either $\alpha = \varepsilon$ (the empty word) and $\beta \in \{\mathbb{c}\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{\mathbb{c}\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; here q represents the current state, $\alpha\beta$ is the current content of the tape, and it is understood that the window contains the first k symbols of β or all of β when $|\beta| \leq k$. A *restarting configuration* is of the form $q_0 \mathbb{c} w \$$, where $w \in \Gamma^*$.

Any computation of M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration. The window is shifted along the tape by move-right and rewrite operations until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, the computation necessarily finishes in a halting configuration – such a phase is called a *tail*. It is required that *in each cycle, M performs at least one rewrite step that is strictly length-decreasing*. Thus, each cycle strictly reduces the length of the tape. We use the notation $u \vdash_M^c v$ to denote a cycle of M that begins with the restarting configuration $q_0 \mathbb{c} u \$$ and ends with the restarting configuration $q_0 \mathbb{c} v \$$; the relation $\vdash_M^{c^*}$ is the reflexive and transitive closure of $\vdash_M^c$.

A word $w \in \Gamma^*$ is *accepted* by M, if there is a computation which starts from the restarting configuration $q_0 \mathbb{c} w \$$, and ends with an application of an accept step. By $L_{\mathrm{C}}(M)$ we denote the so-called *characteristic language of M*, which is the language consisting of all words accepted by M. By Pr^{Σ} we denote the projection from Γ^* onto Σ^*, that is, Pr^{Σ} is the morphism defined by $a \mapsto a$ $(a \in \Sigma)$ and $A \mapsto \varepsilon$ $(A \in \Gamma \setminus \Sigma)$. If $v := \mathsf{Pr}^{\Sigma}(w)$, then v is the *Σ-projection* of w, and w is an *expanded version* of v. For a language $L \subseteq \Gamma^*$, $\mathsf{Pr}^{\Sigma}(L) := \{\, \mathsf{Pr}^{\Sigma}(w) \mid w \in L \,\}$. Further, for $K \subseteq \Gamma$, $|x|_K$ denotes the number of occurrences of symbols from K in x.

Motivated by linguistic considerations to model the analysis by reduction with parallel processing, we are interested in the so-called *proper language of M*, which is the set of words $L_{\mathrm{P}}(M) := \mathsf{Pr}^{\Sigma}(L_{\mathrm{C}}(M))$. Hence, a word $v \in \Sigma^*$ belongs to $L_{\mathrm{P}}(M)$ if and only if there exists an expanded version u of v such that $u \in L_{\mathrm{C}}(M)$. Realize that the main difference between the input and the

proper language lies in the way in which auxiliary symbols are inserted into the (terminal) words of the language.

An FRR-automaton M is called *linearized* if there exists a constant j such that $|w|_{\Gamma\setminus\Sigma} \leq j \cdot |w|_\Sigma + j$ for each $w \in L_C(M)$ [1,2]. Since a linearized FRR-automaton only uses linear space, we see immediately that the proper language of each linearized FRR-automaton is context-sensitive.

In a real process of analysis by reduction of a sentence of a natural language it is desired that whatever is done within the process does not change the correctness of the sentence. For restarting automata this property can be formalized as follows: An FRR-automaton M is *correctness preserving* if $u \in L_C(M)$ and $u \vdash_M^{c^*} v$ imply that $v \in L_C(M)$, too. While it is easily seen that each *deterministic* FRR-automaton is correctness preserving, there are FRR-automata which are not correctness preserving.

Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be an FRR-automaton that is correctness preserving, and let $w \in \Sigma^*$. Then $A_C(w, M) = \{\, w_C \in L_C(M) \mid \ w_C$ is an extended version of $w \,\}$ is called the *characteristic analysis of w by M*.

Definition 1. *An* FRR*-automaton $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ is called* aux-rewriting *if, for each of its rewrite operations $(q', v) \in \delta(q, u)$, $\mathsf{Pr}^\Sigma(v)$ is obtained from $\mathsf{Pr}^\Sigma(u)$ by deleting some symbols, and $\mathsf{Pr}^{\Gamma\setminus\Sigma}(v)$ is obtained from $\mathsf{Pr}^{\Gamma\setminus\Sigma}(u)$ by replacing some symbol by another symbol.*

By AuxRR we denote the class of aux-rewriting FRR-automata that are correctness preserving. For each type X of restarting automata and each $t \in \mathbb{N}_+$, we use t-X to denote the class of X-automata that execute at most t rewrite steps in any cycle.

Parallel Communicating Grammar Systems. A returning PCGS *of degree m* (≥ 1) with regular components is an $(m+1)$-tuple $\Pi = (G_1, \ldots, G_m, K)$, where, for all $i \in \{1, \ldots, m\}$, $G_i = (N_i, T, S_i, P_i)$ are regular grammars, called *component grammars*, satisfying $N_i \cap T = \emptyset$, and $K \subseteq \{Q_1, \ldots, Q_m\} \bigcap \bigcup_{i=1}^m N_i$ is a set of special symbols, called *communication symbols*. A *configuration* of Π is an m-tuple $C = (x_1, \ldots, x_m)$, where $x_i = \alpha_i A_i$, $\alpha_i \in T^*$, and $A_i \in (N_i \cup \{\varepsilon\})$; we call x_i the *i-th component* of the configuration. The *nonterminal cut* of configuration C is the m-tuple $N(C) = (A_1, A_2, \ldots, A_m)$. If $N(C)$ contains at least one communication symbol, it is called an *NC-cut* and denoted by $NC(C)$.

A *derivation* of Π is a sequence of configurations $D = C_1, C_2, \ldots, C_t$, where C_{i+1} is obtained from C_i by one generative step or one communication *step*. If no communication symbol appears in any of the components, then we perform a *generative step*. It consists of synchronously performing a rewrite step in each of the component grammars G_i, $1 \leq i \leq m$. If any of the components is a terminal string, it is left unchanged, and if any of the components contains a nonterminal that cannot be rewritten, the derivation is blocked. If the first component is a terminal word w, then w is the word that is generated by Π in this derivation. In this situation D is usually denoted as $D(w)$. If a communication symbol is present in any of the components, then a *communication step* is performed. It

consists of replacing those communication symbols with the phrases they refer to for which the phrases themselves do not contain communication symbols. Such an individual replacement is called a *communication*. Obviously, in one communication step at most $m-1$ communications can be performed. Communication steps are performed until no more communication symbols are present, or until the derivation is blocked because no communication symbol can be replaced. The maximal sub-sequence of communication steps forms a *communication section*.

A *generative section* is a non-empty sequence of generative steps between two consecutive communication sequences in D (or before the first or after the last communication step). Thus, the communication steps divide the derivation into generative and communication sections.

The *(terminal) language* $L(\Pi)$ generated by Π is the set of terminal words that appear in the component G_1, which is called the *master* of the system:

$$L(\Pi) = \{\, \alpha \in T^* \mid (S_1, \ldots, S_m) \Rightarrow^+ (\alpha, \beta_2, \ldots, \beta_m) \,\}.$$

Several notions are associated with the derivation $D(w)$:

- $g(i,j)$ (or $g(i,j,D(w))$) denotes the (i,j)-*(generative) factor* of $D(w)$, which is the terminal word that is generated by G_i within the j-th generative section of $D(w)$;
- $n(i,j)$ (or $n(i,j,D(w))$) denotes the number of occurrences of $g(i,j)$ in w.
- The *communication structure* $CS(D(w))$ of $D(w)$ captures the connection between the terminal word w and its particular derivation $D(w)$):
  $CS(D(w)) = (i_1,j_1),(i_2,j_2),\ldots,(i_r,j_r)$, if $w = g(i_1,j_1)g(i_2,j_2)\ldots g(i_r,j_r)$.
- For $j \geq 1$ let $N(j,D(w)) = \sum_{i=1}^{m} n(i,j,D(w))$. Then the so-called *degree of distribution* $DD(D(w))$ of $D(w)$ is the maximum over all $N(j,D(w))$.
- The *trace* of a (sub-)derivation D is the sequence $T(D)$ of nonterminal cuts of individual configurations of D: $T(D)$=$N(C_0), N(C_1), \ldots, N(C_t)$. Note that (in general) the trace does not unambiguously identify the derivation.
- The *communication sequence*, resp. the *NC-sequence*, is defined analogously: $NCS(D)$ is the sequence of all NC-cuts in the (sub-)derivation D. Realize that the communication sequence $NCS(D(w))$ unambiguously defines the communication structure of $D(w)$. Moreover, the set of words with the same communication sequence/structure might, in general, be infinite.

A *cycle in a derivation* D is a smallest (continuous) sub-derivation $\mathcal{C} = C_1, \ldots, C_j$ of D such that $N(C_1) = N(C_j)$. If *none* of the nonterminal cuts in $\mathcal{C}$ contains a communication symbol, then the whole cycle is contained in a generative section; we speak about a *generative cycle* in this case. If the first nonterminal cut contains communication symbols, which means that $N(C_1) = N(C_j)$ are NC-cuts, then the cycle is called a *communication cycle*.

If there is a cycle in the derivation $D(w)$, then manifold repetition[3] of the cycle is possible and the resulting derivation is again a derivation of some terminal word. Observe, however, that the repetition or deletion of a generative cycle does

[3] Deletion of a cycle is also possible.

not change the communication structure of a derivation. We call a derivation $D(w)$ *reduced*, if every repetition of any of its cycles leads to a *longer* terminal word ω; $|w| < |\omega|$. Obviously, to every derivation $D(w)$ there is an equivalent reduced derivation $D'(w)$ of the same word. In what follows, we consider only derivations that are reduced.

Finally, we define several complexity measures for PCGS. Informally, the *communication complexity* of a derivation D (denoted $com(D)$) is defined as the number of communications performed within the derivation D; analogously, the *distribution complexity* of a derivation D is the degree of distribution defined above, and the *generation complexity* of the derivation D is the number of generative sections in D. Then the communication/distribution/generation complexity of a language and the associated complexity class are defined in the usual way (always considering the corresponding maximum).

Here, we are mainly interested in those classes of languages for which all the complexity measures considered above are bounded from above by a constant. For natural numbers k, d, g, we denote the corresponding communication complexity class by $\mathsf{COM}(k)$, and the distribution and/or generation complexity class by d-DG, g-DD, and d-g-DDG, respectively. Some relevant observations characterizing the derivations of a PCGS with constant communication complexity are summarized in the following facts.

Fact 1. *Let Π be a* PCGS *with constant communication complexity. Then there are constants $d(\Pi)$, $\ell(\Pi)$, $s(\Pi)$, and $e(\Pi)$ such that*

1. *the number $n(i,j)$ of occurrences of individual $g(i,j)$'s in a* reduced *derivation is bounded by $d(\Pi)$; that is, $n(i,j) \leq d(\Pi)$;*
2. *the length of the communication structure for every (reduced) derivation is bounded by $\ell(\Pi)$;*
3. *the cardinality of the set of all possible communication structures corresponding to a reduced derivation by Π is bounded by $s(\Pi)$.*
4. *Let $D(w)$ be a* reduced *derivation of a terminal word w in Π. If more than $e(\Pi)$ generative steps are performed in the j-th generative section of $D(w)$, then at least one factor $g(i,j,D(w))$ has been changed.*

Based on pumping arguments the following observation now follows easily.

Proposition 1. *Let Π be a* PCGS *with constant communication complexity. Then the set of all derivations by Π without a generative cycle is finite.*

3 Analysis by Reduction for PCGSs

In [1] it is shown how to transform a PCGS with constant communication complexity into a deterministic linearized FRR-automaton. In what follows we will reduce the number of occurrences of auxiliary symbols in the words from the characteristic language of the corresponding restarting automaton to a constant by utilizing nondeterminism. In fact, the resulting automaton will be an AuxRR-automaton for which the positions at which rewrites are performed within a

cycle are severely restricted. These restrictions will be formalized through the notion of a *skeletal set* in Definition 2.

Theorem 1. *For each $L \in g$-d-DDG, there is a d-AuxRR-automaton M such that $L = L_{\mathrm{P}}(M)$. Moreover, the number of auxiliary symbols in $w \in L_{\mathrm{C}}(M)$ is bounded from above by the constant $2 \cdot g \cdot d + 2$.*

Proof. Let $L \in g$-d-DDG, and let Π be a PCGS with m components that generates L with distribution complexity d and generation complexity g. Our construction is based on the fact that Π has only a finite number σ of cycle-free derivations. Let $Cf = \{\hat{D}_1(\hat{w}_1), \cdots, \hat{D}_\sigma(\hat{w}_\sigma)\}$ be the set of these derivations.

We describe a d-AuxRR-automaton M that, given a certain extended version w_C of a word w, performs the analysis by reduction which starts by considering a Π-derivation $D(w)$ of the word $w \in L$, and ends by checking that the Π-derivation $\hat{D}_k(\hat{w}_k)$ obtained is one of the cycle-free derivations listed above.

Let $w \in L$, let $D(w)$ be a derivation of w in Π, and let $g(i,j)$ be the terminal word generated by the component grammar G_i within the j-th generative section. Then w can be written as $w = g(i_1,j_1)g(i_2,j_2)\dots g(i_r,j_r)$. As Π has generation complexity g, there are at most g generative sections in the derivation $D(w)$, and as Π has distribution complexity d, there are at most d occurrences of factors $g(i_t,j_t)$ such that $j_t = j$ for any j. Hence, we have $r \leq d \cdot g$.

To reconstruct the derivation of a factor $g(i,j)$ in detail we utilize the following notion of an *extended j-trace*. Let

$$(A_1,\dots,A_m),(\alpha_{1,1}A_{1,1},\dots,\alpha_{1,m}A_{1,m}),\dots \\ (\alpha_{1,1}\alpha_{2,1}\dots\alpha_{s,1}A_{s,1},\dots,\alpha_{1,m}\alpha_{2,m}\dots\alpha_{s,m}A_{s,m})$$

be the sub-derivation corresponding to the j-th generative section of $D(w)$. It yields the following *extended version* of the trace of the j-th generative section:

$$\begin{pmatrix} A_1 \\ A_2 \\ \cdots \\ A_m \end{pmatrix} \begin{pmatrix} \alpha_{1,1}A_{1,1} \\ \alpha_{1,2}A_{1,2} \\ \cdots \\ \alpha_{1,m}A_{1,m} \end{pmatrix} \begin{pmatrix} \alpha_{2,1}A_{2,1} \\ \alpha_{2,2}A_{2,2} \\ \cdots \\ \alpha_{2,m}A_{2,m} \end{pmatrix} \cdots\cdots \begin{pmatrix} \alpha_{s,1}A_{s,1} \\ \alpha_{s,2}A_{s,2} \\ \cdots \\ \alpha_{s,m}A_{s,m} \end{pmatrix}.$$

This description is denoted $ex\text{-}T(D(w),j)$. It describes the sequence of generative steps of the j-th generative section. Assume that $D(w)$ has g_k generative sections. Then $ex\text{-}T(D(w)) = ex\text{-}T(D(w),1), ex\text{-}T(D(w),2),\dots,ex\text{-}T(D(w),g_k)$ is called the *extended trace* of $D(w)$. Let us note that $ex\text{-}T(D(w))$ can serve as an another representation of $D(w)$.

The restarting automaton M processes the word w_C as follows. In each cycle M first *nondeterministically chooses* an index j of a generative section, and then it consistently removes the *rightmost* generative cycle from each of the factors $g(i,j)$ of w. Simultaneously, it checks the consistency of its guess and makes necessary changes in the delimiters. M repeatedly executes such cycles until a word is obtained that does not contain any generative cycles anymore. From Proposition 1 we see that the set of words of this form is finite.

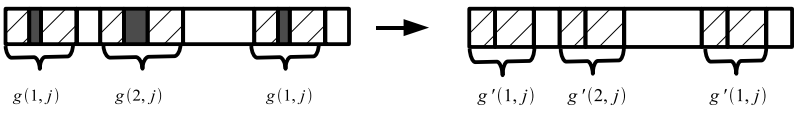


Fig. 1. The situation before and after the execution of a cycle that makes rewrites within the j-th generative section: the reduced parts are grey. Two occurrences of $g(1,j)$ were reduced to $g'(1,j)$; one occurrence of $g(2,j)$ was reduced to $g'(2,j)$.

We show that we only need to store a constant amount of information in the auxiliary symbols to realize this strategy. Accordingly, the word w_C is chosen as

$$w_C := \Delta_{0,k}\Delta_{1,k}\ g(i_1,j_1)\Lambda_{1,k}\Delta_{2,k}\ g(i_2,j_2)\Lambda_{2,k}\ldots\Delta_{r,k}\ g(i_r,j_r)\Lambda_{r,k}\Delta_{r+1,k},$$

where $\Delta_{0,k},\ldots,\Delta_{r+1,k}$ and $\Lambda_{1,k},\ldots,\Lambda_{r,k}$ are auxiliary symbols. These symbols are not only used to separate the individual factors $g(i,j)$ from each other, but also to store relevant information about the derivations $D(w)$ and $\hat{D}_k(\hat{w}_k)$. In fact, the information stored in each symbol $\Delta_{t,k}$ $(0 \le t \le r+1)$ will be *fixed*, while the information stored in each symbol $\Lambda_{t,k}$ $(1 \le t \le r)$ is *temporary*. The information stored in $\Delta_{t,k}$ describes the factor $g(i_t,j_t)$ and the factor $\hat{g}(i_t,j_t)$. The information stored in $\Lambda_{t,k}$ will be changed whenever a deletion is executed in the left neighborhood of the particular delimiter; it describes a suffix of the relevant extended trace $ex\text{-}T(D(w),j_t)$. By $\Lambda_{t,k}(D(w))$ we will denote the particular symbol that corresponds to this information.

▶ If M has decided to try to execute a *cycle*, then it nondeterministically chooses a number $j \in \{1,\ldots,g_k\}$ as the index of the generative section of $D(w)$ that it will reduce in this cycle. It stores j and $\Delta_{0,k}$ in its internal state and moves its head to the right until it reaches the first delimiter $\Delta_{t,k}$ for which $j_t = j$ holds, that is, the leftmost occurrence of a factor of the form $g(i,j)$ is found. Then M moves its window further to the right until $\Delta_{t,k}$ becomes the leftmost symbol inside the window. Now M is going to try to simulate a reduction of the factor $g(i_t,j_t)$ as described above.

(1.) From the description of $ex\text{-}T(\hat{D}_k(\hat{w}_k))$ stored in $\Delta_{0,k}$, M determines the nonterminal cut with which the extended j-trace $ex\text{-}T(D(w),j)$ begins.

(2.) Moving from left to right across the factor $g(i_t,j)$, M guesses the extended j-trace $ex\text{-}T(D(w),j)$ in such a way that it is consistent with the word $g(i_t,j)$; if no such guess is possible, the computation is blocked. Simultaneously, M always remembers the current suffix ℓ_t of length $2\cdot p(\Pi)$ of the part of $ex\text{-}T(D(w),j)$ considered so far. Here $p(\Pi)$ is a constant that is sufficiently large to ensure that any sub-word of any $ex\text{-}T(D(w),j)$ of length at least $p(\Pi)$ contains a generative cycle.

(3.) When the delimiter $\Delta_{t+1,k}$ occurs as a rightmost symbol in M's window, then M tries to execute a reduction of the suffix of $g(i_t,j)$; if none is possible, then the computation is blocked. To perform a reduction M checks whether the current suffix ℓ_t of $ex\text{-}T(D(w,j))$ contains a (generative) cycle, that is, whether the suffix $\ell_{t,2}$ of ℓ_t of length $p(\Pi)$ has the following form:

$$\begin{pmatrix} \alpha_{1,1}A_{1,1} \\ \alpha_{1,2}A_{1,2} \\ \cdots \\ \alpha_{1,m}A_{1,m} \end{pmatrix} \cdots \begin{pmatrix} \alpha_{\gamma,1}A_{\gamma,1} \\ \alpha_{\gamma,2}A_{\gamma,2} \\ \cdots \\ \alpha_{\gamma,m}A_{\gamma,m} \end{pmatrix} \cdots \begin{pmatrix} \alpha_{\gamma+\nu,1}A_{\gamma+\nu,1} \\ \alpha_{\gamma+\nu,2}A_{\gamma+\nu,2} \\ \cdots \\ \alpha_{\gamma+\nu,m}A_{\gamma+\nu,m} \end{pmatrix} \cdots \begin{pmatrix} \alpha_{s',1}A_{s',1} \\ \alpha_{s',2}A_{s',2} \\ \cdots \\ \alpha_{s',m}A_{s',m} \end{pmatrix}$$

such that $A_{\gamma,\mu} = A_{\gamma+\nu,\mu}$ for all $\mu = 1, \ldots, m$. If that is the case, and if $\ell_{t,2}$ coincides with the information stored in the symbol $\Lambda_{t,k}(D(w))$, then M removes the factor $\alpha_{\gamma+1,i_t} \cdots \alpha_{\gamma+\nu,i_t}$ from $g(i_t, j)$, it removes the corresponding cycle from ℓ_t, which yields the suffix ℓ'_t of an extended j-trace, and it replaces the information stored in $\Lambda_{t,k}$ by the suffix of ℓ'_t of length $p(\Pi)$. Observe that the factor $\alpha_{\gamma+1,i_t} \cdots \alpha_{\gamma+\nu,i_t}$ may well be empty, implying that this rewrite step simply replaces the symbol $\Lambda_{t,k}$ by a new symbol $\Lambda'_{t,k}$. Further, M stores ℓ_t in its finite control, in order to be able to verify at later occurrences of factors of the form $g(., j)$ that a consistent reduction is performed, and then M moves further to the right.

If no further factor of the form $g(., j)$ is encountered, then M restarts at the right end of the tape. If, however, another factor $g(i_{t'}, j_{t'}) = g(i', j)$ is found, then M tries to reduce this factor in a way consistent with the reduction applied to $g(i_t, j)$. Essentially, M processes the factor $g(i', j)$ in the same way as $g(i_t, j)$. However, on reaching the symbol $\Lambda_{t',k}$ it checks whether the current suffix $\ell_{t'}$ of the extended j-trace simulated coincides with the suffix ℓ_t stored in its finite control. In the affirmative it can then perform the same replacement in $\Lambda_{t',k}$ that it performed on $\Lambda_{t,k}$, and it can reduce the factor $g(i', j)$ in a way consistent with this replacement; otherwise, the computation is blocked.

► In an *accepting tail* M simply checks whether the current content w'_C of the tape belongs to the finite set of "shortest" characteristic words.

From the description above it follows that M is a nondeterministic aux-rewriting FRR-automaton, that the number of auxiliary symbols occurring in any restarting configuration of an accepting computation of M is bounded from above by the constant $2 \cdot g \cdot d + 2$, and that M performs at most d rewrite steps in any cycle of any computation. Further, it is quite clear that $L_{\mathrm{P}}(M) = L$ holds.

Finally, observe that M is in fact *correctness preserving*. For two different factors $g(i_t, j)$ and $g(i_{t'}, j)$ it may guess different extended j-traces, but because of the information stored in $\Lambda_{t,k} = \Lambda_{t',k}$, the suffixes of length $2 \cdot p(\Pi)$ of these traces coincide. Thus, as long as the corresponding suffix of the extended j-trace considered coincides with $\Lambda_{t,k}$, the reduction performed is consistent with a valid derivation $D(w)$. Further, if an inconsistency is discovered by M, then the computation is blocked immediately, that is, no further restart is performed. It follows that $w'_C \in L_{\mathrm{C}}(M)$ if $w_C \in L_{\mathrm{C}}(M)$ and $w_C \vdash^{\mathrm{c}}_M w'_C$ hold, that is, M is indeed correctness preserving. This completes the proof of Theorem 1. □

A detailed example with some additional explanations can be found in the technical report [4].

The AuxRR-automaton M described in the proof of Theorem 1 processes a given input by first choosing a particular derivation without cycles (and its communication structure) from among the finite set of possible derivations without

cycles by inserting delimiters. Then, in each cycle a specific generative section j is chosen nondeterministically, and the rightmost generative cycle is removed from each factor $g(i,j)$. In fact, each rewrite operation of each cycle executed by M replaces an auxiliary symbol of the form $A(D(w))$ by another auxiliary symbol of the form $A(D'(w'))$, and there is at least one rewrite operation in each cycle that removes a non-empty factor consisting of input symbols[4]. These observations motivate the following definition of a *skeletal set*.

Definition 2. *Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$ be a t-*AuxRR*-automaton, $r, s \in \mathbb{N}_+$, and let SP be a subalphabet of Γ of cardinality $|SP| \leq s \cdot r \cdot t$. We call SP a* skeletal set of type (r,t)*, if there is an injection $\phi : SP \to \{1,\ldots,s\} \times \{1,\ldots,r\} \times \{1,\ldots,t\}$ such that the properties below are satisfied:*

1. *Elements of SP are neither inserted, nor removed, nor changed during any computation of M; accordingly, we call them* islands.
2. *For all $w \in L_C(M)$ and all $\chi \in SP$, $|w|_\chi \leq 1$, that is, w contains at most one occurrence of χ.*
3. *For $1 \leq i \leq s$, let $SP(i) = \{\, \chi \in SP \mid \phi(\chi) = [i,a,b] \,\}$ be the i-th* skeleton *of SP. For each word $w \in L_C(M)$, there exists a unique index $i \in \{1,\ldots,s\}$ such that $\mathsf{Pr}^{SP}(w) \subseteq SP(i)$ holds. Thus, w only contains islands of a single skeleton.*
4. *Each rewrite operation O of M has the form $xyz\gamma\chi \to xz\gamma'\chi$, where $xyz \in \Sigma^*$, $|y| \geq 0$, $\gamma, \gamma' \in (\Gamma \setminus (\Sigma \cup SP))$ are auxiliary symbols that are not islands, and $\chi \in SP$ is an island. The symbol χ is called the* island visited *by O.*
5. *For $1 \leq i \leq s$ and $1 \leq j \leq r$, let $SP(i,j) = \{\, \chi \in SP \mid \phi(\chi) = [i,j,b] \,\}$, which is the j-th* level *of the i-th skeleton of SP. Within a cycle of a computation of M, the level of a skeleton is kept fixed, that is, if a rewrite operation O is applied in a cycle such that the island visited by O is from $SP(i,j)$, then for* every *rewrite operation executed during this cycle the island visited belongs to $SP(i,j)$.*
6. *There exists a constant $\ell(M)$ such that, for each $w = xyz \in L_C(M)$, where $|y| > \ell(M)$ and y does not contain any island, then starting from the restarting configuration corresponding to w, M will execute at least one cycle before it accepts.*

If SP is a *skeletal set* of type (r,t), then the auxiliary symbols in $\Gamma \setminus SP$ are called *variables* of M. Thus, Γ is partitioned into three disjoint subsets: the set of input symbols Σ, the skeletal set SP, and the set of variables.

Based on the properties of a skeletal set the following result can be derived similarly as in [1].

Corollary 1 (SP semi-linearity). *Let $t \in \mathbb{N}$ be a positive integer, and M be a t-*AuxRR*-automaton with a skeletal set. Then the languages $L_C(M)$ and $L_P(M)$ are semi-linear, that is, their Parikh images are semi-linear (see [5]).*

[4] All derivations considered are reduced.

Observe that the copy language $L_{copy} = \{ ww \mid w \in \{a,b\}^* \}$ can be generated by a returning PCGS with regular components and constant communication complexity, but that it cannot be generated by any centralized returning PCGS with regular components. Thus, Corollary 1 is not a special case of the corresponding result for the latter class of PCGSs given in [5].

Obviously, with almost no change the delimiters $\Delta_{0,k}$ and $\Delta_{1,k}$ in the proof of Theorem 1 can be shifted just before the delimiter $\Delta_{2,k}$. Thus, the set of delimiters of the form $\Theta_{1,k} = (\Delta_{0,k}, \Delta_{1,k}, \Delta_{2,k})$, and $\Theta_{t,k} = \Delta_{t+1,k}$ for $t > 1$, can serve as a skeletal set for a newly constructed automaton M'. The characteristic word w_C from $L_C(M')$ will then be of the following form:

$$w_C := g(i_1, j_1)\Lambda_{1,k}\Theta_{1,k}\ g(i_2, j_2)\Lambda_{2,k}\Theta_{2,k} \dots\ g(i_r, j_r)\Lambda_{r,k}\Theta_{r,k}.$$

Corollary 2. *For each $L \in g$-d-DDG, there exists a nondeterministic d-AuxRR-automaton M with a skeletal set SP of type (g,d) such that $L = L_P(M)$. Each variable of $w \in L_C(M)$ is positioned immediately to the left of an element of SP.*

The d-AuxRR-automaton M in Corollary 2 is inherently nondeterministic. To avoid this nondeterminism we now consider a slight generalization of the underlying FRR-automaton: the FRL-automaton. The FRL-automaton is obtained from the FRR-automaton by introducing *move-left* steps. For example, such an automaton can first scan its tape completely from left to right, then move back to the left end of the tape, and then perform some rewrite operations during a second left-to-right sweep. A d-AuxRL-automaton is an FRL-automaton that is aux-rewriting, correctness preserving, and that performs at most d rewrite operations in any cycle.

Obviously, the d-AuxRR-automaton M from Corollary 2 can be seen as a d-AuxRL-automaton of a restricted form. Hence, Theorem 3.4 in [6] applies, which states that there exists a deterministic d-FRL-automaton M_{det} that accepts the same characteristic language as M. In fact, if $w \vdash^c_{M_{det}} w'$, then also $w \vdash^c_M w'$ holds, and if $w \vdash^c_M w'$, then $w \vdash^c_{M_{det}} w''$ for some word w''. Since the transformation from M to M_{det} in the proof of Theorem 3.4 in [6] does not change the existence of a skeletal set, we can restate Corollary 2 as follows.

Corollary 3. *For each $L \in g$-d-DDG, there exists a deterministic d-AuxRL-automaton M with a skeletal set SP of type (g,d) such that $L = L_P(M)$. Moreover, in each $w_C \in L_C(M)$ each variable is positioned immediately to the left of an element of SP.*

Let M be a deterministic d-FRL-automaton. Given an input w of length n, M will execute at most n cycles, before it either accepts or rejects. Each of these cycles requires a number of steps that is linear in n. It follows that the membership problem for the language $L_C(M)$ is decidable in quadratic time.

If M is a deterministic d-AuxRL-automaton with a skeletal set SP of type (g,d), then an input word w of length n belongs to the proper language $L_P(M)$, if there exists an extended variant w_C of w that is in the characteristic language $L_C(M)$. From the form of the skeletal set we see that w_C is obtained from

w by inserting at most $g \cdot d$ factors of the form $\lambda\delta$, where λ is a variable and δ is an island. Hence, there are $O(|\Gamma \setminus \Sigma|^{2 \cdot g \cdot d} \cdot n^{g \cdot d})$ many candidates for w_C. They can all be enumerated systematically, and then for each of them membership in $L_C(M)$ can be tested in time $O((n + 2 \cdot g \cdot d)^2)$. Thus, we obtain the following.

Proposition 2. *Let M be a deterministic d-*AuxRL*-automaton with a skeletal set SP of the type (g, d) such that each variable in $w_C \in L_C(M)$ is positioned immediately to the left of an element of SP. Then, for each $w \in \Sigma^*$, the size of $A_C(w, M)$, the characteristic analysis of w by M, is at most $O(|\Gamma \setminus \Sigma|^{2 \cdot g \cdot d} \cdot n^{g \cdot d})$, and this set can be computed in time $O(|\Gamma \setminus \Sigma|^{2 \cdot g \cdot d} \cdot n^{g \cdot d} \cdot (n + 2 \cdot g \cdot d)^2)$.*

This proposition together with Corollary 3 has the following consequence.

Corollary 4. *For each language $L \subseteq \Sigma^*$, if $L \in g$-d-*DDG*, then there exists a d-*AuxRR*-automaton M such that $L = L_P(M)$, and for each $w \in \Sigma^*$, the size of the set $A_C(w, M)$ is at most $O(|\Gamma \setminus \Sigma|^{2 \cdot g \cdot d} \cdot n^{g \cdot d})$, and it can be computed in time $O(|\Gamma \setminus \Sigma|^{2 \cdot g \cdot d} \cdot n^{g \cdot d} \cdot (n + 2 \cdot g \cdot d)^2)$.*

Concluding remarks. Similar results as in this paper can be derived for PCGSs with linear-grammar components. On the other hand, that will surely not be the case for PCGSs with context-free components. Let us note that a polynomial upper bound for the (simple) membership problem for returning PCGSs with regular components already follows from [7]. Some results illustrating the generative power of the classes of PCGSs considered here are given already in [2] by the use of a deterministic variant of FRR-automata.

References

1. Pardubská, D., Plátek, M.: Parallel communicating grammar systems and analysis by reduction by restarting automata. In: Bel-Enguix, G., Jimenez-Lopez, M. (eds.) ForLing 2008. Proc. Research Group on Mathematical Linguistics, pp. 81–98. Universitat Rovira i Virgili, Tarragona (2008)
2. Pardubská, D., Plátek, M., Otto, F.: On PCGS and FRR-automata. In: Vojtáš, P. (ed.) ITAT 2008, Proc. Pavol Jozef Šafárik University, Košice, pp. 41–47 (2008)
3. Lopatková, D., Plátek, M., Sgall, P.: Towards a formal model for functional generative description - analysis by reduction and restarting automata. The Prague Bulletin of Mathematical Linguistics 87, 7–26 (2007)
4. Pardubská, D., Plátek, M., Otto, F.: On parallel communicating grammar systems and correctness preserving restarting automata. Kasseler Informatikschriften, Universität Kassel (2008), `https://kobra.bibliothek.uni-kassel.de/bitstream/urn:nbn:de:hebis:34-2008111825149/3/Technicalreport2008_4.pdf`
5. Czuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G. (eds.): Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Gordon and Breach Science Publishers (1994)
6. Messerschmidt, H., Otto, F.: On determinism versus nondeterminism for restarting automata. Information and Computation 206, 1204–1218 (2008)
7. Cai, L.: The computational complexity of PCGS with regular components. In: Dassow, J., Rozenberg, G., Salomaa, A. (eds.) Proc. of DLT 1995, pp. 209–219. World Scientific, Singapore (1996)

Finitely Generated Synchronizing Automata

Elena V. Pribavkina[1,⋆] and Emanuele Rodaro[2,⋆⋆]

[1] Ural State University, 620083, Ekaterinburg, Russia
[2] Universitá dell'Insubria, Via Valleggio 11, 22100 Como, Italy
elena.pribavkina@usu.ru, emanuele.rodaro@mate.polimi.it

Abstract. A synchronizing word w for a given synchronizing DFA is called *minimal* if no proper prefix or suffix of w is synchronizing. We characterize the class of synchronizing automata having finite language of minimal synchronizing words (such automata are called *finitely generated*). Using this characterization we prove that any such automaton possesses a synchronizing word of length at most $3n - 5$. We also prove that checking whether a given DFA $\mathscr{A}$ is finitely generated is co-NP-hard, and provide an algorithm for this problem which is exponential in the number of states $\mathscr{A}$.

1 Introduction

A *synchronizing* automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is a deterministic and complete finite-state automaton (DFA) possessing a *synchronizing* word, that is a word w which takes all states of $\mathscr{A}$ to a particular one: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$. By $\mathscr{L}(\mathscr{A})$ we denote the language of all words synchronizing $\mathscr{A}$.

Over the past forty years synchronizing automata and especially shortest synchronizing words have been widely studied, motivated mostly by the famous Černý conjecture [1] which states that any n-state synchronizing automaton possesses a synchronizing word of length at most $(n-1)^2$. This conjecture has been proved for a large number of classes of synchronizing automata, nevertheless in general it remains one of the most longstanding open problems in automata theory. For more details see the surveys [2,3,4].

In this paper we deal with *minimal synchronizing* words which in some sense generalize the notion of a shortest synchronizing word. Namely, a synchronizing word is called *minimal* if none of its proper prefixes nor suffixes is synchronizing. It is obvious that the language $\mathscr{L}(\mathscr{A})$ of all synchronizing words is a two-sided ideal generated by the language $\mathscr{L}_{min}(\mathscr{A})$ of all minimal synchronizing words: $\mathscr{L}(\mathscr{A}) = \Sigma^* \mathscr{L}_{min}(\mathscr{A}) \Sigma^*$. Thus it is rather natural to consider the class of synchronizing automata whose language of minimal synchronizing words is finite. Such automata are referred to as *finitely generated synchronizing automata* and

⋆ The author acknowledges support from CIMO Fellowship Programme TM-07-5127 during her stay at the University of Turku, Finland.

⋆⋆ Part of this work was done during the two visits of the second named author to the Ural State University. These visits were partially supported by the ESF project Automatha and the group GNSAGA of INDAM.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 672–683, 2009.

the class of such automata is denoted by **FG**. In Section 3 we give a characterization of this class. Moreover using the characterization we prove in Section 4 that the shortest synchronizing word for such automata has length at most $3n - 5$.

Example 1. Let $\Sigma = \{a, b\}$ and consider the minimal automaton $\mathscr{A}$ recognizing the language $\mathrm{L} = \Sigma^* aba \Sigma^*$ (see Fig. 1.)

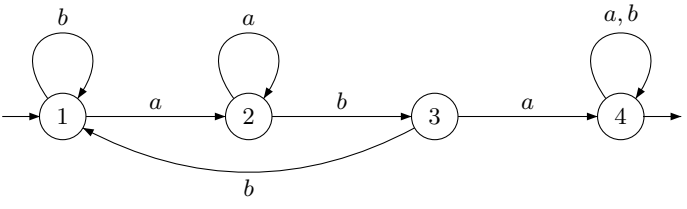


Fig. 1. Automaton $\mathscr{A}$ recognizing $\mathrm{L} = \Sigma^* aba \Sigma^*$

It is easy to see that $\mathscr{A}$ is synchronizing, and $\mathscr{L}(\mathscr{A}) = \mathrm{L}$, thus $\mathscr{L}_{min}(\mathscr{A}) = \{aba\}$, so $\mathscr{A} \in \mathbf{FG}$. Analogously for any $w \in \Sigma^*$ the minimal automaton recognizing $\Sigma^* w \Sigma^*$ is finitely generated. Moreover it is well-known that this automaton has $n = |w|+1$ states, hence its minimal synchronizing word is of length $n-1$. Clearly, in general, the minimal automaton recognizing the language $\Sigma^* M \Sigma^*$ for a finite language M is finitely generated.

Another natural question arising in this contest is whether testing $\mathscr{A} \in \mathbf{FG}$ is decidable. An easy argument shows that the answer is "yes". Indeed, in general, for a given language $\mathcal{L}$, we can consider the set $\mathcal{L}_{min}$ of its minimal words. Analogously it is the set of words from $\mathcal{L}$ such that none of its proper prefixes nor suffixes belongs to $\mathcal{L}$. It is not hard to see that $\mathcal{L}_{min} = \mathcal{L} \setminus (\Sigma^+ \mathcal{L} \cup \mathcal{L} \Sigma^+)$. In particular, if $\mathcal{L}$ is a two-sided ideal, this expression reduces to:

$$\mathcal{L}_{min} = \mathcal{L} \setminus (\Sigma \mathcal{L} \cup \mathcal{L} \Sigma). \tag{1}$$

Observe that if $\mathcal{L}$ is a regular language and it is represented by an n-state DFA, then clearly $\mathcal{L}_{min}$ is also a regular language recognized by an automaton with $O(n^3)$ states, and it is easy to see that checking finiteness of $\mathcal{L}_{min}$ takes $O(n^6)$ operations. In framework we are dealing the language of synchronizing words of a given automaton $\mathscr{A}$ is well known to be regular (it is recognized by the power automaton $\mathcal{P}(\mathscr{A})$ with Q as an initial state and singletons as terminal ones). Thus the language $\mathscr{L}(\mathscr{A})$ of an n-state synchronizing automaton $\mathscr{A}$ is recognized by an automaton with at most 2^n states. Hence by (1) the language $\mathscr{L}_{min}(\mathscr{A})$ is regular and checking whether $\mathscr{A} \in \mathbf{FG}$ takes $O(2^{6n})$ operations. Therefore one may ask whether checking finiteness of the language of minimal synchronizing words is indeed a hard task and if there are better algorithms than the naive one. We formally state the `FINITENESS` problem:

- *Input*: A synchronizing DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.
- *Question*: Is the language $\mathscr{L}_{min}(\mathscr{A})$ finite?

Our characterization gives rise to another algorithm for solving FINITENESS which is a slight improvement of the naive one, it is discussed in Section 3. Furthermore in Section 5 we show that FINITENESS is co-NP-hard. So the problem is not likely to have a polynomial time algorithm. Pawel Gawrychowski [5] has shown that this problem is in PSPACE, but we are not aware whether it belongs to some lower complexity class.

2 Preliminaries

We fix a synchronizing DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$. For convenience for each $v \in \Sigma^*$ and $q \in Q$ we will write $q \,.\, v = \delta(q, v)$ and put $S \,.\, v = \{q \,.\, v \mid q \in S\}$ for any $S \subseteq Q$.

A subset S of Q is called *reachable* if there is a word $v \in \Sigma^*$ with $S = Q \,.\, v$.

Given a subset S of Q by $\mathcal{C}(S)$ we denote the set of all words *stabilizing* S:

$$\mathcal{C}(S) = \{w \in \Sigma^* \mid S \,.\, w = S\}.$$

By $\mathcal{R}(S)$ we denote the set of all words bringing S to a singleton:

$$\mathcal{R}(S) = \{w \in \Sigma^* \mid |S \,.\, w| = 1\}.$$

Note that $\mathscr{L}(\mathscr{A})$ is contained in $\mathcal{R}(S)$ for any S.

Lemma 1. *Given a word $w \in \Sigma^*$ there exists an integer $\beta \geq 0$ such that the set $m(w) = Q \,.\, w^\beta$ is fixed by w. Moreover $m(w)$ is the largest subset of Q with this property.*

Proof. Consider the sets $Q \,.\, w^\alpha \subseteq Q$ for any $\alpha \geq 0$. Since the number of subsets of Q is finite, there are $\beta \geq 0$ and $\gamma > 0$ such that $Q \,.\, w^\beta = Q \,.\, w^{\beta+\gamma}$. It is easy to see that

$$Q \,.\, w^\beta \supseteq Q \,.\, w^{\beta+1} \supseteq \ldots \supseteq Q \,.\, w^{\beta+\gamma} = Q \,.\, w^\beta.$$

Hence all inclusions are in fact equalities and in particular $Q \,.\, w^{\beta+1} = Q \,.\, w^\beta$, so the set $Q \,.\, w^\beta$ is fixed by the word w. On the other hand take any $S \subseteq Q$ fixed by w, then applying w we obtain

$$S = S \,.\, w \subseteq Q \,.\, w, \ldots, S = S \,.\, w^\beta \subseteq Q \,.\, w^\beta,$$

so $m(w)$ is the largest subset fixed by the word w. □

Given a word w, the subset $m(w)$ of Q from the previous Lemma is called *the maximal fixed set with respect to w*.

Let $k(w)$ be the least integer with the property $Q \,.\, w^{k(w)} = m(w)$. Then we have the following

Lemma 2. *Given a word $w \in \Sigma^*$*

$$k(w) \leq |Q| - |m(w)|.$$

Proof. We have $|Q\,.\,w^{\beta+1}| < |Q\,.\,w^{\beta}|$ for any $0 \leq \beta < k(w)$. Indeed, suppose $|Q\,.\,w^{\beta+1}| = |Q\,.\,w^{\beta}|$ for some $0 \leq \beta < k(w)$. Since $Q\,.\,w^{\beta+1} \subseteq Q\,.\,w^{\beta}$ we have $Q\,.\,w^{\beta+1} = Q\,.\,w^{\beta}$, hence $Q\,.\,w^{\beta} \subseteq m(w)$. On the other hand, $m(w) = Q\,.\,w^{k(w)} \subseteq Q\,.\,w^{\beta}$, thus $m(w) = Q\,.\,w^{\beta}$, which is a contradiction with the choice of $k(w)$. Therefore $|Q| > |Q\,.\,w| > \ldots > |Q\,.\,w^{k(w)}|$, and $|Q\,.\,w^{k(w)}| \leq |Q| - k(w)$, hence $k(w) \leq |Q| - |m(w)|$. □

Lemma 3. *Given a word $w \in \Sigma^*$ and $\alpha \in \mathbb{N}$*

$$m(w^{\alpha}) = m(w).$$

Proof. Obviously $w^{\alpha} \in \mathcal{C}(m(w))$ for any $\alpha \in \mathbb{N}$, hence $m(w) \subseteq m(w^{\alpha})$. Conversely, by Lemma 1 $m(w^{\alpha}) = Q\,.\,w^{\alpha\beta}$ for some $\beta \geq 0$. Since $m(w^{\alpha})$ is fixed by w^{α}, we have $|m(w^{\alpha})\,.\,w^{\alpha}| = |m(w^{\alpha})|$ in particular; thus $|m(w^{\alpha})\,.\,w| = |m(w^{\alpha})|$ and hence $|Q\,.\,w^{\alpha\beta+1}| = |Q\,.\,w^{\alpha\beta}|$. On the other hand, $Q\,.\,w^{\alpha\beta+1} \subseteq Q\,.\,w^{\alpha\beta}$, thus $Q\,.\,w^{\alpha\beta+1} = Q\,.\,w^{\alpha\beta}$, so $m(w^{\alpha})$ is fixed by w as well, and $m(w^{\alpha}) \subseteq m(w)$. Therefore $m(w^{\alpha}) = m(w)$. □

3 The Class FG

In this section we characterize the class **FG** and provide another algorithm for checking whether a given DFA is in **FG**.

Theorem 1. *Given a synchronizing DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ the following are equivalent:*

(i) $\mathscr{A} \in \boldsymbol{FG}$
(ii) for any reachable subset $S \subseteq Q$ such that $1 < |S| < |Q|$, for each $w \in \mathcal{C}(S)$

$$\mathcal{R}(S) = \mathcal{R}(m(w)).$$

Proof. $(i) \Rightarrow (ii)$. Consider an arbitrary reachable subset $S \subset Q$ with $1 < |S| < |Q|$. If $\mathcal{C}(S) = \varnothing$ then there is nothing to prove, so we can assume that $\mathcal{C}(S) \neq \varnothing$ and take an arbitrary $w \in \mathcal{C}(S)$. Note, that the inclusion $\mathcal{R}(m(w)) \subseteq \mathcal{R}(S)$ always holds true. Indeed, the set S is contained in the maximal fixed set $m(w)$, and all the words synchronizing the set $m(w)$ synchronize also S.

In case $\mathcal{R}(S) = \mathscr{L}(\mathscr{A})$ we have $\mathscr{L}(\mathscr{A}) \subseteq \mathcal{R}(m(w)) \subseteq \mathscr{L}(\mathscr{A})$, therefore $\mathcal{R}(m(w)) = \mathscr{L}(\mathscr{A})$, and the equality holds.

Suppose now that $\mathcal{R}(S) \neq \mathscr{L}(\mathscr{A})$. It means that there exists a word v which brings the set S to a singleton, but does not synchronize the whole automaton. On the other hand, since the set S is reachable there exists a word u such that $S = Q\,.\,u$. Consider now the infinite sequence of words uw^iv for $i \geq 0$. Note that these words are synchronizing, and since $\mathscr{L}_{min}(\mathscr{A})$ is finite, among them there can be only a finite number of minimal synchronizing words. By the choice of u and w such a minimal synchronizing word always contains a prefix of v as a suffix. Note also that if all the minimal synchronizing words in the given sequence would start with some suffix of u, then there would be an infinite sequence of

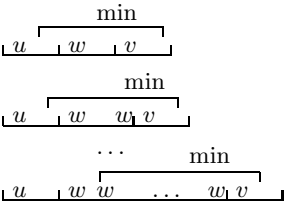


Fig. 2. Finite Sequence of Minimal Synchronizing Words

minimal synchronizing words $u_i'' w^i v$ where $u = u_i' u_i''$ for $i = 0, 1, 2 \ldots$. Therefore there exists a positive integer β such that the word $w^\beta v$ is synchronizing (not necessarily minimal, see Fig. 2).

By Lemma 1 as β we can choose an integer such that $Q \cdot w^\beta = m(w)$, thus v brings this set to a singleton. Thus if the language $\mathscr{L}_{min}(\mathscr{A})$ is finite then $\mathcal{R}(S) \subseteq \mathcal{R}(m(w))$. Since the opposite inclusion always holds true, we have $\mathcal{R}(S) = \mathcal{R}(m(w))$.

$(ii) \Rightarrow (i)$. Arguing by contradiction suppose that the condition (ii) holds, but $\mathscr{L}_{min}(\mathscr{A})$ is infinite. Since this language is regular, applying the pumping lemma we have that any long enough word in $\mathscr{L}_{min}(\mathscr{A})$ can be factorized as uwv so that $w \neq 1$ (where 1 denotes the empty word) and $uw^\alpha v$ is in $\mathscr{L}_{min}(\mathscr{A})$ for any integer $\alpha \geq 0$. If $u = 1$ then $w^\alpha v \in \mathscr{L}_{min}(\mathscr{A})$ for all $\alpha \geq 0$, in particular for $\alpha = 0$ we obtain that the word v is synchronizing, but this means that the words $w^\alpha v$ do not belong to $\mathscr{L}_{min}(\mathscr{A})$, a contradiction. Analogously we get a contradiction in case $v = 1$. Thus we may assume that both u and v are non-empty words. Consider sets $Q \cdot uw^\alpha$ for $\alpha \geq 0$. Since $uw^\alpha v$ is minimal synchronizing, we have $1 < |Q \cdot uw^\alpha| < |Q|$. Since the number of subsets of Q is finite there are integers $\alpha_0 \geq 0$ and $\gamma > 0$ such that $Q \cdot uw^{\alpha_0} = Q \cdot uw^{\alpha_0 + \gamma}$. Put $S = Q \cdot uw^{\alpha_0}$. This set is fixed by w^γ, so by (ii) we have $\mathcal{R}(S) = \mathcal{R}(m(w^\gamma))$, with $m(w^\gamma) = Q \cdot (w^\gamma)^\beta$ for some $\beta \geq 0$. Since $v \in \mathcal{R}(S)$ we have $v \in \mathcal{R}(m(w^\gamma))$, so $|Q \cdot w^{\gamma\beta} v| = 1$. Therefore the word $w^{\gamma\beta} v$ is synchronizing and for $\alpha > \gamma\beta$ the word $uw^\alpha v$ does not belong to the language $\mathscr{L}_{min}(\mathscr{A})$, again a contradiction. Thus the language $\mathscr{L}_{min}(\mathscr{A})$ is finite. □

Corollary 1. *Let $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ be a synchronizing automaton such that there is a letter $a \in \Sigma$ with $Q \cdot a = Q$ and there are no letters $b \in \Sigma$ with $|Q \cdot b| = 1$. Then $\mathscr{L}_{min}(\mathscr{A})$ is infinite.*

Proof. Consider the shortest synchronizing word w for the automaton $\mathscr{A}$. Since no letter is synchronizing we have $w = xv$ with $x \in \Sigma$, $v \in \Sigma^+$, and $1 < |Q \cdot x| < Q$. Since a is a permutation letter it is clear that $a^\alpha \in \mathcal{C}(Q \cdot x)$ for some positive integer α. On the other hand $Q \cdot a^\alpha = Q$, hence $m(a^\alpha) = Q$. Note that $\mathcal{R}(Q \cdot x) \neq \mathcal{R}(Q)$ since $v \in \mathcal{R}(Q \cdot x)$ but not in $\mathcal{R}(Q) = \mathscr{L}(\mathscr{A})$ (otherwise it would be a shorter synchronizing word). Thus by Theorem 1 the language $\mathscr{L}_{min}(\mathscr{A})$ is infinite. □

Theorem 1 gives rise to an algorithm FinCheck different from the straightforward one presented in section 1 for the problem of recognizing finitely

generated synchronizing automata. Actually Theorem 1 says that for a given reachable subset S of Q for all the words $w \in \mathcal{C}(S)$ we must check whether $\mathcal{R}(S) = \mathcal{R}(m(w))$. The problem is that the set $\mathcal{C}(S)$ might be infinite. On the other hand there are only finitely many subsets of Q of the form $m(w)$ for all $w \in \mathcal{C}(S)$. So for a given S we can check the property of Theorem 1 for all the subsets T of Q containing S with $\mathcal{C}(S) \cap \mathcal{C}(T) \neq \varnothing$. Indeed, obviously among such subsets there are those of the form $m(w)$ for all possible w. So if $\mathcal{R}(S) = \mathcal{R}(T)$ for all such T, then the condition of the theorem holds. If $\mathcal{R}(S) \neq \mathcal{R}(T)$ for some $T \supseteq S$, then it does not hold for $m(w)$ either, $w \in \mathcal{C}(S) \cap \mathcal{C}(T) \neq \varnothing$, since $T \subseteq m(w)$. Now we present the algorithm.

FinCheck($\mathscr{A}$):

1 From the DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ construct its power automaton $\mathcal{P}(\mathscr{A})$ consisting only of subsets reachable from Q.
2 For each state S of $\mathcal{P}(\mathscr{A})$ do:
  2.1 For each T of $\mathcal{P}(\mathscr{A})$ with $S \subseteq T$ do:
    2.2 If $C(T) \cap C(S) \neq \varnothing$, then
      2.3 If $\mathcal{R}(T) \neq \mathcal{R}(S)$, then exit and return NO
3 Otherwise exit and return YES

A rather technical calculation shows that the cost of this algorithm is of $O\left(2^{2n}3^n\right)$ operations, so it is slightly better than the straight-forward algorithm but still is exponential in the number of states of the initial automaton.

4 Černý's Conjecture for the Class FG

Using the characterization from the previous section here we show a linear upper bound on the length of the shortest synchronizing word for the class of finitely generated synchronizing automata.

Recall that *the deficiency* of a word $w \in \Sigma^*$ with respect to a given automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is the difference $\mathrm{df}(w) = |Q| - |Q \,.\, w|$. We make use of the following result from [6] (see also [7]):

Theorem 2. *Given a synchronizing automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ and the words $u, v \in \Sigma^+$ such that $\mathrm{df}(u) = \mathrm{df}(v) = k > 1$, there exists a word τ, with $|\tau| \leq k+1$, such that $\mathrm{df}(u\tau v) > k$.*

Theorem 3. *Let $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ be a finitely generated synchronizing automaton with n states. There is a synchronizing word of length at most $3n - 5$.*

Proof. Take any $a \in \Sigma$. By Corollary 1, if a acts as a permutation on the state set Q, then there exists a synchronizing letter b, so the statement of the theorem trivially holds. Thus we can assume that a is not a permutation letter, thus $Q \,.\, a^{k(a)} = m(a) \subsetneq Q$. Suppose first that $|m(a)| = 1$, then $1 = |m(a)| = |Q \,.\, a^{k(a)}|$ and by Lemma 2, $k(a) \leq n - |m(a)| = n - 1 \leq 3n - 5$ for $n \geq 2$.

Now suppose that $|m(a)| > 1$, and consider non-singleton subsets of $m(a)$ reachable from $m(a)$:

$$\textsc{Reach}(a) = \{S \subseteq m(a) \mid S = m(a)\,.\,u, u \in \Sigma^*, |S| > 1\}.$$

Obviously $m(a) \in \textsc{Reach}(a)$, so $\textsc{Reach}(a)$ is not empty. Also note that for any $S \in \textsc{Reach}(a)$ it holds

$$\mathcal{R}(S) = \mathcal{R}(m(a)). \tag{2}$$

Indeed, since $S \subseteq m(a)$ we have $a^\alpha \in \mathcal{C}(S)$ for some $\alpha \in \mathbb{N}$. Then since $\mathscr{A}$ is finitely generated applying Theorem 1 and Lemma 3 we get $\mathcal{R}(S) = \mathcal{R}(m(a^\alpha)) = \mathcal{R}(m(a))$

Now let $H = Q\,.\,a^{k(a)}u$ be an element of $\textsc{Reach}(a)$ of minimal cardinality and let $k' = n - |H| = df(a^{k(a)}u)$. Since $|H| > 1$ we have $k' \leq n-2$. Since $\mathscr{A}$ is synchronizing, there exists a word of deficiency $n-1$, therefore by Theorem 2 there is a word τ with $|\tau| \leq k'+1$ such that $\mathrm{df}(a^{k(a)}u\tau a^{k(a)}u) > k'$, i.e. $|Q\,.\,a^{k(a)}u\tau a^{k(a)}u| < |H|$. Next we prove that the word $a^{k(a)}u\tau a^{k(a)}ua^{ka}$ is synchronizing. Indeed, suppose on the contrary that $|Q\,.\,a^{k(a)}u\tau a^{k(a)}ua^{k(a)}| > 1$. It is easy to see that $Q\,.\,a^{k(a)}u\tau a^{k(a)}ua^{k(a)} \subseteq m(a)$, hence $Q\,.\,a^{k(a)}u\tau a^{k(a)}ua^{k(a)} \in \textsc{Reach}(a)$. However the inequality

$$|Q\,.\,a^{k(a)}u\tau a^{k(a)}ua^{k(a)}| \leq |Q\,.\,a^{k(a)}u\tau a^{k(a)}u| < |H|$$

contradicts the choice of H. In fact even the word $a^{k(a)}u\tau a^{k(a)}$ is synchronizing. Indeed, consider the set $S = Q\,.\,a^{k(a)}u\tau a^{k(a)}$. If $|S| > 1$ then $S \in \textsc{Reach}(a)$, hence $ua^{k(a)} \in \mathcal{R}(S) = \mathcal{R}(m(a))$, so $1 = |m(a)\,.\,ua^{k(a)}| = |H\,.\,a^{k(a)}|$. But by the choice of H we have $|H\,.\,a^{k(a)}| = |H| > 1$, which is a contradiction.

Thus

$$\tau a^{k(a)} \in \mathcal{R}(Q\,.\,a^{k(a)}u) = \mathcal{R}(H) = \mathcal{R}(m(a)) = \mathcal{R}(Q\,.\,a^{k(a)}),$$

hence the word $a^{k(a)}\tau a^{k(a)}$ is synchronizing and $|a^{k(a)}\tau a^{k(a)}| \leq 2k(a) + k' + 1 \leq 2(n-2) + n - 1 = 3n - 5$. □

Remark 1. Under conditions of the Theorem 3 if $k = \min\limits_{a \in \Sigma} k(a)$, then there is a synchronizing word of length at most $2k + n - 1$.

5 Co-NP-Hardness

In this section we prove that the `FINITENESS` problem for a given DFA is co-NP-hard. To prove the result we introduce another problem which we refer to as `REACHABILITY`. Formally:

Input: A DFA $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$ and a subset $H \subseteq Q$.
Question: Is there a word $w \in \Sigma^*$ such that $Q\,.\,w = H$?

The proof proceeds in two stages. First we show that any instance from a particular set I of instances of `REACHABILITY` can be polynomially reduced to an

instance of the complement of FINITENESS. Next we complete the proof by polynomially reducing any instance of SAT to an instance of REACHABILITY belonging to I.

In our reductions we essentially make use of a particular class of automata, called *nilpotent* automata. This notion was introduced by Perles et al. in 1962 under the name of *definite table* [8]. Later such automata were studied by Rystsov in [9] in view of Černý's conjecture. In the present paper we use the definition from [9]. Namely, we say that a DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is *nilpotent* if there is a state $q \in Q$ and a positive integer n such for any word $w \in \Sigma^*$ of length at least n it holds $Q \,.\, w = \{q\}$.

Obviously any nilpotent automaton is synchronizing with a *sink state*, i.e. the state fixed by all the letters of the alphabet (q in the definition). In the following lemma we state without proof some simple known properties of nilpotent automata.

Lemma 4. *Let $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ be a DFA with a unique sink state.*

(1) If $\mathscr{A}$ is nilpotent, then for any word $w \in \Sigma^+$ there exists a positive integer m such that $Q \,.\, w^m = \{q\}$.
(2) $\mathscr{A}$ is nilpotent iff there are no cycles or loops passing through non-sink states.

Next we introduce a particular subclass of nilpotent automata and a particular set I of instances of REACHABILITY.

Consider a nilpotent automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ with a sink state $\mathfrak{p}$ and a non-empty set $\mathcal{R}_{\mathscr{A}}$ of states $\mathfrak{r}$ satisfying the following conditions:

- $\mathfrak{r} \,.\, \Sigma = \mathfrak{p}$;
- if there exists a word w such that $Q \,.\, w = \{\mathfrak{p}, \mathfrak{r}\}$, then there exists such word of length at least 2.

We denote this class of automata by $\mathcal{N}$. Note that the language $\mathscr{L}_{min}(\mathscr{A})$ is finite for every $\mathscr{A} \in \mathcal{N}$. It is an easy consequence of Theorem 1 and Lemma 4.

The set I is defined as follows:

$$I = \{(\mathscr{A}, H) \mid \mathscr{A} \in \mathcal{N}, H = \{\mathfrak{p}, \mathfrak{r}\}, \mathfrak{r} \in \mathcal{R}_{\mathscr{A}}\}.$$

The next proposition polynomially reduces any instance of REACHABILITY belonging to I to an instance of the complement of FINITENESS. Its proof is rather technical and due to the space limitation is omitted here.

Proposition 1. *Let $(\mathscr{A}, H) \in I$ be an instance of* REACHABILITY*, with $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ and $H = \{\mathfrak{p}, \mathfrak{r}\}$, $\mathfrak{r} \in \mathcal{R}_{\mathscr{A}}$. Then there is a synchronizing automaton $\mathscr{A}'$ such that the language $\mathscr{L}_{min}(\mathscr{A}')$ is infinite if and only if there exists $w \in \Sigma^+$ such that $Q \,.\, w = H$.*

In the sequel we polynomially reduce an instance of SAT to an instance of REACHABILITY belonging to I. To this end we present another auxiliary construction. Recall that a cartesian product of n deterministic finite automata

$\mathscr{A}_i = \langle Q_i, \Sigma, \delta_i \rangle$ is a DFA $\times_{i=1}^{n} \mathscr{A}_i = \langle Q, \Sigma, \delta \rangle$ with $Q = Q_1 \times Q_2 \times \cdots \times Q_n$, and a transition function $\delta : Q \times \Sigma \to Q$ defined component-wise

$$\delta((q_1, q_2, \ldots, q_n), a) = (\delta_1(q_1, a), \delta_2(q_2, a), \ldots, \delta_n(q_n, a)).$$

Clearly this action extends in a natural way to the free monoid Σ^*. It is not difficult to check that the following statement holds

Lemma 5. *Let $\mathscr{A}_1, \ldots, \mathscr{A}_m$ be nilpotent automata over a fixed alphabet with the sink states $q_1, \ldots, q_m$ respectively. Then their cartesian product $\times_{i=1}^{m} \mathscr{A}_i$ is also nilpotent with a sink state $q = (q_1, \ldots, q_m)$.*

Next we fix a positive integer $n \geq 2$ and for each such n construct two particular nilpotent automata over the alphabet $\Sigma_n = \{a_1, b_1, \ldots, a_n, b_n\}$.

For convenience let $\gamma_i = \{a_i, b_i\}$ (in the sequel γ_i will correspond to the two possible values for the i^{th} variable). Consider an automaton $A_n = \langle Q_A, \Sigma_n, \delta_A \rangle$ with $Q_A = \{0, 1, \ldots, n, p_1\}$ and the transition function defined as follows (Fig. 3):

$i \,.\, \gamma_{i+1} = i + 1$ for $0 \leq i \leq n-1$, $\quad i \,.\, (\Sigma_n \setminus \gamma_{i+1}) = p_1$ for $0 \leq i \leq n-1$,
$n \,.\, \Sigma_n = p_1$, $\quad p_1 \,.\, \Sigma_n = p_1$.

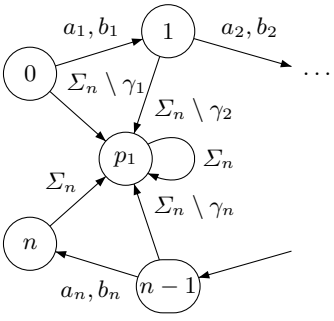


Fig. 3. Automaton A_n

Next let $B_n = \langle Q_B, \Sigma_n, \delta_B \rangle$ with $Q_B = \{0, 1, \ldots, n\}$ and the following transition rules (Fig. 4): $i \,.\, \Sigma_n = i + 1$ for $0 \leq i \leq n-1$ and $n \,.\, \Sigma_n = n$.

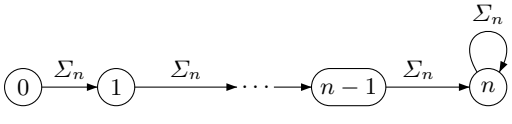


Fig. 4. Automaton B_n

Now we construct their cartesian product $\mathcal{V}_n = A_n \times B_n$ which has $(n+1)(n+2)$ states and will serve as a gadget encoding the correct assignment of the values to variables. The following proposition establishes the main properties of this automaton.

Proposition 2. *The automaton* $\mathcal{V}_n = A_n \times B_n = \langle Q_{A,B}, \Sigma_n, \delta_{A,B}\rangle$ *is nilpotent with the sink state* $p = (p_1, n)$ *and the state* $r = (n, n)$ *satisfies* $r \,.\, \Sigma_n = p$. *Moreover* $Q_{A,B} \,.\, w = \{p, r\}$ *if and only if* $w = x_1 \dots x_n$ *with* $x_i \in \gamma_i$.

Proof. By Lemma 5 $\mathcal{V}_n$ is nilpotent with the sink state $p = (p_1, n)$. The state $r = (n, n)$ satisfies $r \,.\, \Sigma_n = p$.

Suppose that $Q_{A,B} \,.\, w = \{p, r\} = \{p_1, n\} \times \{n\}$. It is obvious that for any word w it holds $Q_{A,B} \,.\, w = Q_A \,.\, w \times Q_B \,.\, w$. Thus we have $Q_A \,.\, w = \{p_1, n\}$ and $Q_B \,.\, w = \{n\}$. An inspection of the automaton B_n shows that $Q_B \,.\, w = \{n\}$ if and only if $|w| \geq n$. Suppose that $w = x_j w'$ for some $x_j \in \gamma_j$, then by the definition of the automaton A_n, we get $Q_A \,.\, x_j = \{p_1, j\}$. Hence, not to kill the state j and to lead it till the state n, we must have $w = x_j x_{j+1} \dots x_n$, $x_i \in \gamma_i$ for all $i \geq j$. Combining this with the condition $|w| \geq n$ we get $j = 1$, i.e. $w = x_1 \dots x_n$ with $x_i \in \gamma_i$ for all $i \geq 1$.

Conversely, let $w = x_1 \dots x_n$ with $x_i \in \gamma_i$. An easy computation shows that $Q_A \,.\, w = \{n, p_1\}$ and $Q_B \,.\, w = \{n\}$. Thus $\{p, r\} = \{p_1, n\} \times \{n\} = Q_A \,.\, w \times Q_B \,.\, w = Q_{A,B} \,.\, w = \{p, r\}$. □

Let an instance of SAT consist of m clauses $\chi = \{c_1, \dots, c_m\}$ over n variables $X_1, \dots, X_n$. Without loss of generality we can assume $n \geq 2$.

Next for each $x_i \in \gamma_i = \{a_i, b_i\}$, $1 \leq i \leq n$ we define $\chi(x_i) = \{c_{i_1}, \dots, c_{i_k}\}$ to be the subset of χ consisting of clauses which contain positive literal X_i if $x_i = a_i$, and of clauses containing negative literal $\neg X_i$ if $x_i = b_i$. Without loss of generality we can assume that $\chi(a_i) \cap \chi(b_i) = \varnothing$, otherwise the common clause $c_k \in \chi(a_i) \cap \chi(b_i)$ would contain both X_i and $\neg X_i$, and so it would be trivially satisfied. Moreover we can assume that all such subsets are non-empty. Indeed if some $\chi(a_i) = \varnothing$, then all the clauses contain only negative literal $\neg X_i$, hence we can put $X_i = 0$ and reduce the problem to one with less variables.

We say that the set $\{x_1, \dots, x_n\}$ with $x_i \in \gamma_i$ for $i = 1, \dots, n$ is a *satisfiable assignment* for χ if and only if

$$\bigcup_{i=1}^{n} \chi(x_i) = \chi.$$

With these definitions, it is clear that to a satisfiable assignment $\{x_1, \dots, x_n\}$ corresponds a satisfiable assignment for the Boolean formula $\wedge_{i=1}^{m} c_i$ given by $X_i = 1$ if $x_i = a_i$, and $X_i = 0$ if $x_i = b_i$, and vice versa.

We now define a nilpotent automaton $\mathcal{S}(c_1, \dots, c_m) = \langle S, \Sigma_n, \Delta\rangle$ associated to the clauses $c_1, \dots, c_m$. This automaton is a gadget which chooses satisfiable assignments from all possible ones. This automaton has $m(n+1)+1$ states with $S = \chi^1 \cup \chi^2 \cup \dots \chi^n \cup \chi \cup \{p'\}$ where $\chi^i = \{c_1^i, \dots, c_m^i\}$ for $i = 1, \dots, n$ are copies of $\chi = \{c_1, \dots, c_m\}$. We denote by $\chi^i(x_k)$ the subset of χ^i which is the corresponding copy of $\chi(x_k)$. The action of Δ on S is defined as follows. For each $1 \leq i \leq n-1$, each $1 \leq j \leq m$, and $c_j^i \in \chi^i$ we have

if $c_j^i \in \chi^i(a_i)$ then $c_j^i \,.\, a_i = p'$, and $c_j^i \,.(\Sigma_n \setminus \{a_i\}) = c_j^{i+1} \in \chi^{i+1}$,
if $c_j^i \in \chi^i(b_i)$ then $c_j^i \,.\, b_i = p'$ and $c_j^i \,.(\Sigma_n \setminus \{b_i\}) = c_j^{i+1} \in \chi^{i+1}$,

if $c_j^i \in \chi^i \setminus (\chi^i(a_i) \cup \chi^i(b_i))$ then $c_j^i \,.\, \Sigma_n = c_j^{i+1} \in \chi^{i+1}$;
if $c_j^n \in \chi^n(a_n)$ then $c_j^n \,.\, a_n = p'$ and $c_j^n \,.(\Sigma_n \setminus \{a_n\}) = c_j \in \chi$,
if $c_j^n \in \chi^n(b_n)$ then $c_j^n \,.\, b_n = p'$ and $c_j^n \,.(\Sigma_n \setminus \{b_n\}) = c_j \in \chi$,
if $c_j^n \in \chi^n \setminus (\chi^n(a_n) \cup \chi^n(b_n))$ then $c_j^n \,.\, \Sigma_n = c_j \in \chi$,
$\chi \,.\, \Sigma_n = p'$ and $p' \,.\, \Sigma_n = p'$.

Clearly $\mathcal{S}(c_1, \ldots, c_m)$ is nilpotent with the sink state p'. The property stated in the following lemma is rather clear from the construction, thus we omit here the formal proof:

Lemma 6. *Let $\mathcal{S}(c_1, \ldots, c_m) = \langle S, \Sigma_n, \Delta \rangle$ be constructed as above and $w = x_1 \ldots x_n \in \Sigma_n^+$ be a word with $x_i \in \gamma_i$ for $i = 1, \ldots, n$, then $S \,.\, w = \{p'\}$ if and only if $x_1, \ldots, x_n$ is a satisfiable assignment.*

Proposition 3. *Let $c_1, \ldots, c_m$ be clauses over $n \geq 2$ variables. The automaton $\mathscr{A} = \mathcal{V}_n \times \mathcal{S}(c_1, \ldots, c_m) = \langle Q, \Sigma_n, \delta \rangle$ belongs to $\mathcal{N}$. Moreover, putting $\mathfrak{p} = (p, p')$, $\mathfrak{r} = (r, p')$, and $H = \{\mathfrak{p}, \mathfrak{r}\}$, we have $(\mathscr{A}, H) \in I$ and there is a word $w \in \Sigma_n^+$ such that $Q \,.\, w = H$ if and only if the Boolean formula $\wedge_{i=1}^m c_i$ is satisfiable.*

Proof. By Lemma 5, $\mathscr{A} = \langle Q, \Sigma_n, \delta \rangle$ is nilpotent since both $\mathcal{V}_n$ and $\mathcal{S}(c_1, \ldots, c_m)$ are nilpotent automata. Moreover, with the notation of Proposition 2, $\mathfrak{p} = (p, p')$ is the sink state for $\mathscr{A}$ and since $r \,.\, \Sigma_n = p$ and $p' \,.\, \Sigma_n = p'$, we obtain that the state $\mathfrak{r} = (r, p')$ satisfies $\mathfrak{r} \,.\, \Sigma_n = \mathfrak{p}$.

Let us first prove that $Q \,.\, w = H = \{\mathfrak{p}, \mathfrak{r}\}$ if and only if $\wedge_{i=1}^m c_i$ is satisfiable. Since $\{\mathfrak{p}, \mathfrak{r}\} = \{p, r\} \times \{p'\}$, then $Q \,.\, w = Q_{A,B} \,.\, w \times S \,.\, w$. Thus $Q \,.\, w = H$ if and only if $Q_{A,B} \,.\, w = \{p, r\}$ and $S \,.\, w = \{p'\}$. On the other hand, by Proposition 2, $Q_{A,B} \,.\, w = \{p, r\}$ if and only if $w = x_1 \ldots x_n$ with $x_i \in \gamma_i$ for $i = 1, \ldots, n$ and, by Lemma 6, $S \,.\, w = \{p'\}$ if and only if $\{x_1, \ldots, x_n\}$ is a satisfiable assignment.

Now we prove that $\mathcal{R}_{\mathscr{A}}$ is not empty. By its definition if there is a word w such that $Q \,.\, w = \{\mathfrak{p}, \mathfrak{r}\}$, then the previous argument and the condition $n \geq 2$ imply $|w| \geq 2$. So $\mathfrak{r} \in \mathcal{R}_{\mathscr{A}}$, hence $\mathscr{A} \in \mathcal{N}$. □

Now we are ready to state the main theorem of this section.

Theorem 4. *The problem* FINITENESS *is co-NP-hard.*

Proof. Let $\{c_1, \ldots, c_m\}$ be an instance of SAT with $X_1, \ldots, X_n$ variables, $n \geq 2$. Combining Propositions 1 and 3 we obtain the automaton $\mathscr{A}' = \langle Q', \Sigma_n, \delta' \rangle$ over an alphabet with $2n + 1$ symbols and having $(n + 1)(n + 2)(m(n + 1) + 1) + 2$ states, such that $\mathscr{L}_{min}(\mathscr{A}')$ is infinite if and only if $\wedge_{i=1}^m c_i$ is satisfiable. □

Remark 2. Note that in our reduction the size of the alphabet is not constant and depends on the input. In fact choosing the following coding φ of letters from Σ_n in words of length n over a binary alphabet

$$\varphi(a_i) = 1^i 0^{n-i}; \quad \varphi(b_i) = 0^i 1^{n-i}, \quad i = 1, 2, \ldots, n$$

gives our co-NP-hardness result also in case of a constant-size alphabet. The proof of the result is rather technical, so we omit it here.

6 Open Problems

Theorem 3 shows that every n-state finitely generated synchronizing automaton has a synchronizing word of length at most $3n - 5$. On the other hand, as discussed in the introduction, there are examples of automata in **FG** having shortest synchronizing words of length $n-1$. We are interested in finding a series of finitely generated synchronizing automata whose shortest synchronizing word has length exactly $3n - 5$ or proving that this length is always not greater than $n - 1$.

Another interesting question concerns the precise complexity class of the problem `FINITENESS`. In particular, is it in co-NP?

Let $\mathscr{A} \in$**FG**. Since the language $\mathscr{L}_{min}(\mathscr{A})$ is finite, one may ask how many elements it might have? Another problem is to give a bound for the length of the longest word in $\mathscr{L}_{min}(\mathscr{A})$.

Acknowledgement

The authors thank professor Mikhail V.Volkov for proposing the problem and for the precious suggestions. They deeply appreciate useful remarks by professor Juhani Karhumäki and are grateful to Pawel Gawrychowski for communicating them his PSPACE result. They also thank anonymous referees for careful reading of the paper and a number of helpful remarks.

References

1. Černý, J.: Poznámka k homogénnym eksperimentom s konečnými automatami. Mat.-Fyz. Čas. Slovensk. Akad. Vied. 14, 208–216 (1964) (in Slovak)
2. Mateescu, A., Salomaa, A.: Many-valued truth functions, Černý's conjecture and road coloring. EATCS Bull. 68, 134–150 (1999)
3. Sandberg, S.: Homing and synchronizing sequences. In: Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.) Model-Based Testing of Reactive Systems. LNCS, vol. 3472, pp. 5–33. Springer, Heidelberg (2005)
4. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer, Heidelberg (2008)
5. Gawrychowski, P.: Finiteness problem for the language of minimal synchronizing words is in pspace. Private Communication (2008)
6. Pin, J.E.: Utilisation de l'algèbre linéaire en théorie des automates. Actes du 1er Colloque AFCET-SMF de Mathématiques Appliquèes, AFCET, Tome II, 85–92 (1978) (in French)
7. Margolis, S.W., Pin, J.E., Volkov, M.V.: Words guaranteeing minimum image. Int. J. Foundations Comp. Sci. 15(2), 259–276 (2004)
8. Perles, M., Rabin, M.O., Shamir, E.: The theory of definite automata. IEEE Trans. Electr. Comp. 12(3), 233–243 (1963)
9. Rystsov, I.K.: Resetting words for decidable automata. Cybernetics and Systems analysis 30(6), 807–811 (1994)

Genetic Algorithm for Synchronization

Adam Roman

Institute of Computer Science
Jagiellonian University, Cracow, Poland
roman@ii.uj.edu.pl

Abstract. We present a novel approach to the synchronization problem. It is a well-known fact that a problem of finding minimal (or: the shortest) synchronizing word (MSW) for a given synchronizing automaton is NP-complete. In this paper we present the genetic algorithm which tries, for a given automaton, to find possibly short word that synchronizes it. We use a modified version of a classical simple genetic algorithm (SGA).

1 Introduction

Synchronizing automata have many practical applications. They are used in robotics (for designing so-called part orienters) [1], bioinformatics (the reset problem) [2], network theory [3], theory of codes [4] etc. On the other hand, synchronizing theory is nowadays a field of the very intensive research, motivated mainly by the famous and still open Černý Conjecture. It claims that $(n-1)^2$ is an upper bound for the length of minimal synchronizing word (MSW) in any n-state synchronizing automaton. This conjecture, stated in 1964 by Černý [5], is the most longstanding open problem in automata theory.

When trying to apply the synchronization theory in some practical issues, the main problem we have to deal with is to find the minimal (or as short as possible) synchronizing word in some automaton. This problem is known as NP-complete [6]. For automata with small number n of states the exponential algorithm works well, but when n grows, it becomes useless. Instead of it we can use one of the well-known polynomial algorithms, such like Eppstein algorithm, cycle algorithm or others [6,7]. Each of them works good for some class of automata, but in general they rarely find the optimal solutions.

In this paper we present a genetic algorithm for solving the problem. According to our best knowledge, this is the first attempt to apply an evolutionary approach to the synchronization problem. Our approach is based on the model of simple genetic algorithm (SGA). The research on the abilities and limitations of the SGA (that is, what kinds of problems it can and cannot solve) has been heavily influenced by a theory of adaptation called the building block hypothesis [8,9,10]. It has become an object of a strong critique. The main reproach was that it lacks theoretical justification. Moreover some experimental results have been published that draw its veracity into question. The researchers tried to find some other models of evolutionary computations that would work better

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 684–695, 2009.

than SGA. New paradigms of evolutionary computations have arisen: evolutionary programming (with arbitrary chromosome structure), genetic programming, evolutionary strategies. In this paper we try to support somehow the idea lying behind the theory of SGA. The algorithm presented in Section 3 gives a promising result, mainly because the synchronizing problem nature fits well into SGA paradigm.

In SGA chromosome population is, in fact, the population of solutions to the encoded version of the original problem. When the best solution is found, it must be decoded into the language of the original problem. Usually the encoding and decoding functions are not identities. In the synchronizing problem an automaton is given and we want to find the shortest word which synchronizes it. Notice that in case of binary alphabet the word itself *is* the solution and it represents the chromosome directly - no encoding/decoding function is needed. In other words, genotype equals phenotype here. Moreover, it seems that the building block hypothesis works in this case, because the algorithm converges to local optima.

The paper is organized as follows: in Section 2 we introduce the basic notions referring to the synchronization theory and we formulate the synchronization problem. In Section 3 we present the genetic algorithm based on SGA paradigm, but with some modifications (mainly related to parameter adaptation issues). All genetic operations and heuristics used in GA are described there. Section 4 is devoted to the experiments and their results. Finally, in Section 5 we summarize our results and describe the future work in the field of evolutionary approach to synchronization problem.

2 Synchronizing Automata

Let A be a finite set. By A^* we denote the free monoid over A. If we interpret A as an alphabet, then A^* is the set of all finite words over A. For example, if $A = \{a, b\}$, then $A^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, ...\}$, where ε is an empty word. The length of $w \in A^*$ is the number of its letters. We denote the length of w by $|w|$ and we define $|\varepsilon| = 0$. By $|w|_a$ we denote the number of occurrences of a in w.

An *automaton* is a triple $\mathcal{A} = (Q, A, \delta)$, where Q is a finite set of *states*, A is a finite *alphabet* and $\delta : Q \times A \to Q$ is a *transition function*. It can be naturally extended on $2^Q \times A^* \to 2^Q$ in the following way: $\delta(P, \varepsilon) = P$, $\delta(P, aw) = \bigcup_{p \in P} \delta(\delta(p, a), w)$, where $p, q \in Q$, $P \subseteq Q$, $w \in A^*$. Since now we assume that we deal only with full, deterministic and strongly connected automata, that is: (1) δ is a total function on $Q \times A$, (2) $\forall p, q \in Q\ \exists w \in A^* :\ \delta(p, w) = q$. For the sake of simplicity, we will write $p.a = q$ instead of $\delta(p, a) = q$.

Definition 1. *Let $\mathcal{A} = (Q, A, \delta)$ be an automaton. If there exists $w \in A^*$ such that $|Q.w| = 1$, we call w a synchronizing word for $\mathcal{A}$. We also say that $\mathcal{A}$ is synchronizing.*

For a given $\mathcal{A} = (Q, A, \delta)$ and $w \in A^*$ we define the *deficiency* of w as follows: $\mathrm{df}(w) = |Q| - |Q.w|$. If w synchronizes $\mathcal{A}$, then $\mathrm{df}(w) = |Q| - 1$.

It is clear that if w synchronizes $\mathcal{A}$, then each word of the form uwv $(u, v \in A^*)$ also does it. Therefore a natural question arises: what is the shortest synchronizing word for $\mathcal{A}$? Such a word will be called a *minimal synchronizing word* (MSW) for $\mathcal{A}$. By $m(\mathcal{A})$ we denote the length of the MSW for $\mathcal{A}$.

Conjecture 1 (Černy, 1964). Let $\mathcal{A}$ be an n-state synchronizing automaton. Then

$$m(\mathcal{A}) \leq (n-1)^2.$$

For many classes of automata the conjecture was shown to be true (see for example [1,11,12,13,14]), but in general it is still an open problem. Černý showed that for each n there exists an n-state automaton with MSW of length $(n-1)^2$. These are so-called Černý automata and are denoted by $\mathcal{C}_n$.

Now we can formally define the synchronizing problem:

Definition 2 (Synchronizing problem). *Given* $\mathcal{A} = (Q, A, \delta)$, *find*

$$w^\circ = \underset{w:\ |Q.w|=1}{\operatorname{argmin}} \{|w|\}.$$

3 Genetic Algorithm

SGA transforms population of solutions in a way that imitates nature. The algorithm is described in Listing 1. We assume the reader to be familiar with basic notions and properties of SGA and we do not describe SGA here (see [8] for details). Throughout the paper the following notions will be used: N denotes the population size, t denotes the discrete time - in each time unit one population is processed $(t = 0, 1, 2, ...)$. P^t is a population in time t. By f we denote the fitness function and $d_i^t \in A^*$ is the i-th chromosome in P^t $(i = 1, 2, ..., N)$.

Algorithm 1. Simple Genetic Algorithm

```
1: begin
2:     t ← 0
3:     initiate population P^0
4:     evaluate P^0
5:     while (not stop criterium) do
6:     begin
7:         T^t ← selection in P^t
8:         O^t ← crossover and mutation in T^t
9:         evaluate O^t
10:        P^{t+1} ← O^t
11:        t ← t + 1
12:    end
13: end
```

Let us now describe some important differences between SGA and our algorithm.

1. In synchronization problem each solution is a word over some alphabet A. As we mentioned earlier, we will understand that words *are* chromosomes and their letters represent genes. But these words can have different lengths. Therefore we assume that the chromosome length is not fixed. This will have some important consequences when defining crossover and mutation;
2. We use adaptation methods for changing mutation and crossover probability during the algorithm work. Adaptation is also used for modifying the probability distribution $\mathcal{P}(A)$ over an alphabet. This distribution is utilized in the initiation phase and also in the mutation;
3. We use elitist selection - two copies of the best chromosome from P^{t-1} pass into P^t and the remaining $N-2$ are chosen using proportional selection based on the roulette wheel method. Here by "best" we understand the shortest word among all words in P^{t-1} with the highest deficiency.

In the next subsections we describe all details of our algorithm. All experiments will be performed for automata over binary alphabet, but in the theoretical parts of this paper we assume that A is arbitrary (that is, $|A| \geq 2$).

Initial Probability Distribution for Letters. The initial population P^0 is constructed using the heuristic on the distribution $\mathcal{P}(A)$. Let $\mathcal{A} = (Q, A, \delta)$ be an automaton and let $C_a = \{P \subset Q : \forall p, q \in P \ \exists i \geq 1 \ p.a^i = q\} = \{P_1, P_2, ..., P_{|C_a|}\}$. C_a is the set of all cycles in directed graph $G_a = (Q, E)$, where $(p, q) \in E \Leftrightarrow p.a = q$. We define $\mu(a) = \mathrm{LCM}(|P_1|, |P_2|, ..., |P_{|C_a|}|)$ as the lowest common multiple of the lengths of all cycles over a in $\mathcal{A}$. We also define $\xi(a) = \min\{k : \ \forall l > k \ Q.a^l = Q.a^k\}$. In other words, $\xi(a)$ is the longest path (over $a \in A$) leading from some state $p \in Q \setminus \bigcup C_a$ to some state $q \in \bigcup C_a$, assuming that no state appears twice or more on this path. If $\mu(a)$ is small, then there are more states lying on the paths leading to cycles, so they can be easily synchronize by a^k for some k into $Q.a^k$, such that $|Q.a^k|$ equals the number of cycles. When $\mu(a)$ is large, then more states are lying on cycles and usually it is harder to synchronize these states than in a previous case.

Now we are ready to define the probability distribution for alphabet letters:

$$\forall a \in A \ \mathcal{P}(a) = \frac{\xi(a) + \mu(a)}{\sum_{l \in A} \xi(l) + \mu(l)}. \tag{1}$$

Distribution (1) will be used for generating genes in initial population, as well as in the mutation operator.

Chromosome Length in P^0. The lengths of the words generated in the initial population are random according to a uniform distribution on $\{1, 2, ..., M\}$, where M is a fixed maximal length of chromosome. From [15] and [16,17] we know that for each n there exists an automaton $\mathcal{C}_n$ such that $m(\mathcal{C}_n) = (n-1)^2$ and that for each synchronizing $\mathcal{A}$ it is true that $m(\mathcal{A}) \leq \frac{n^3-n}{6}$, so necessarily $(n-1)^2 \leq M \leq \frac{n^3-n}{6}$. From the other hand, "Higgins [18] has shown that if Q is a set with n elements (where n is sufficiently large), then on average any product of $2n$ randomly chosen transformations of Q is a constant map. Being retold in

automata-theoretic terms, it implies that a randomly chosen automaton having n states and over a sufficiently large input alphabet tends to be synchronized, and, moreover, it is synchronized by any word of length $2n$" [19]. Numerical computations confirm that the mean length of MSW tends to behave linear (or even logarithmic) in n (see Fig. 1).

We strongly believe that Černý conjecture is true at least for binary alphabet, therefore in all experiments M has been usually defined as a value a little greater than n^2. According to the results described above, a more theoretical study is needed on the probability distribution of MSWs lengths for a given n. We could, for example, use a normal distribution $\mathcal{N}(m, \sigma)$ with $m = \Theta(n)$, but we don't know which σ would be the best choice. Nevertheless, we have decided to use a uniform distribution on $\{1, 2, ..., M\}$, because the algorithm works well even if the words in the initial population are much more longer than $m(\mathcal{A})$. The use of a normal (or some other) distribution could increase the algorithm convergence, but, as we said, little is known about the parameters of such a distribution.

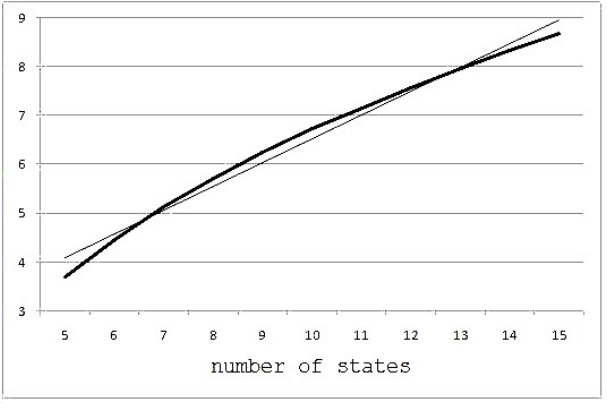


Fig. 1. Mean MSW length for a random n-state synchronizing automaton. Sample size $= 1000$ for each $n = 5, 6, ..., 15$. Linear regression: $y = 0,486x + 1,654$, $R^2 = 0,985$.

Probability Distribution Adaptation. When the algorithm works, the probability distribution for letters is modified. Let P^t be the currently processed population and let $\mathcal{P}$ be the probability distribution for letters in step $t-1$. If

$$\frac{\sum_{i=1}^{N} \mathrm{df}(d_i^t)}{N} > \frac{\sum_{i=1}^{N} \mathrm{df}(d_i^{t-1})}{N},$$

then we compute the empirical distribution for letters from P^t by the formula

$$\mathcal{P}_{emp}(a) = \frac{\sum_{i=1}^{N} |d_i^t|_a}{\sum_{i=1}^{N} |d_i^t|} \tag{2}$$

and we put $\mathcal{P} \leftarrow \mathcal{P}_{emp}$. The reason for implementing such an adaptation is simple: the optimal distribution should be equal to the distribution $\mathcal{P}_w$ in the MSW w, but this word is of course unknown. If the average deficiency in P^t increases in relation to the average deficiency in P^{t-1}, it means that P^t is "on average" better than P^{t-1} and it's empirical distribution $\mathcal{P}^t_{emp}$ is more similar to $\mathcal{P}_w$ than the current one. Therefore we make $\mathcal{P}^t_{emp}$ to be our new distribution $\mathcal{P}$.

Genetic Operators. Both mutation and crossover probabilities are defined by two values: p_m and p'_m for mutation and p_c and p'_c for crossover. At the beginning $p_m = p'_m$, $p_c = p'_c$ and p'_m, p'_c remain unchanged during the algorithm work. We use an adaptive technique for p_m and p_c values: if, in step t, $\max_{p\in P^t} f(p) \leq \max_{p\in P^{t-1}} f(p)$, we increase p_m by 0.001 and p_c by 0.01. In other case we put $p_m = p'_m$ and $p_c = p'_c$. Experiments showed that this simple adaptation mechanism speeds up the algorithm convergence.

Crossover. We use a modified version of one-point crossover, taking under consideration the variability of the chromosome length. Given two chromosomes $d^t_i = a_1a_2...a_s$ and $d^t_j = b_1b_2...b_t$ we pick two random numbers q, r, such that $1 \leq q \leq s$ and $1 \leq r \leq t$. The crossover results with two new offsprings $a_1a_2...a_qb_{r+1}b_{r+2}...b_t$ and $b_1b_2...b_ra_{q+1}a_{q+2}...a_s$, provided that $q + (t - r) \leq M$ and $r + (s - q) \leq M$, where M is the maximal allowed chromosome length.

Mutation. There are three types of mutation:

M1 for each letter in a given word flip a coin. If it's head, substitute the current letter a with another one. If $|A| > 2$ the new letter should be chosen according to the uniform distribution on $A \setminus \{a\}$;
M2 insert a random subword in a random place of the given word; letters for the new subword are generated with the current probability distribution $\mathcal{P}$;
M3 delete a random subword from a given word.

Mutation is realized as follows: Let $u \in [0, 1]$ be a random number. If $u \leq p_m$ flip a coin. If head, perform (M2) else perform (M3). At the end perform (M1) with probability p'_m for each letter. If mutation results with an empty word ε, replace it by a one-letter word. The letter is chosen randomly, according to the current probability distribution $\mathcal{P}$.

Fitness Function. In order to evaluate the chromosome, we have to take into consideration two values: $|w|$ and $\mathrm{df}(w)$. The shorter word is and the greater deficiency it has, the better it is. Moreover, we want f to have the following two properties: (P1) if $|Q.w| < |Q.v|$, then $f(w) > f(v)$; (P2) if $|Q.w| = |Q.v|$ and $|w| < |v|$, then $f(w) > f(v)$. After several series of experiments we have chosen the following fitness function:

$$f(w) = \frac{\mathrm{df}^4(w)}{\sqrt[4]{|w|}}. \tag{3}$$

Let w, v be two words such that $\mathrm{df}(w) > \mathrm{df}(v)$. If $|w| \leq |v|$, then clearly (P1) and (P2) holds. If $|w| > |v|$, (P1) and (P2) are almost always fulfilled,

because in practice, especially when n is large, $\frac{|w|}{|v|}$ is usually close to 1 and then $(1+\frac{1}{\mathrm{df}(w)-1})^4 > \sqrt[4]{\frac{|w|}{|v|}}$.

Algorithm. In this section we give the formal algorithm description. It is shown in Listing 2. $\mathcal{P}$ denotes the current probability distribution for letters.

Algorithm 2. SynchroGA

```
1: begin
2:    INPUT: A = (Q, A, δ)
3:    popSize← 40, p_m ← 0.03, p'_m ← p_m, p_c ← 0.7, p'_c ← p_c
4:    compute P
5:    t ← 0
6:    initialize P_0 using P
7:    evaluate P_0
8:    while (t < mboxpopNumber) do
9:    begin
10:      T_t ← selection P_t //elitist + roulette wheel
11:      O_t ← crossover and mutation (with P) on T_t // use p'_m and p'_c
12:      evaluate O_t
13:      if max_{p∈P^t} f(p) ≤ max_{p∈P^{t-1}} f(p)
14:         p'_m ← p_m + 0.001, p_c ← p_c + 0.01
15:      else p'_m ← p_m, p'c ← p_c
16:      if (1/N) Σ_{p∈P^t} df(p) > (1/N) Σ_{p∈P^{t-1}} df(p)
17:         compute P_emp in P^t
18:         P ← P_emp
19:      P_{t+1} ← O_t
20:      t ← t + 1
21:   end
22: end
```

4 Experiments and Results

In order to evaluate the quality of the SynchroGA algorithm three experiments were performed.

Experiment 1. SynchroGA was launched for several automata, one hundred times for each of them. In each run $i = 1, 2, ..., 100$ algorithm processed 2000 generations and returned $c_i \in P^{1999}$, which is the shortest among all words in P^{1999} with the highest deficiency. The results of this experiment are gathered in Table 1. Two first columns represent an automaton and the length of its MSW. Let $m = \max_i\{\mathrm{df}(c_i)\}$ and $W = \{c_i : \mathrm{df}(c_i) = m\}$. Columns 3-5 give some statistical information on the set W: the minimal, average and the maximal length of the words from W. Columns 6-8 give the same statistical information (but now for all words $c_1, c_2, ..., c_{100}$) on the value of $|Q.c_i|$. Column 9 shows a mean number of the first generation in which a word from W has been found. Column 10 describes the maximal allowed chromosome length.

Table 1. Results of the first experiment

$\mathcal{A}$	$m(\mathcal{A})$	Genetic algorithm							
		$\lvert MSW\rvert$			$\lvert Q.w\rvert$			MSW	max
		min	mean	max	min	mean	max	$\overline{N}$	ChrSize
$\mathcal{A}_{21}$	20	20	20.51	23	1	1	1	466.47	500
$\mathcal{A}_{41}$	40	40	41.85	46	1	1	1	643.86	2000
$\mathcal{A}_{61}$	60	60	62.6	70	1	1	1	778.23	4000
$\mathcal{B}_{21}$	20	20	20	20	1	1	1	248.76	500
$\mathcal{B}_{41}$	40	40	40.12	41	1	1	1	757.16	2000
$\mathcal{B}_{61}$	60	60	60.73	63	1	1	1	1123.97	4000
$\mathcal{C}_6$	25	25	28.42	33	1	1	1	1336.44	200
$\mathcal{C}_{11}$	100	117	139.4	168	1	1.06	2	1619.46	200
$\mathcal{C}_{16}$	225	262	262	262	1	2.02	3	1572	300
$\mathcal{C}_{21}$	400	282	391.9	493	2	2.59	4	1790.56	500
$\mathcal{C}_{41}$	1600	1562	1562	1562	3	6.09	9	1977	2000
$\mathcal{D}_7$	30	30	31.57	38	1	1.14	2	1254.81	40
$\mathcal{D}_{21}$	380	421	421	421	1	2	3	1682	500
$\mathcal{D}_{41}$	1560	1337	1548.57	1791	2	4.1	6	1757.43	2000

Automaton $\mathcal{A}_n = (\{1, 2, ..., n\}, \{a, b\}, \delta)$ has the following transition function: for $i < n$ $i.a = i + 1$; $n.a = n$; for all i $i.b = i$. It is clear that MSW length for $\mathcal{A}_n$ is a^{n-1} and $m(\mathcal{A}_n) = n - 1$. $\mathcal{B}_n = (\{1, 2, ..., n\}, \{a, b\}, \delta)$ is defined as follows: for even i, $i.a = i - 1$ and $i.b = i$; for odd $i \geq 3$, $i.a = i$ and $i.b = i - 1$; $1.a = 1.b = 1$. Minimal synchronizing word for $\mathcal{B}_n$ is $(ba)^{\frac{n}{2}}$ for n odd and $a(ba)^{\frac{n-1}{2}}$ for n even. Automata $\mathcal{A}_n$ and $\mathcal{B}_n$ are similar, but their initial probability distributions are different, so the experiment allows us to check if the probability adaptation mechanism works. $\mathcal{C}_n$ are n-state Černý automata and it is known that $m(\mathcal{C}_n) = (n - 1)^2$ [15]. MSW here is $(ba^{n-1})^{n-2}b$. Automaton $\mathcal{D}_n$ (n odd) with $Q = \{0, 1, ..., n - 1\}$ has the following transition function: $i.a = i - 2 (\text{mod } n)$ if $i = 0, 1$; $i.a = i$ if $2 \leq i \leq n - 1$; $i.b = i - 1 (\text{mod } n)$. It was proved [20] that $m(\mathcal{D}_n) = (n - 1)(n - 2)$ and it is realized by the word $w = (ab^{2k-1})^{k-1}(ab^{2k-2})^{k-1}a$, where $n = 2k + 1$.

Experiment 2. A thousand random 14-state synchronizing automata were generated. For each of them SynchroGA found a minimal synchronizing word. Fig. 2 shows the distribution of the number of the first generation t, in which the best word was found.

A typical run of the algorithm is shown in Fig. 3. Characteristic slow growths and immediate decreases of $|Q.w|$ are caused by the adaptive change of mutation and crossover probability. If, for a long period of time, no solution with higher deficiency is found, p_m and p_c grow. The algorithm then tends to behave more randomly, so the mean fitness decreases. When a new, better solution is found, p_m and p_c are set to their initial values and we can observe an immediate decrease of mean $|Q.w|$ and increase of the mean fitness.

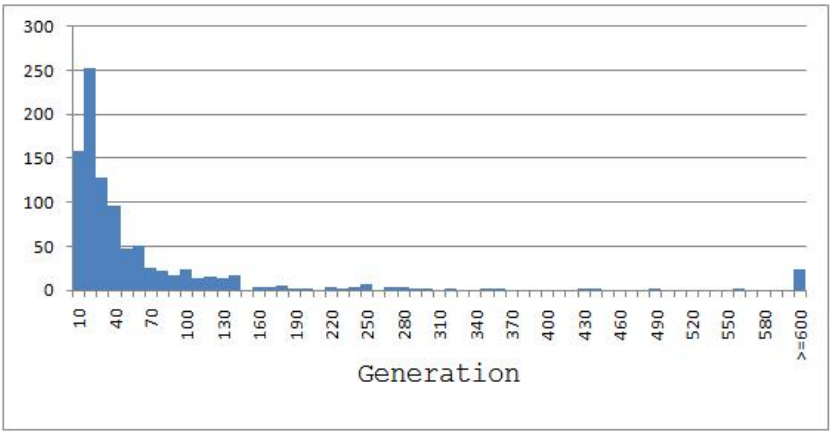


Fig. 2. For $n = 10, 20, ..., 600$ histogram shows the number of algorithm runs in which MSW was found in generation t, $n - 10 \leq t < n$

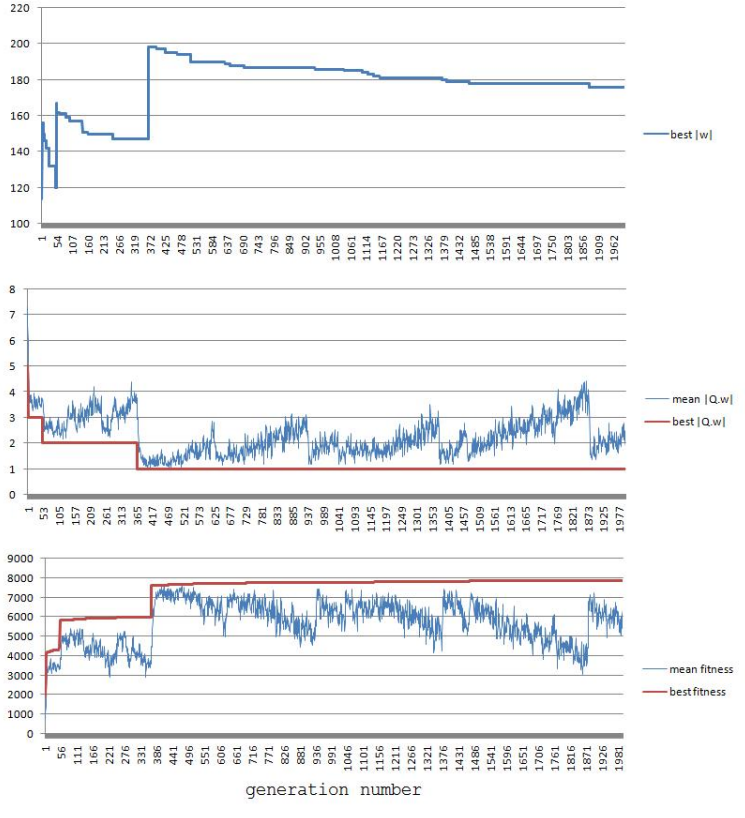


Fig. 3. A typical run of SynchroGA

Experiment 3. In the previous experiment computations were done for automata with small number of states. This is because we had to compare the result of SynchroGA with the *minimal* synchronizing word for a given automaton. This required the use of an exponential algorithm, so the number n of states had to be small. In this experiment we operate on bigger automata. In this case computing MSW is not possible, so instead of MSW value we use a regression line from Fig. 1. For each $n = 15, 16, ..., 100$ a ten random n-state automata were generated. In Fig. 4 the mean value of the shortest synchronizing word found by SynchroGA is shown in comparison with the values obtained from the regression equation for $x = n$. We see that these values tend to behave as a linear function, but with a lower slope than the regression line $y = 0.486x+1.654$ from Fig. 1. Taking into account that a genetic algorithm usually does not find the best solution but rather a sub-optimal one, results shown in Fig. 4 reinforce our conviction that the mean MSW length for a random n-state synchronizing automaton is proportional to an, with a much smaller than 0.486.

Fig. 4. Mean value of the shortest synchronizing word found by SynchroGA ($n = 15, ..., 100$) and regression line $y = 0.486x + 1.654$ from Fig. 1

5 Conclusions and Future Work

Experiment 1 shows that SynchroGA works good, but for "harder" automata (that is, with MSWs lengths quadratic in n, such like MSWs for Černý automata), the convergence decreases when n grows. For random automata (experiment 2) our algorithm deals with the synchronization problem very well and converges quickly. This can be explained by the fact that the expected value of MSW in a random automaton is (probably) linear or even logarithmic in n, so these words are rather short: results for automata $\mathcal{A}_n$ and $\mathcal{B}_n$ (see Table 1) confirm that SynchroGA works very good if MSW is short. Therefore, if we

see that the algorithm has a problem with finding a word with a high deficiency, we can almost be sure that the automaton has a long MSW.

We can improve the mean length of the synchronizing words found by SynchroGA. When it finds the best solution w, we can check if its length can be shortened. Let $w = a_1a_2...a_n$. If there are $i < j$ such that $Q.w_1...w_i = Q.w_1...w_j$ it is clear that for $v = a_1...a_ia_{j+1}...a_n$ we have $Q.w = Q.v$, but $|v| < |w|$ (notice that even if $|Q.w| = 1$, it does not imply that v will be the *minimal* synchronizing word, but only that it can be shorter than w). The optimizing step can be done not only at the end but, for example, after each k generations for each chromosome in the population. This, however, would highly increase the run time.

Notice that in a few cases the algorithm failed to find a synchronizing word. In such situation we can combine the genetic algorithm with any of the heuristic algorithms for finding short synchronizing words (for example, an Eppstein one [6]) in a following way: when a word w is found by the genetic algorithm and $|Q.w| > 1$, then run Eppstein algorithm for $Q.w$. It will return a word v such that $|Q.wv| = 1$, so the result is a synchronizing word wv.

In conclusion, we think that SynchroGA can be a useful tool in finding short synchronizing words. Future work should concentrate on tuning up the algorithm: finding the optimal mutation and crossover probability, dealing with another selection methods (for example a tournament selection method) and so on. As it was said before, a research on the relation between the automaton structure and it's $m(\mathcal{A})$ value should be done. This could be useful for choosing better initial probability distribution $\mathcal{P}$.

Algorithm can be found at www.ii.uj.edu.pl/∼roman/publications.html.

References

1. Ananichev, D., Volkov, M.: Synchronizing monotonic automata. In: Ésik, Z., Fülöp, Z. (eds.) DLT 2003. LNCS, vol. 2710, pp. 111–121. Springer, Heidelberg (2003)
2. Benenson, Y., Adar, R., Paz-Elizur, T., Livneh, L., Shapiro, E.: DNA molecule provides a computing machine with both data and fuel. Proc. National Acad. Sci. USA 100, 2191–2196 (2003)
3. Kari, J.: Synchronization and stability of finite automata. JUCS 8(2), 270–277 (2002)
4. Jürgensen, H.: Synchronization. Information and Computation 206(9-10), 1033–1044 (2008)
5. Černý, J.: Poznámka k. homogénnym experimentom s konecnymi automatmi. Mat. fyz. cas SAV 14, 208–215 (1964)
6. Eppstein, D.: Reset sequences for monotonic automata. SIAM J. Comput. 19, 500–510 (1990)
7. Roman, A.: Synchronizing finite automata with short reset words. Appl. Math. Comp. (in press, 2008), doi:10.1016/j.amc.2008.06.019
8. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
9. Holland, J.H.: Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press Edition (1992)

10. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge (1998)
11. Dubuc, L.: Les automates circulaires et la conjecture de Černý. Inform. Theor. Appl. 32, 21–34 (1998)
12. Kari, J.: Synchronizing finite automata on eulerian digraphs. Theor. Comp. Sci. 295, 223–232 (2003)
13. Rystsov, I.: Reset words for commutative and solvable automata. Theor. Comp. Sci. 172, 273–279 (1997)
14. Trahtman, A.N.: The existence of synchronizing word and Černý conjecture for some finite automata. In: Second Haifa Workshop on Graph Theory, Combinatorics and Algorithms, Haifa (2002)
15. Černý, J., Pirická, A., Rosenauerova, B.: On directable automata. Kybernetica 7, 289–298 (1971)
16. Klyachko, A.A., Rystsov, I., Spivak, M.A.: An extremal combinatorial problem associated with the bound on the length of a synchronizing word in an automaton. Kybernetika 2, 16–20 (1987)
17. Pin, J.E.: On two combinatorial problems arising from automata theory. Annals of Discrete Mathematics 17, 535–548 (1983)
18. Higgins, P.M.: The range order of a product of i transformations from a finite full transformation semigroup. Semigroup Forum 37, 31–36 (1988)
19. Volkov, M.: Personal communication (2008)
20. Ananichev, D., Volkov, M., Zaks, Y.: Synchronizing automata with a letter of deficiency 2. In: H. Ibarra, O., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 433–442. Springer, Heidelberg (2006)

Constructing Infinite Words of Intermediate Arithmetical Complexity

Paul V. Salimov

Sobolev Institute of Mathematics, prosp. Koptyuga 4, 630090,
Novosibirsk, Russia
ch.cat.s.smile@gmail.com

Abstract. Arithmetical complexity of an infinite word, defined by Avgustinovich, Fon-Der-Flaass and Frid in 2000, is the number of words of length n which occur in its arithmetical subsequences. We present a construction of infinite words whose arithmetical complexity function grows faster than any polynomial, but slower than any exponential. Also we give a rough upper bound for the arithmetical complexity of the Sierpiński word.

1 Introduction

Let Σ be a finite alphabet, and $x = x_0x_1 \cdots \in \Sigma^{\mathbb{N}_0}$ be an infinite word on Σ with indices in $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. A *factor* or *subword* of x is a finite word that occurs as a block of successive letters in x, i.e., is equal to $x_kx_{k+1} \dots x_{k+n}$ for some k and n, and if $k = 0$ such a subword is called *prefix*. The notation F_x is used for the set of subwords of x. The *length* of a finite word u is a number of letters in it and is denoted by $|u|$.

A classical complexity measure of a sequence x on a finite alphabet is the *subword complexity* that is the function f_x counting the number of its distinct subwords of the given length. A survey on subword complexity can be found at [1].

We consider another complexity measure that is the function a_x which counts not only factors of a given word but all distinct words occurring in its arithmetical progressions. Namely, the *arithmetical complexity* function a_x of n is the number of distinct words of the form $x_kx_{k+d} \dots x_{k+(n-1)d}$ for arbitrary initial positions $k \geqslant 0$ and differences $d \geqslant 1$. The set of such words is called the *arithmetical closure* of F_x and is denoted by A_x.

The arithmetical complexity was introduced in 2000 by Avgustinovich, Fon-Der-Flaass and Frid [2] and has become one of the most well-explored modified complexity functions relative to the subword complexity.

Characterizing the set of possible arithmetical complexity functions is a challenging problem and only partial results in this direction are known. Most of them concern words of polynomial arithmetical complexity.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 696–701, 2009.

It was shown in [3] how to construct words of intermediate subword complexity. In this paper we obtain the similar result for the arithmetical complexity. Namely, we present a construction of infinite words on the binary alphabet of arithmetical complexity that satisfies $2^{c_1 n^c} \leqslant a_x(n) \leqslant c_2 n^4 2^{c_3 n^c}$ for some $c < 1$ and some c_1,c_2,c_3.

2 Definitions

Let $\Sigma = \Sigma_s = \{0, 1, \ldots, s-1\}$ be a finite alphabet. The set of all finite words on Σ is denoted by Σ^*, the set of all non-empty finite words is denoted by Σ^+ and the set of all words of length n is denoted by Σ^n.

A *morphism* $\varphi : \Sigma^* \to \Sigma^*$ is a map that obeys the identity $\varphi(xy) = \varphi(x)\varphi(y)$ for all words $x, y \in \Sigma^*$. A morphism is called *m-uniform* if all images of letters have the same length equal to m.

An m-uniform morphism $\varphi : \Sigma^* \to \Sigma^*$ with $m > 0$ generates a mapping that maps an infinite word $x = x_0 x_1 \ldots$ to the word $\varphi(x_0)\varphi(x_1)\ldots$. We denote this mapping by the same letter φ.

If a morphism φ is m-uniform with $m \geqslant 2$ and the word $\varphi(a)$ starts with a letter a for some $a \in \Sigma$, then there always exists an infinite word x satisfying $x = \varphi(x)$ and it can be obtained as a limit $\lim_{n\to\infty} \varphi^n(a)$. Such a word x is called a *fixed point* of the morphism φ.

Example 1. The well-known Sierpiński word $s = 0101110101111\ldots$ is a fixed point of the 3-uniform morphism φ_S on Σ_2 defined by $\varphi_S(0) = 010$ and $\varphi_S(1) = 111$.

An m-uniform morphism $\varphi : \Sigma_s^* \to \Sigma_s^*$ is *symmetric* if the word $\varphi(i)$ is a result of a symbol-by-symbol addition modulo s of $\varphi(0)$ with $ii\ldots i$ for each i. It was shown in [4] that non-periodic fixed points of a symmetric morphism are of exponential arithmetical complexity. In [5] uniformly recurrent words of linear arithmetical complexity were studied, and some of them are fixed points of morphisms too.

We will consider some subclass of m-uniform morphisms and will prove that the arithmetical complexity functions of their fixed points grow slower than any exponential. Then we will show how to obtain words of intermediate arithmetical complexity from fixed points of such morphisms.

A word x is called *universal* if $F_x = \Sigma^*$.

We will need some facts about mechanical words [6]. An *upper mechanical* word $s = s_0 s_1 \ldots$ is an infinite word that can be defined by $s_i = \lceil (i+1)\alpha + \rho \rceil - \lceil i\alpha + \rho \rceil$ for some real $\alpha \in [0, 1]$ and ρ, i.e., mechanical words are either periodic or Sturmian words. Let the function $\chi(n)$ be the number of distinct words which are factors of mechanical words. It was proved (see Theorem 2.2.36 in [6]) that $\chi(n) = 1 + \sum_{i=1}^{n} (n + 1 - i)\phi(i)$ where ϕ is Euler's totient function, so $\chi(n) \leqslant c_\chi n^3$.

3 The Construction

Let us consider any morphism φ satisfying the following conditions:

$$\begin{aligned} &\varphi \text{ is an } m\text{-uniform } \Sigma_2^* \to \Sigma_2^* \text{ morphism } ; \\ &\varphi(0) \text{ starts with } 0 \ ; \\ &\varphi(0) \text{ contains exactly } t \text{ occurrences of } 0, \text{ where } 1 < t < m \ ; \\ &\varphi(1) = 1^m \ . \end{aligned} \tag{1}$$

The upper bound for the arithmetical complexity function of its fixed point grows slower than any exponential. Namely, we will prove the following statement.

Theorem 1. *If an infinite word x is a fixed point of a morphism φ satisfying conditions (1), then its arithmetical complexity function obeys the inequality*

$$a_x(n) \leqslant 4m^2 n\chi(mn) 2^{t^2 n^{\log_m t}}.$$

The main idea of the proof is that words of the arithmetical closure of x do not differ "very much" from those of the set F_x, due to conditions (1). More precisely, we will show that every word of the arithmetical closure of x can be obtained from some subword of x of length mn by some character deletion of a special kind and replacing some symbols 0 with 1. And since the number of occurrences of 0 in subwords of x grows slowly, we get a non-exponential upper bound for the arithmetical complexity function.

Let us define μ_0 as the function counting occurrences of a symbol 0 in a finite word.

Lemma 1. *If v is a subword of a fixed point of a morphism φ satisfying conditions (1), then $\mu_0(v) \leqslant t^{\lceil \log_m |v| \rceil}$.*

Proof. Let $k = \lceil \log_m |v| \rceil$. The word x is a fixed point of φ, so, by the definition, $x = \varphi^k(x) = \varphi^k(x_0)\varphi^k(x_1)\ldots$ where $|\varphi^k(x_i)| = m^k \geqslant |v|$. Hence, v is a subword of some word of the form $\varphi^k(x_j)\varphi^k(x_{j+1})$. Conditions (1) in particular contain $\varphi^k(1) = 1^{m^k}$ thus the maximum of $\mu_0(v)$ is achieved when v is a subword of $\varphi^k(00)$. In this case $\mu_0(v) \leqslant \mu_0(\varphi^k(0))$. From conditions (1) we have $\mu_0(\varphi^k(0)) = t^k$. □

There is some linear upper bound for the subword complexity function of a fixed point of an m-uniform morphism.

Lemma 2. *The subword complexity of a fixed point x of an m-uniform morphism on the binary alphabet satisfies the inequality*

$$f_x(n) \leqslant 4mn \ .$$

This lemma is a particular case of Theorem 10.3.1 from the book [7].

Proof (of Theorem 1). Let us consider a word $u = x_k x_{k+d} \dots x_{k+(n-1)d}$ of the arithmetical closure of x with $m^c \leqslant d < m^{c+1}$. The word x is a fixed point of φ so, by the definition, $x = \varphi^c(x) = \varphi^c(x_0)\varphi^c(x_1)\dots$, where $|\varphi^c(x_i)| = m^c$ (here φ^0 is the identity map). Thus u is in the arithmetical closure of some word $w = \varphi^c(v)$ of length equal to $m^{c+1}n$ where the word v is a subword of length equal to mn of x. More precisely, $u = w_{k_0} w_{k_0+d} \dots w_{k_0+(n-1)d}$, where $k_0 = k$ modulo m^c.

Let us divide the set $\{0, 1, \dots, m^{c+1}n - 1\}$ into blocks $B_i = \{im^c, im^c + 1, \dots, im^c + m^c - 1\}$ where $0 \leqslant i < mn$.

Now we can define the characteristic word $\theta = \theta_0\theta_1 \dots \theta_{mn-1}$ of the progression $k_0 + jd$ as follows: $\theta_i = 1$ if and only if there exists some j such that $k_0 + jd \in B_i$, otherwise $\theta_i = 0$.

The word θ is a factor of some mechanical word. More precisely, if s is a word defined by $s_i = \lceil (i+1)m^c/d - k_0/d \rceil - \lceil im^c/d - k_0/d \rceil$, then θ is a prefix of s. Let us prove it. If $s_i = 1$, then there exists such j that $(i+1)m^c/d - k_0/d > j$ and $im^c/d - k_0/d \leqslant j$; in other words, $(i+1)m^c > k_0 + dj \geqslant im^c$.

So, we have mn blocks; each of them contains at most 1 integer belonging to the progression $k_0 + jd$. Let us consider a mapping M that maps each number i of a block to a symbol of w at a position within the block B_i. And let us define the word $u' = u'(M, \theta)$ for such a mapping M and θ by $u'_i = M(j)$ where j is a position of the i's occurrence of 1 in θ.

Obviously, if M is such that $M(i) = w_{k_0+jd}$ when $k_0 + jd \in B_i$ for some j, then $u'(M, \theta) = u$. Thus $a_x(n)$ is less then or equal to the product of a number of possible θs, which is not greater than $\chi(mn)$, and a number of possible Ms, which we will estimate next.

It follows from conditions (1) that B_i contains a position of 0 in w only if $v_i = 0$. Therefore, from Lemma 1 we have that at most $\mu_0(v) \leqslant t^{\lceil \log_m mn \rceil}$ blocks contain positions of 0 in w. Hence, the number of possible Ms for current v is not greater than $2^{t^{\lceil \log_m mn \rceil}}$ and the number of all possible Ms is not greater than $2^{t^{\lceil \log_m mn \rceil}} f_x(mn)$.

Thus, using Lemma 2 we have

$$a_x(n) \leqslant f_x(mn)\chi(mn)2^{t^{\lceil \log_m mn \rceil}} \leqslant 4m^2 n\chi(mn)2^{t^{1+\log_m mn}} =$$
$$= 4m^2 n\chi(mn)2^{t(mn)^{\log_m t}} = 4m^2 n\chi(mn)2^{t^2 n^{\log_m t}} . \qquad \square$$

Corollary 1. *The arithmetical complexity of the Sierpiński word satisfies the inequality*

$$a_{sp}(n) \leqslant 36n\chi(3n)2^{4n^{\log_3 2}} .$$

Suppose that in an infinite word x from Theorem 1 some 0 were replaced with 1. Obviously, this will not increase the number of possible concerned mappings M. Namely, the following statement is true.

Proposition 1. *If an infinite word x is obtained from a fixed point of a morphism φ satisfying conditions (1) by replacing* 0 *with* 1 *at some positions, then its arithmetical complexity function obeys the inequality*

$$a_x(n) \leqslant 4m^2 n\chi(mn)2^{t^2 n^{\log_m t}} \quad .$$

This proposition and Lemma 1 allow us to construct words of intermediate complexity. Indeed, fixed points of morphisms satisfying conditions (1) are recurrent, which mean that they have infinitely many occurrences of each prefix. And each prefix p contains $t^{\lfloor \log_m |p| \rfloor}$ symbols 0. So if we replace 0 with 1 on proper positions in x then we obtain a word with subword complexity not less than $2^{t^{\lfloor \log_m n \rfloor}}$.

We are to formalize the replacing method. Let x and y be infinite or finite words on Σ_2. We denote by $R_0(x, y)$ a word obtained from x by replacing the i's occurrence of 0 in x with a symbol y_i.

Example 2. Let the word $s = 0101110101\ldots$ be the Sierpiński word from Example 1 and $y = y_0 y_1 \ldots$ be an infinite word, then $R_0(s, y) = y_0 1 y_1 111 y_2 1 y_3 \ldots$.

Proposition 2. *If an infinite word x is a fixed point of a morphism φ satisfying conditions (1) and y is a universal word, then the arithmetical complexity function of the word $z = R_0(x, y)$ obeys the inequality*

$$a_z(n) \geqslant 2^{\frac{n^{\log_m t}}{t}} \quad .$$

Proof. Obviously, $a_z(n) \geqslant f_z(n)$. The word x is a fixed point of φ so, by the definition, $x = \varphi^k(x) = \varphi^k(x_0)\varphi^k(x_1)\ldots$ for each k, and $\mu_0(\varphi^k(x_i))$ is either t^k or 0. Hence, the set of subwords of the word z contains all words of the form $R_0(\varphi^k(0), y_{it^k} y_{it^k+1} \ldots y_{(i+1)t^k-1})$ for arbitrary i.

Each word $s \in \Sigma^{t^k}$ can be found at position jt^k for some j in the word y. This follows from the fact that y is universal and thus contains a subword $(s0)^{t^k-1}s$.

So we have that the set of subwords of the word $R_0(x, y)$ contains at least $2^{t^{\lfloor \log_m n \rfloor}}$ distinct words of length n. Hence

$$a_z(n) \geqslant 2^{t^{\lfloor \log_m n \rfloor}} \geqslant 2^{t^{\log_m n - 1}} = 2^{t^{-1} n^{\log_m t}} \quad . \qquad \square$$

The definition of $R_0(x, y)$ fits the conditions of Proposition 1. So the main result of this paper can be formulated as follows

Proposition 3. *For arbitrary $t, m \in \mathbb{N}$, where $1 < t < m$, an infinite word x on Σ_2 with arithmetical complexity satisfying*

$$2^{\frac{n^{\log_m t}}{t}} \leqslant a_x(n) \leqslant 4m^2 n\chi(mn)2^{t^2 n^{\log_m t}}$$

can be constructed.

Acknowledgements

The author is grateful to Anna Frid and Sergey Avgustinovich.

References

1. Ferenczi, S.: Complexity of sequences and dynamical systems. Discrete Math. 206(1), 145–154 (1999)
2. Avgustinovich, S.V., Fon-Der-Flaass, D.G., Frid, A.E.: Arithmetical complexity of infinite words. In: Ito, M., Imaoka, T. (eds.) Words, Languages and Combinatorics III, pp. 51–62. World Scientific Publishing, Singapore (2003)
3. Cassaigne, J.: Constructing infinite words of intermediate complexity. In: Ito, M., Toyama, M. (eds.) DLT 2002. LNCS, vol. 2450, pp. 173–184. Springer, Heidelberg (2003)
4. Frid, A.E.: Arithmetical complexity of symmetric D0L words. Theoret. Comput. Sci. 306(1), 535–542 (2003)
5. Frid, A.E.: Sequences of linear arithmetical complexity. Theoret. Comput. Sci. 339, 68–87 (2005)
6. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2002)
7. Allouche, J.P., Shallit, J.: Automatic Sequences. Theory, Applications, Generalizations. Cambridge University Press, Cambridge (2003)

From Gene Trees to Species Trees through a Supertree Approach

Celine Scornavacca[1,2,⋆], Vincent Berry[2], and Vincent Ranwez[1]

[1] Institut des Sciences de l'Evolution (ISEM, UMR 5554 CNRS), Université Montpellier II, Place E. Bataillon - CC 064 - 34095
[2] Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM,UMR 5506, CNRS), Université Montpellier II 161, rue Ada, 34392 Montpellier Cedex 5, France
{scornava,vberry}@lirmm.fr, Vincent.Ranwez@univ-montp2.fr

Abstract. Gene trees are leaf-labeled trees inferred from molecular sequences. Due to duplication events arising in genome evolution, gene trees usually have multiple copies of some labels, *i.e.* species. Inferring a species tree from a set of multi-labeled gene trees (MUL trees) is a well-known problem in computational biology. We propose a novel approach to tackle this problem, mainly to transform a collection of MUL trees into a collection of evolutionary trees, each containing single copies of labels. To that aim, we provide several algorithmic building stones and describe how they fit within a general species tree inference process. Most algorithms have a linear-time complexity, except for an FPT algorithm proposed for a problem that we show to be intractable.

1 Introduction

An evolutionary tree (or *phylogeny*), is a tree displaying the evolutionary history of a set of sequences or organisms. A *gene tree* is an evolutionary tree built by analyzing a gene family, *i.e.* homologous molecular sequences appearing in the genome of different organisms. Gene trees are primarily used to estimate *species trees*, *i.e.* trees displaying the evolutionary relationships among studied species. Unfortunately, most gene trees can significantly differ from the species tree for methodological or biological reasons, such as long branch attraction, lateral gene transfers, deep gene coalescence and, principally, gene duplications and losses [1]. For this reason, species trees are usually estimated from a large number of gene trees.

Inferring a species tree from gene trees is mostly done in a two-step approach. First, a micro-evolutionary model that takes into account events affecting individual sites is used to infer the gene trees. The species tree is then inferred on the basis of a macro-evolutionary model, *i.e.* minimizing the number of transfer, duplication and loss events [2,3,4,5,6]. To produce more biologically meaningful trees, unified models have been proposed in which the micro and

⋆ Corresponding author.

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 702–714, 2009.

macro-evolutionary dimensions are entangled [7,8,9]. However, it is difficult to determine how to incorporate events occurring on different spatial and temporal scales, as well as belonging to neutral and non-neutral processes, in a single model [9]. Lately, a hybrid approach has been proposed, where a first draft of a species tree is inferred with a micro-evolutionary model, the most uncertain parts of which are then corrected according to a macro-evolutionary model [9].

In this paper, we propose instead to take advantage of the very large number of gene trees present in recent phylogenomic projects to avoid entering into the detail of all possible macro-evolutionary scenarios (*e.g.* is a parsimony approach always justified? Should only the most parsimonious scenario be retained?). We propose to extract the non-ambiguous part of the topological information contained in the gene trees, *i.e.* that resulting from speciation events as opposed to duplication events, and then apply a traditional supertree method letting the weight of evidence decide in favor of one candidate species tree [10,11,12].

This approach is only possible when the number of gene trees is very large, and indeed this is now the case in projects such as the HOMOLENS database (`http://pbil.univ-lyon1.fr/databases/homolens.php`), storing several thousands of gene trees. In the release 04 of this database, 51% of gene families have paralogous sequences, *i.e.* sequences where duplications and losses have actually taken place. Currently, these gene families are discarded when inferring a supertree of the concerned species. Disentangling information derived from speciation events from that resulting from duplication events would thus provide more information for species tree inference.

Supertree methods combine source trees whose leaves are labeled with individual species into a larger species tree. The source trees are *single-labeled*, *i.e.* each species labels at most one leaf. Note that, by definition, the inferred supertree is also single-labeled. In contrast, gene trees are usually *multi-labeled*, *i.e.* a single species can label more than one leaf, since duplication events resulted in the presence of several copies of the genes in the species genomes. The task we therefore have to solve is to extract the largest amount of unambiguous topological information from the multi-labeled gene trees under the form of single-labeled trees. This paper presents a number of results in this direction, that all play a role in the general scheme that is fully described below. The rest of the paper details these results, though in a different order for the sake of dependancies between definitions.

First of all, we propose to separately preprocess MUL trees in order to remove their redundant parts with respect to speciation events. For this purpose, we extend the tree isomorphism algorithm of [13] making it applicable to MUL trees while preserving a linear running time (section 5). This algorithm is then applied to the pairs of subtrees hanging from duplication nodes in MUL trees. This preprocess lowers the number of duplication nodes in gene trees. We also give in passing a linear time algorithm to identify duplication nodes in MUL trees (section 4). For the gene trees that still have duplication nodes, we define a set $\mathcal{R}$ of triplets (binary rooted trees on three leaves [14,12]) containing the topological information of a MUL tree that can be thought of as being unambiguously

related to speciation events. We show that this set of triplets can be computed in $O(|\mathcal{R}|)$ time (section 2). When this set is compatible, the MUL tree contributes a coherent topological signal to build the species tree. In such a case, we can replace the MUL tree with a single-labeled tree representing its associated set of triplets by using the BUILD algorithm [14] (section 3). When a MUL tree is not auto-coherent, we propose to extract a maximum subtree that is both auto-coherent and free of duplication events. Surprisingly, this optimization problem can be solved in linear time (section 6). When extracting largest single-labeled subtrees from MUL trees it is possible to obtain an incompatible collection, when a compatible collection could have been obtained by choosing subtrees of MUL trees in a coordinated way. However, solving this problem is computationally harder, as we show by providing an NP-completeness proof (section 7).

2 Preliminaries

In this paper we focus on rooted binary multi-labeled (MUL) trees. Let M be a MUL tree and v a vertex of M. We denote by **s(v)** and **s'(v)** the two sons of v and by **sons(v)** the set {s(v),s'(v)}. We define by **subtree(v)** the subtree with v as root and by **L(v)** the multiset of labels of subtree(v). We denote by $L(M)$ the multiset $L(root(M))$.

Definition 1. *A node v of M is called an **observed duplication node** (odn) if the intersection of $L(s(v))$ and $L(s'(v))$ is not empty.*

Note that, for an odn v, $L(v)$ will always contain some label more than once. We denote by $\mathcal{D}(M)$ the set of odn. A label $l \in L(M)$ is a *repeated label* for M iff the label l occurs more than once in $L(M)$. We say that f is a *repeated leaf* for M iff $L(f)$ is a repeated label. For every three leaves we can have three different rooted tree binary shapes, called *triplets*. We denote by $AB|C$ the rooted tree that connects the pair of taxa (A, B) to C via the root.

We denote by $\mathcal{R}(M)$ the set of triplets of a (single/multi)labeled evolutionary tree M *i.e.* $\mathcal{R}(M) = \{ab|c$ *s.t.* there exist three leaf nodes $x, y, z \in M : L(x) = a$, $L(y) = b$, $L(z) = c$ and $lca(x, y) \neq (lca(x, z) = lca(y, z))\}$[1].

Definition 2. *Let M be a MUL tree. We define by $\mathcal{R}_{wd}(M)$ ($\mathcal{R}(M)$ without duplications) the set of triplets $ab|c$ of M* s.t. *there exist three leaf nodes $x, y, z \in M : L(x) = a$, $L(y) = b$, $L(z) = c$ and $lca(x, y) \notin \mathcal{D}(M)$, $lca(x, y, z) \notin \mathcal{D}(M)$, $lca(x, y) \neq (lca(x, z) = lca(y, z))$.*

For example, for the MUL tree in Fig. 1, $\mathcal{R}_{wd}(M)$={$ac|b$, $ac|d$,$ab|d$,$bc|d$,$ac|o$,$ab|o$, $ad|o$,$bc|o$,$cd|o$,$bd|o$,$ab|c$}. Hence, not all the triplets of $\mathcal{R}(M)$ are kept. This is due to the fact that, once a duplication event occurred in a gene's history, the two copies of the gene evolved independently. The history of each copy is influenced by the species history but, considering them simultaneously may produce information unrelated to the species evolution. Therefore, it is more appropriate to discard the triplets mixing the histories of distinct copies of a gene.

[1] $lca(x, y)$ denotes the least common ancestor of nodes x and y, i.e. the lowest node in the tree that has both x and y as descendants.

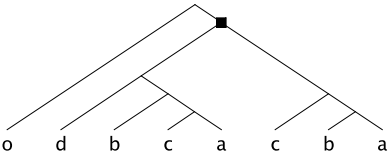


Fig. 1. An example of MUL tree with one odn indicated by a black square

$\mathcal{R}_{wd}(M)$ has an $O(n^3)$ size and can be computed in $O(n^3)$ time. Indeed, once the lca of all pairs of nodes in M are computed in $O(n)$ time (see [15,16]), checking for three leaf nodes x, y, z of M if they satisfy Definition 2 can be done in $O(1)$ time, thus in $O(n^3)$ for all triplets of leaves in M.

3 Auto-coherency of a MUL Tree

Let M be a multi-labeled gene tree. Since M contains several copies of the same gene, we can wonder whether the evolutionary signal of each copy is coherent or not.

Definition 3. *A MUL tree M is said to be* ***auto-coherent*** *if there exists a single-labeled tree T such that $\mathcal{R}_{wd}(M) \subseteq \mathcal{R}(T)$.*

In the case of an auto-coherent MUL tree, we know that we can find a tree T containing all the information in $\mathcal{R}_{wd}(M)$, *i.e.* the information of M that is considered reliable. To find such a tree, we use the AncestralBuild algorithm of [17]. For a set of triplets $\mathcal{R}_{wd}(M)$, this algorithm indicates in $O(|\mathcal{R}_{wd}(M)|\cdot\log^2 n)$ time whether there exists a tree T *s.t.* $\mathcal{R}_{wd}(M) \subseteq \mathcal{R}(T)$ and returns T in case of a positive answer. At present, it is not clear whether an implicit representation of $\mathcal{R}_{wd}(M)$ using only $O(n)$ triplets, as that reported in [18] to encode an binary tree, could be used here. Indeed, the presence of multiple occurrences of labels might lead to obtain a small compatible sets of triplets, while the whole set $\mathcal{R}_{wd}(M)$ would be incompatible.

4 Computing $\mathcal{D}(M)$ in Linear Time

The easiest way to compute $\mathcal{D}(M)$ is checking for each node v if the sets $L(s(v))$ and $L(s'(v))$ have at least one label in common; in the case of a positive answer, v is inserted in $\mathcal{D}(M)$. The complexity of this approach is $O(n^2)$, since it requires computing $O(n)$ intersections of two lists of $O(n)$ elements. The algorithm 1 uses the lca to find the set of odn $\mathcal{D}(M)$ and requires only linear time. To demonstrate the correctness of algorithm 1, we need to determine some relationships between the lca and the odn.

Lemma 1. *A node is an odn if and only if it is the lca of at least two repeated leaves m and p.*

Proof. Indeed, from the definition 1, v is an odn iff L(s(v)) $\cap$ L(s'(v)) $\neq \emptyset$. Therefore, $\exists m \in subtree(s(v))$ and $\exists p \in subtree(s'(v))$: $L(m) = L(p)$. Thus v is a common ancestor of the two leaves m and p with the same label. Now, m and p belong to two different subtrees having v as father ($m \in subtree(s(v))$ and $p \in subtree(s'(v))$), hence v is their lowest common ancestor in M. Reciprocally, if v is the lca of two leaves m and p with the same label, this means that L(s(v)) $\cap$ L(s'(v)) $\neq \emptyset$, then v is an odn according to definition 1. □

According to Lemma 1, we can search for the lca of any two leaves m and p with the same label. To determine the lca between multiple pairs of nodes, one can use an algorithm in [15] which preprocesses a data structure in O(n) time, where n is the number of nodes and returns the lca of any two specific nodes from the data structure in O(1). We still have O(n2) of these couples, and even constant time for each gives an O(n2) total complexity. However, since there are only O(n) odn, checking the lca of any pair of leaves computes the same lca several times. A smarter approach is used in algorithm 1: first of all, the subtrees of M are ordered from the left to the right in an arbitrary way. Then, each repeated leaf, starting from the left of the tree and moving to the right, is tagged with the repeated label followed by its occurrence number. Then, for each repeated label e, the lca of any two successive occurrences of e, e_i and e_{i+1} is inserted in $\mathcal{D}(M)$. This leads to a linear time complexity. Indeed, we have $O(n)$ of these couples since each leaf of M is involved in at most two pairs (e_i, e_{i+1}).

Algorithm 1. CompDuplicationNodes(r)

Data: A MUL tree M.
Result: A set of odn $\mathcal{D}(M)$.
Order M in an arbitrary way. In this order, tag each duplicated leaf with the repeated label followed by its occurrence number. Compute the lca for each couple of leaves.
$\mathcal{D}(M) \leftarrow \emptyset$;
foreach (*repeated label* e) **do**
  foreach ($\{e_j, e_{j+1}\}$) **do** $\mathcal{D}(M) \leftarrow lca(e_j, e_{j+1})$;
return $\mathcal{D}(M)$;

The correctness of algorithm 1 is justified by Lemma 2 showing that algorithm 1 retrieves all the odn of M.

Lemma 2. *Let M be a MUL tree. For each odn v, $\exists$ two successive occurrences of a label e denoted by e_i and e_{i+1} s.t. $v = lca(e_i, e_{i+1})$.*

Proof. Given an odn v, there exists at least one label e present in both subtrees $s(v)$ and $s'(v)$. We denote by A the set of leaves a_i *s.t.* $a_i \in subtree(s(v))$ and $L(a_i) = e$ and by B the set of leaves b_j *s.t.* $b_j \in subtree(s'(v))$ and $L(b_j) = e$. If we take the last element of B ($b_{|B|}$) and the first one of A (a_1), we know that v is their lca. Additionally, due to the way we tagged M, we know that there is no other occurrence of the label e between $b_{|B|}$ and a_1. Indeed, if there was another leaf x labeled with e, it would be either in $s(v)$ (and then $x = a_1$) or

in $s'(v)$ (and then $x = b_{|B|}$). Then $b_{|B|}$ and a_1 are two successive occurrences of the same label and their lca is the node v. □

5 Isomorphic Subtrees

Definition 4. *Two rooted trees T_1 , T_2 are isomorphic (denoted by T_1=T_2) iff there exists a one-to-one mapping from the nodes of T_1 onto the nodes of T_2 preserving leaf labels and descendancy.*

We are interested in testing if, for each odn v, the two subtrees $s(v)$ and $s'(v)$ are isomorphic or not. In the positive, we can prune one of the two isomorphic subtrees without losing any information contained in $\mathcal{R}_{wd}$ and eliminate the odn v, as in the example of Fig. 2. For detecting isomorphism of *multi-labelled* trees, we propose Algorithm 2, an extension of the CHECK-ISOMORPHISM-OR-FIND-CONFLICT algorithm [19]. Indeed, the latter does not deal with MUL trees. Alternatively, we could have proposed an appropriate variant of the tree isomorphism algorithm detailed in [20]. However, such an algorithm would likely have been less space efficient than the one we present here due to a number of string sorting steps using a number of queues and lists to ensure linear running time.

A node that has only two leaves as children is called **cherry**. In the case of single-labeled trees we have the following lemma:

Lemma 3. *[13] Let T_1, T_2 be two isomorphic trees and let c_1 be a cherry in T_1. Then, there is a cherry $c_2 \in T_2$ s.t. $L(c_1) = L(c_2)$.*

In the case of MUL trees, we can have several copies of the same cherry. We call a **multiple cherry** the list of cherries on the same two labels. For a multiple cherry mc, we note $|mc|$ the number of occurrences of the cherry in its tree.

Lemma 4. *Let M_1, M_2 be two isomorphic MUL trees and let mc_1 be a multiple cherry in M_1. Then, there is a multiple cherry $mc_2 \in M_2$ s.t. $L(mc_1) = L(mc_2)$ and $|mc_1| = |mc_2|$.*

The proof is inspired from that of Lemma 3 in [13] and left to the reader.

5.1 Outline of the Algorithm

First of all, we find all the multiple cherries for the MUL trees M_1 and M_2. We store them in the list L_{mc} using a simple linked list. Additionally, we use a

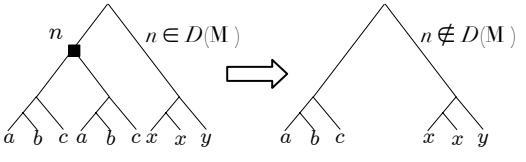


Fig. 2. An example of MUL tree where the sons of the duplication node are isomorphic

Algorithm 2. CheckIsomorphismMULTree(M_1,M_2)

Data: Two MUL tree M_1 and M_2.
Result: TRUE if M_1 and M_2 are isomorphic, FALSE otherwise.
Let L_{mc} be the list of multiple cherries in M_1 and M_2. Let H be the hashtable where each $mc \in L_{mc}$ is a key. To each $H(mc)$, we associates two lists $O_2(mc)$ and $O_1(mc)$, resp. of the occurrences of mc in M_1 and M_2;
while ($L_{mc} \neq \emptyset$) **do**
  $mc \leftarrow$ the first multiple cherry in L_{mc}; delete mc form L_{mc};
  if ($O_2(mc) = O_1(mc)$) **then**
    Turn all cherries in $O_2(mc)$ and $O_1(mc)$ into leaves to which a same new label is assigned;
    add the new multiple cherries in L_{mc} and H;
  else return FALSE;
return TRUE;

hashtable H where each $mc \in L_{mc}$ is a key. H associates to each multiple cherry mc two linked lists, $O_1(mc)$ and $O_2(mc)$, storing pointers to nodes of M_1 and M_2 respectively that correspond to the occurrences of mc. The multiple cherries of a MUL tree are then examined in a bottom-up process. Given a multiple cherry mc in L_{mc} we check if the size of $O_1(mc)$ is the same as that of $O_2(mc)$. If this is not the case, we have found a multiple cherry for which we do not have the same number of occurrences in M_1 and M_2. In this instance, M_1 and M_2 are not isomorphic (Lemma 4) and the algorithm returns FALSE. Otherwise we turn all the cherries in $O_1(mc)$ and $O_2(mc)$ into leaves to which a same new label, different from all other labels in M_1 and M_2, is assigned. This modification of M_1 and M_2 can turn the fathers of some cherries in $O_1(mc)$ and $O_2(mc)$ into new cherries. Then L_{mc} is updated and the processing of cherries in M_1 is iterated until both MUL trees are reduced to a single leaf with the same label if M_1 and M_2 are isomorphic, or a FALSE statement is returned.

Theorem 1. *Let M_1 and M_2 be two rooted MUL trees with $L(M_1) = L(M_2)$ of cardinality n. In time $O(n)$, algorithm 3 returns TRUE if M_1 and M_2 are isomorphic, FALSE otherwise.*

Proof. This algorithm is an extension of the CHECK-ISOMORPHISM-OR-FIND-CONFLICT algorithm [19] applicable to MUL trees. We show here that we can keep a linear time execution, using supplementary data structures.

A simple depth-first search of trees M_1 and M_2 initializes L_{mc} and H in $O(n)$ time. At each iteration of the algorithm, choosing a multiple cherry mc to process is done in $O(1)$ by removing the first element mc of L_{mc}. H then provides in $O(1)$ the lists $O_1(mc)$ and $O_2(mc)$ of its occurrences in the trees. Checking that these lists have the same number of elements is proportional to the number of nodes they contain, hence costs $O(n)$ amortized time, as each node is only once in such a list, and the list is processed once during the whole algorithm. Replacing all occurrences of mc by a new label is done in $O(n)$ amortized time, since each replacement is a local operation replacing

three nodes by one in a tree and at most $O(n)$ such replacements can take place in a tree to reduce it down to a single node (the algorithm stops when this situation is reached). Reducing a cherry can create a new occurrence $o_{mc'}$ of a cherry mc'. Checking in $O(1)$ time if mc' is a key in H allows to know whether occurrences of mc' have already been encountered or not. In the positive, we simply add $o_{mc'}$ to the beginning of the list $O_1(mc)$ (if $o_{mc'} \in M_1$) or $O_2(mc)$ (if $o_{mc'} \in M_2$), requiring $O(1)$ time. In the negative, we add mc' to the beginning of L_{mc}, create a new entry in H for mc', and initialize the associated lists $O_1(mc)$ and $O_2(mc)$ so that one contains $o_{mc'}$ and the other is the empty list. Again, this requires only $O(1)$ time. Thus, performing all operations required by the algorithm globally costs $O(n)$ time. □

Apply algorithm 2 to each $subtree(s(v))$ and $subtree(s'(v))$ of each odn of a MUL tree M in a bottom-up approach requires $O(dn)$ time, where d is the number of duplication nodes in M.

6 Computing a Largest Duplication-Free Subtree of a MUL Tree

If a MUL tree is not auto-coherent, identifying duplication nodes allows for the discrimination of leaves representing orthologous and paralogous sequences. Since only orthologous sequence history reflects the species history, a natural question is to determine the most informative sequence set for a given gene. As long as the gene tree contains odn, it will also contain leaves representing paralogous sequences. Yet, if for each node $v \in \mathcal{D}(M)$ of M we choose to keep either $s(v)$ or $s'(v)$, we obtain a pruned single-labeled tree containing only apparent orthologous sequences (observed paralogous have been removed by pruning nodes). Note that the so obtained single-labeled tree is auto-coherent by definition.

Definition 5. *Let M be a MUL tree. We say that T is obtained by (duplication)* pruning *M iff T is obtained from M choosing for each odn v either $s(v)$ or $s'(v)$. We denote this operation by the symbol $\lesssim$.*

One can wonder, for a non auto-coherent MUL tree M, what is the most informative single-labeled tree T *s.t.* $T \lesssim M$. We define this problem as the $MIPT$ (Most Informative Pruned Tree) problem.

To evaluate the informativeness of a tree we can use either the number of triplets of T (see [21,22,11]) that, for binary trees, depends only on the number of leaves, or the CIC criterion (see [23,12]). The CIC of a not fully resolved and incomplete[2] tree T with $|L(T)|$ leaves among the n possible is a function of both the number $n_R(T,n)$ of fully resolved trees T' on $L(T)$ such that $\mathcal{R}(T) \subseteq \mathcal{R}(T')$ and the number $n_R(n)$ of fully resolved trees on n leaves. More precisely,

$$CIC(T,n) = -\log\Big(n_R(T,n)/n_R(n)\Big)$$

[2] A tree is called incomplete when it misses some taxa.

Algorithm 3. pruning(v,M,$\mathcal{D}(M)$)

Data: A node v, a MUL tree M, and a set of odn $\mathcal{D}(M)$.
Result: The most informative MUL tree M' *s.t.* $subtree(v)_{M'} \lesssim subtree(v)_M$.
foreach $(m \in sons(v))$ **do** pruning(m,M,$\mathcal{D}(M)$);
if $(v \in \mathcal{D}(M))$ **then**
  if $(|L(s(v))| > |L(s'(v))|)$ **then** $v \leftarrow s(v)$;
  else $v \leftarrow s'(v)$;
  $\mathcal{D}(M) \leftarrow \mathcal{D}(M) - \{v\}$;
return M;

In the case of binary trees, $n_R(T, n)$ depends only on the number of source taxa missing in T since T does not contain multifurcations. Thus, dealing with binary trees, maximizing the information of a tree (*i.e.* maximizing the number of triplets or minimizing the CIC value) consists in finding the tree with the largest number of leaves. A natural approach for the MIPT problem for binary MUL trees is an algorithm that, after having computed $\mathcal{D}(M)$, uses the bottom-up algorithm 3, with v =root(M), to keep the most informative subtree between $subtree(s(m))$ and $subtree(s'(m))$, for each odn m.

Theorem 2. *Let M a MUL tree on a set of leaves of cardinality n. In time $O(n)$, pruning(M,root(M),$\mathcal{D}(M)$) returns the most informative tree T* s.t. $T \lesssim M$.

Proof. First of all, it's obvious that pruning(M,root(M),$\mathcal{D}(M)$) returns a tree. Indeed, if for each odn v only one node between $s(v)$ and $s'(v)$ is kept, at the end of the bottom-up procedure one copy of each duplicated leaf is present in the modified M. Now, we have to show that the resulting tree is the most informative tree *s.t.* $T \lesssim M$, *i.e.* the tree with as many leaves as possible. For an odn v that is the ancestor of other duplication nodes, the choices made for $s(v)$ do not influence the choices for $s'(v)$ since for each duplication node we can keep only one of the two subtrees, the most populous one. Thus we can search for the best set of choices left/right for $s(v)$ and $s'(v)$ independently and then choose the most populous pruned subtree between $s(v)$ and $s'(v)$. Iterating recursively this reasoning, we demonstrate that the tree obtained by Algorithm 3 is the most informative tree T *s.t.* $T \lesssim M$. The computation of the set of odn $\mathcal{D}(M)$ takes linear time. The subroutine $pruning(M, root(\mathrm{M}), \mathcal{D}(M))$ requires a tree walk, thus the time complexity of Algorithm 3 is $O(n)$. □

7 The Compatibility Issue of Single-labeled Subtrees Obtained from MUL Trees

We can also ask if it is possible, given a collection of MUL tree $\mathcal{M}$, to discriminate leaves representing orthologous and paralogous sequences in a gene tree using the information contained in the other gene trees to obtain a compatible forest $\mathcal{T}$, *i.e.* a forest for which there exists a tree T *s.t.* $\cup_{T_i \in \mathcal{T}} \mathcal{R}(T_i) \subseteq \mathcal{R}(T)$. We denote this problem by EPCF, Existence of a Pruned and Compatible Forest.

Unfortunately, the EPCF problem is NP-complete.

EPCF	**Instance** : A set of leaves X and a collection $\mathcal{M}=\{M_1, \cdots M_k\}$ of MUL trees on X. **Question** : $\exists$ a set S of choices left/right, $S : \mathcal{M} \rightarrow \mathcal{T}$, with $\mathcal{T}=\{T_1, \cdots T_k\}$ *s.t.* $T_i \lesssim M_i$ and $\mathcal{T}$ is compatible?

Theorem 3. *The EPCF problem is NP-complete.*

Proof. We start by proving that EPCF is in NP, *i.e.* checking if a set S of choices left/right is a solution for the instance $\mathcal{I} = (\mathcal{M}, X)$ can be done in polynomial time. First of all, for each MUL tree $M_j \in \mathcal{M}$, we place the choices left/right on M_j, *i.e.* we discard the subtrees not chosen, obtaining a forest of trees $\mathcal{T}$. We check then the compatibility of $\mathcal{T}$ with the Aho graph[14]. Constructing this graph can be done in polynomial time.

Given that EPCF is in NP, we use a reduction of 3-SAT to EPCF to demonstrate that it is NP-complete.

3-SAT	**Instance** : A boolean expression $\mathcal{C}=(C_1 \wedge C_2 \wedge \cdots \wedge C_n)$ on a finite set $L=\{l_1, l_2, \cdots, l_m\}$ of variables with $C_j=(a \vee b \vee c)$ where $\{a, b, c\} \in \{l_1, l_2, \cdots, l_m, \overline{l_1}, \overline{l_2} \cdots, \overline{l_m}\}$ **Question** : $\exists$ a truth assignment for L that satisfies all C_j in C ?

We need to show that every instance of 3-SAT can be transformed into an instance of EPCF; then we will show that given an instance $\mathcal{I} = (\mathcal{C}, L)$ of 3-SAT, $\mathcal{I}$ is a positive instance, *i.e.* an instance for which a solution exists, iff the corresponding instance for EPCF is positive.

Given an instance $\mathcal{I} = (\mathcal{C}, L)$ of 3-SAT, we build an instance $\mathcal{I}' = (\mathcal{M}, X)$ of EPCF associating to each l_i in L the binary tree[3] $T(l_i) = (((x_i, y_i), z_i), d)$ and to $\overline{l_i}$ the binary tree $T(\overline{l_i}) = (((z_i, y_i), x_i), d)$ (see Fig. 3 for an example).

The set of subtrees $\left\{T(a) \mid a \in \{l_1, l_2, \cdots, l_m, \overline{l_1}, \overline{l_2}, \cdots, \overline{l_m}\}\right\}$ is denoted by $\mathcal{T}_L$. Then, for each clause $C_j = (a \vee b \vee c)$ in C, a binary MUL tree M_j is built, formed by three subtrees $((T(a), T(b)), T(c))$. Note that M_j has exactly two duplication nodes due to the presence of d in $T(a)$, $T(b)$ and $T(c)$, so that any left/right choice of M_j will reduce it to either $T(a)$, $T(b)$ or $T(c)$. In Fig. 4 an example of a MUL tree built from a clause. In this way we obtain a forest of MUL trees $\mathcal{M}$ on the leaf set $X = \left\{\left\{\bigcup_{i=1}^{m}\{x_i, y_i, z_i\}\right\} \cup \{d\}\right\}$, *i.e.* an instance of the EPCF problem. Clearly $\mathcal{M}$ can be built in polynomial time.

We now need to show that a positive instance of 3-SAT gives a positive instance of EPCF through the previous transformation. Having a positive instance for 3-SAT implies that for each $C_j \in \mathcal{C}$ with $C_j = (a \vee b \vee c)$, at least one of the

[3] $T(l_i)$ is expressed in the Newick format.

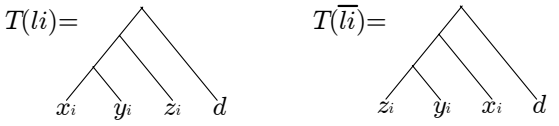


Fig. 3. Binary trees on four leaves associated to l_i and to $\overline{l_i}$

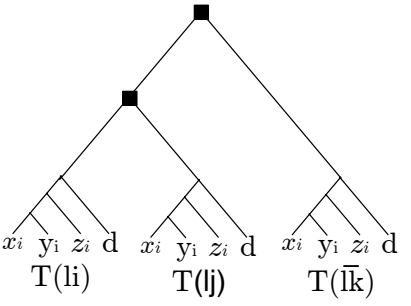


Fig. 4. MUL tree built from the clause $\{l_i \vee l_j \vee \overline{l_k}\}$. Odn are indicated by black squares

three literals is TRUE. Without loss of generality, let us suppose that a is TRUE. Then in the MUL tree M_j corresponding to C_j we set the choice left/right so that only the subtree $T(a)$ is kept. We then obtain a forest $\mathcal{T}$ that is a subset of $\mathcal{T}_L$. We need to prove that $\mathcal{T}$ is compatible. Let $\tilde{T}(a)$ denote the tree $T(a)|(L(T(a)) - \{d\})$[4] and $\tilde{\mathcal{T}}$ the forest composed by all trees $\{\tilde{T}(a)|T(a) \in \mathcal{T}\}$. Then, we can build a tree $T_s = (\tilde{T}_1, \tilde{T}_2, \cdots, \tilde{T}_{|\tilde{\mathcal{T}}|}, d)$. Since l_i cannot have the value TRUE and FALSE at the same time, we have either $T(l_i)$ or $T(\overline{l_i})$ in $\mathcal{T}$. The tree T_s is therefore a single-labeled tree. Moreover, by construction $T_s|(L(T(a))$ is identical to $T(a)$, for all $T(a)$ in $\mathcal{T}$ ensuring that $\bigcup_{T_i \in \mathcal{T}} \mathcal{R}(T_i) \subseteq \mathcal{R}(T_s)$. Thus $\mathcal{T}$ is compatible.

Now, the only thing left to prove is that a positive instance of EPCF gives a positive instance of 3-SAT.

The repetition of the taxon d in each subtree makes the two nodes connecting the subtrees in each M_j be odn. Thus a left/right choice set S reduces each M_j in $\mathcal{M}$ into a tree $T(a) \in \mathcal{T}_L$, providing the forest $\mathcal{T}$. Setting the value of a to TRUE ensures that the clause C_j corresponding to M_j is TRUE. This can be done simultaneously for all clauses $\in \mathcal{C}$ since the forest compatibility implies that there is no contradiction among the trees in $\mathcal{T}$, all the more so direct contradictions. Then, either $T(l_i)$ or $T(\overline{l_i})$ is in $\mathcal{T}$. This ensures us that either l_i or $\overline{l_i}$ is assigned to TRUE, but not both. □

Note that the problem to find the most informative forest $\mathcal{T} = \{T_1, \cdots T_k\}$ *s.t.* $T_i \lesssim M_i$ and $\mathcal{T}$ is compatible, denoted by MIPCF (Most Informative Pruned and Compatible Forest) is FPT. Indeed, analyzing all possible scenarios of choices left/right is FPT on the total number of duplication nodes in $\mathcal{M}$.

[4] Given a tree T and a label set S, we denote by $T|S$ the restriction of T to the set S.

8 Conclusions

In this paper we presented several algorithms to transform multi-labeled evolutionary trees into single-labeled ones so that they can be used by all existent supertree methods. In the future we plan to go further and extend the algorithm FIND-REFINEMENT-OR-CONFICT in [19] applicable to MUL trees and to work on a more sophisticated FPT algorithm for the MIPCF problem.

This work was funded by the ANR-08-EMER-011 project.

References

1. Cotton, J., Page, R.: Rates and patterns of gene duplication and loss in the human genome. Proceedings of Royal Soceity of London 272, 277–283 (2005)
2. Ma, B., Li, M., Zhang, L.: From gene trees to species trees. SIAM J. Comput. 30, 729–752 (2000)
3. Hallett, M.T., Lagergren, J.: New algorithms for the duplication-loss model. In: RECOMB 2000, Fourth Annual International Conference on Computational Molecular Biology, pp. 138–146 (2000)
4. Chen, K., Durand, D., Farach-Colton, M.: Notung: a program for dating gene duplications and optimizing gene family trees. J. Comput. Biol. 7, 429–444 (2000)
5. Vernot, B., Stolzer, M., Goldman, A., Durand, D.: Reconciliation with non-binary species trees. J. Comput. Biol. 15(8), 981–1006 (2008)
6. Chauve, C., Doyon, J.P., El-Mabrouk, N.: Gene family evolution by duplication, speciation, and loss. J. Comput. Biol. 15(8), 1043–1062 (2008)
7. Goodman, M., Czelusniak, J., Moore, G.W., Romero-Herrera, A.E., Matsuda, G.: Fitting the Gene Lineage into its Species Lineage, a Parsimony Strategy Illustrated by Cladograms Constructed from Globin Sequences. Systematic Zoology 28(2), 132–163 (1979)
8. Arvestad, L., Berglund, A., Lagergren, J., Sennblad, B.: Bayesian gene/species tree reconciliation and orthology analysis using MCMC. Bioinformatics 19(suppl. 1), 7–15 (2003)
9. Durand, D., Halld–rsson, B., Vernot, B.: A hybrid micro-macroevolutionary approach to gene tree reconstruction. J. Comput. Biol. 13, 320–335 (2006)
10. Baum, B.R., Ragan, M.A.: The MRP method. In: Bininda-Emonds, O. (ed.) Phylogenetic supertrees: combining information to reveal the Tree of Life, pp. 17–34. Kluwer, Dordrecht (2004)
11. Page, R.D.M.: Modified mincut supertrees. In: Guigó, R., Gusfield, D. (eds.) WABI 2002. LNCS, vol. 2452, pp. 537–552. Springer, Heidelberg (2002)
12. Scornavacca, C., Berry, V., Lefort, V., Douzery, E.J.P., Ranwez, V.: Physic_ist: cleaning source trees to infer more informative supertrees. BMC Bioinformatics 9(8), 413 (2008)
13. Gusfield, D.: Efficient algorithms for inferring evolutionary trees. Networks 21, 12–28 (1991)
14. Aho, A.V., Sagiv, Y., Szymanski, T.G., Ullman, J.D.: Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. SIAM J. Comp. 10(3), 405–421 (1981)
15. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. 13(2), 338–355 (1984)

16. Bender, M.A., Farach-Colton, M.: The lca problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
17. Berry, V., Semple, C.: Fast computation of supertrees for compatible phylogenies with nested taxa. Syst. Biol. 55, 270–288 (2006)
18. Henzinger, M., King, V., Warnow, T.: Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. Algorithmica 24, 1–13 (1999)
19. Berry, V., Nicolas, F.: Improved parameterized complexity of the maximum agreement subtree and maximum compatible tree problems. IEEE/ACM Trans. Comput. Biol. Bioinformatics 3(3), 289–302 (2006)
20. Aho, A., Hopcroft, J., Ullman, J.: The Design and Analysis of Computer Algorithms. Addison-Wesley Longman Publishing Co., Inc., Boston (1974)
21. Ranwez, V., Berry, V., Criscuolo, A., Fabre, P., Guillemot, S., Scornavacca, C., Douzery, E.: PhySIC: a veto supertree method with desirable properties. Syst. Biol. 56, 798–817 (2007)
22. Semple, C., Steel, M.: A supertree method for rooted trees. Discrete Appl. Math. 105, 147–158 (2000)
23. Thorley, J., Wilkinson, M., Charleston, M.: The information content of consensus trees. In: Rizzi, A., Vichi, M., Bock, H.H. (eds.) Advances in Data Science and Classification. Studies in Classification, Data Analysis, and Knowledge Organization, pp. 91–98 (1998)

A Kleene Theorem for Forest Languages

Lutz Straßburger

INRIA Saclay, Île-de-France, Équipe-projet Parsifal
École Polytechnique, LIX, Rue de Saclay, 91128 Palaiseau Cedex, France
http://www.lix.polytechnique.fr/~lutz

Abstract. This paper proposes an alternative approach to the standard notion of rational (or regular) expression for tree languages. The main difference is that in the new notion we have only one concatenation operation and only one star-operation, instead of many different ones. This is achieved by considering forests instead of trees over a ranked alphabet, or, algebraicly speaking, by considering cartesian categories instead of term-algebras. The main result is that in the free cartesian category the rational languages and the recognizable languages coincide. For the construction of the rational expression for a recognizable language it is not necessary to extend the alphabet. We only use operations that can be defined with the algebraic structure provided by cartesian categories.

1 Introduction

Kleene's theorem on the coincidence of the rational and the recognizable languages in a free monoid [1] is considered to be one of the cornerstones of theoretical computer science. Although it does not hold in every monoid [2], it has been generalized in several directions, e.g., [3,4,5]. Thatcher and Wright presented in [6] a similar result for tree languages. In recent years, recognizable tree languages have received increased attention because they form a foundation for XML schemas for expressing semi-structured data (e.g., [7,8,9]). In this paper I will only consider ranked trees and tuples of ranked trees, which I call *forests* to distinguish them from *hedges* [10] which are sequences of unranked trees [9].

The recognizable tree languages are defined by means of *finite-state tree automata* (*fsta*), which are a generalization of ordinary *finite state automata* (*fsa*) over words. While an fsa works on an alphabet A, an fsta works on a *ranked alphabet* Σ, i.e., every symbol in Σ is equipped with a rank, which is a natural number. The language recognized by an fsta is a subset of the set T_Σ of all finite trees over Σ. The running example for this paper is the ranked alphabet $\Sigma = \{\sigma, \gamma, \alpha\}$, where σ has rank 2, γ has rank 1, and α has rank 0. Let us define an fsta $\mathfrak{A}$ with two states p and f, where f is a final state. Observe that (contrary to fsa on words) there are no initial states. Usually the behaviour of an fsta is represented as set of rules of a term rewriting system [11]. In the example, let the behaviour of $\mathfrak{A}$ be described by the rules

$$\alpha \to p \quad , \qquad \alpha \to f \quad , \qquad \gamma(p) \to p \quad , \qquad \sigma(p, f) \to f \quad .$$

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 715–727, 2009.

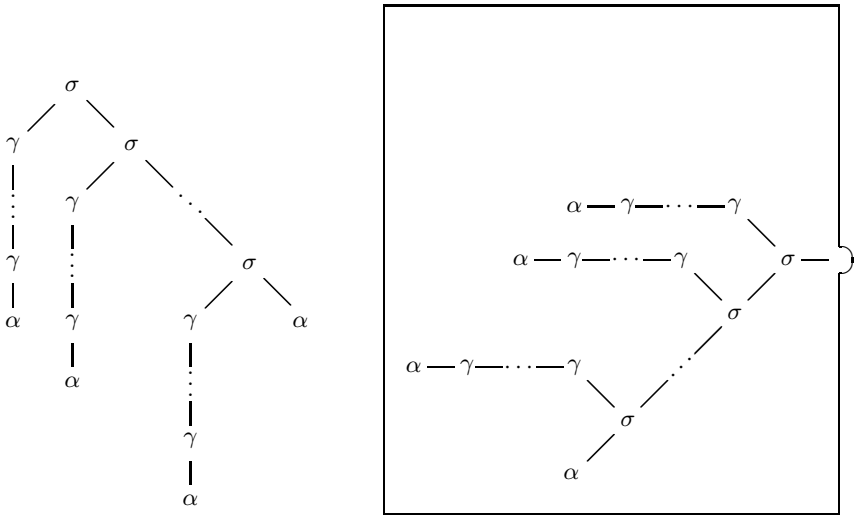


Fig. 1. Left: The form of the trees in language L. **Right:** Depicted as forests.

The tree language L accepted by this automaton contains the tree α and all trees of the shape $\sigma(\gamma \dots \gamma\alpha, \sigma(\gamma \dots \gamma\alpha, \sigma(\dots, \sigma(\gamma \dots \gamma\alpha, \alpha) \dots)))$, which is more vivid if drawn as a tree as on the left of Fig. 1, where the number of γ's in each chain and the number of σ's can be any natural number (including zero).

In order to find for this recognizable tree language a rational expression in the same sense as for recognizable word languages, we encounter the problem that there is no straightforward concatenation operation for trees. The concatenation of two words is simply their juxtaposition. But what is the concatenation of two trees? The approach taken in [6] is to paste one tree into some leaves of the other tree. In order to decide into which leaves of the first tree the second tree has to be plugged in, we have to name those leaves. In a tree over Σ, the leaves are labeled by the 0-ary symbols of Σ. Hence, for every 0-ary symbol of Σ, there is another concatenation operation. In order to obtain a rational expression for every recognizable tree language, it is necessary to extend Σ by additional 0-ary symbols. The rational expression for the language in our example is

$$L = \{\alpha\} \cup (\{\sigma(p, f)\} \cdot_p \{\gamma(p)\}^{*_p} \cdot_p \{\alpha\})^{*_f} \cdot_f \{\alpha\} \quad . \tag{1}$$

Observe that there are two additional symbols p and f with rank 0 that are not elements of Σ. This means that although the language L is a subset of T_Σ, the sets $\{\sigma(p, f)\}$ and $\{\sigma(p, f)\} \cdot_p \{\gamma(p)\}^{*_p}$, that occur in in (1), are not subsets of T_Σ but subsets of $T_{\Sigma \cup \{p,f\}}$. We see two different concatenation operations, $\cdot_p$ and $\cdot_f$, in (1), because there are two states in the corresponding automaton.

It has been shown in [6], that for every recognizable tree language there exists such a rational expression, and that every tree language that can be represented by a rational expression is indeed recognizable. There are three points that one might find disturbing:

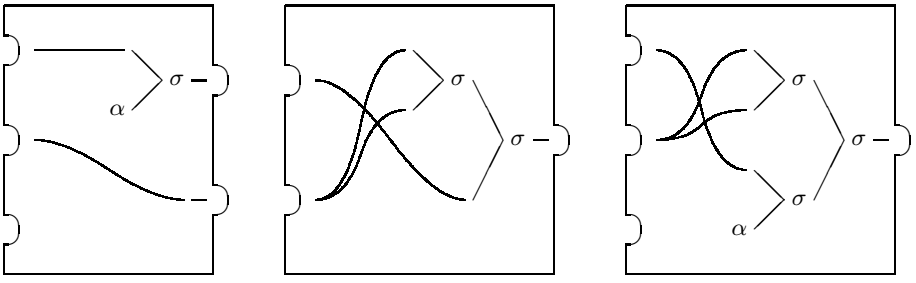


Fig. 2. The three forests s, t and r

(i) The language for formalizing the rational expressions is not predefined by Σ. For a given recognizable tree language $L \subseteq T_\Sigma$, the number of different concatenation operations that occur in the rational expression for L is determined by the automaton that recognizes L. (In rational expressions for word languages we have only one concatenation operation.)

(ii) The tree languages that are obtained by evaluating a subexpression of a rational expression for a language $L \subseteq T_\Sigma$ are usually not subsets of T_Σ. (In the case of word languages, a subexpression of a rational expression for a language $L \subseteq A^*$ always represents a subset of A^*.)

(iii) The set T_Σ of all trees over Σ forms the free Σ-algebra generated by the empty set. The tree concatenation operations have no correspondence in the Σ-algebra structure. (The set A^* of all words over A forms the free monoid generated by A, and the word concatenation operation is the multiplication in the monoid.) As a consequence, the notion of rational (tree) language cannot easily be generalized to any Σ-algebra, as it has been done very successfully for rational languages in any monoid [2,3,4,5].

These three problems can be entirely avoided by a remarkable simple generalization: instead of concatenating trees, we concatenate tuples of trees, i.e., forests. The set of all forests over a ranked alphabet Σ will be denoted by $\Sigma^{\circledast}$. Algebraicly speaking, we generalize the notion of monoid not to a Σ-algebra but to a cartesian category, and $\Sigma^{\circledast}$ is the free cartesian category generated by Σ. The usual free monoid of words can be seen as the special case in which all symbols in Σ have rank 1. One can visualize a forest as a box with holes on one side and plugs on the other. As example, Figure 2 shows three forests (for the same Σ as above). If we call them s, t and r, respectively, then s contains two trees and t and r contain each one tree. The leaves either contain a symbol of Σ with rank 0 or are connected to a hole of the box. The concatenation operation, which is the composition of arrows in the category, becomes now a plugging together of two forests with a matching number of plugs and holes. In our example, s and t can be plugged together and the result is r. With this approach, the many tree concatenation operations are reduced to a single forest concatenation operation.

Note that this category has been around since the work by Lawvere on algebraic theories [12]. It has also been extensively studied in the context of

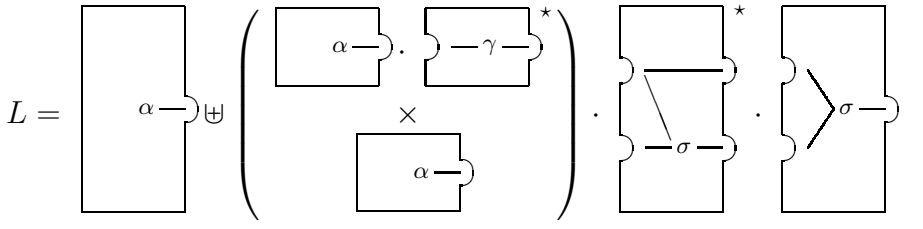


Fig. 3. The rational expression in for the language L

iteration theories [13], in which the regular forests have been characterized in various ways [14]. But the structure is different from preclones [15], which have another concatenation operation (see also Remark 2.2 in [15]). Our category is also a special case of a magmoid [16]. However, it seems that the structure of cartesian category has not yet been used to define the notion of *rational expression* in order to provide a Kleene-theorem (but see [17] for related work on unranked trees).

The notion of recognizability presented in this paper is the same as the one by Thatcher and Wright [6] or the one by Ésik and Weil [15] (modulo the cosmetic difference that we consider forests instead of trees). The novelty is the new notion of rational expression, whose definition is closer to the rational languages in a monoid [2]. The only operations that are allowed are the componentwise union $\uplus$, the cartesian product $\times$, the concatenation $\cdot$, and the concatenation closure *, which is the generalization of the Kleene star * for words. Although it is straightforward to translate the many concatenation and star-operations of [6] for trees into the corresponding forest operations presented in this paper, the converse translation is not so immediate. Nonetheless, it follows from [6] and our work, that the two notions coincide for tree languages. Thus, this paper provides yet another characterization of the recognizable (or regular) tree languages.

For giving an example, the right-hand side of Fig. 1 shows how the language L can be depicted as forest language. The new rational expression is depicted in Fig. 3. Observe that there is only one concatenation operation and that every subexpression represents a subset of $\Sigma^{\circledast}$. Note that the operations used in a rational expression are available in any cartesian category (a category with a terminal object and all finite products). This means that the notions of recognizability and rationality defined in this paper are available in all cartesian categories, where the two notions do in general not coincide.

Related to this work is the general notion of substitution (e.g., [18]). However, the set of substitutions forms a monoid and the composition of substitutions is the multiplication in that monoid. Imposing the notions of recognizability and rationality on substitutions would mean to use the well-known notions of recognizability and rationality in a monoid [2], whereas the purpose of this paper is to capture the notion of recognizability for trees. For this it is necessary to add to each substitution also the information about which variables occur in the domain and in the codomain; and this in turn leads to the notion of forest.

All the technical details of this paper are available in [19].

2 Forests

Forests are tuples of trees (or terms) over a ranked alphabet. In other words, they are the arrows in the free cartesian category generated by the alphabet, and the forest concatenation is the usual term substitution, which is the arrow composition in that category [12]. In order to make the paper self-contained, the definitions are given if full (without using category theory).

Definition 1. *A* ranked alphabet Σ *is a finite set of symbols together with a function rank* $: \Sigma \to \mathbb{N}$*, which assigns to every symbol* $\sigma \in \Sigma$ *its rank. For a given* $k \in \mathbb{N}$*, let* $\Sigma_k = \{\sigma \in \Sigma \mid rank(\sigma) = k\}$.

Example 1. For the example from the introduction we have $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$ with $\Sigma_0 = \{\alpha\}$, $\Sigma_1 = \{\gamma\}$ and $\Sigma_2 = \{\sigma\}$.

Definition 2. *Let* Σ *be a ranked alphabet. For every* $n, m \geq 0$*, the set* $\Sigma^{\circledast}[n, m]$ *of all* (n, m)-forests over Σ *is the smallest set such that*

(i) *the set* $\Sigma^{\circledast}[n, 0] = \{\langle\rangle_n\}$ *is a singleton,*
(ii) *for every* $i \in \{1, \ldots, n\}$ *(with* $n \geq 1$*), there is a forest* $\pi_i^n \in \Sigma^{\circledast}[n, 1]$*,*
(iii) *for every* $k \geq 0$ *and* $t \in \Sigma^{\circledast}[n, k]$ *and* $\sigma \in \Sigma_k$*, there is a forest* $t\sigma \in \Sigma^{\circledast}[n, 1]$*,*
(iv) *if* $m \geq 2$ *and* $t_1, \ldots, t_m \in \Sigma^{\circledast}[n, 1]$*, then* $\langle t_1, \ldots, t_m\rangle \in \Sigma^{\circledast}[n, m]$.

The elements of the set $\Sigma^{\circledast} = \bigcup_{n,m\in\mathbb{N}} \Sigma^{\circledast}[n, m]$ *are called* forests over Σ.

Example 2. Let Σ be as above. Then $s = \langle\langle\pi_1^3, \langle\rangle_3\alpha\rangle\sigma, \pi_2^3\rangle$ is a $(3, 2)$-forest and $t = \langle\langle\pi_2^2, \pi_2^2\rangle\sigma, \pi_1^2\rangle\sigma$ is a $(2, 1)$-forest. Both are depicted in Fig. 2.

Notation 1. By some abuse of notation, we can see Σ_k as subset of $\Sigma^{\circledast}[k, 1]$, by identifying $\sigma \in \Sigma_k$ with $\langle\pi_1^k, \ldots, \pi_k^k\rangle\sigma \in \Sigma^{\circledast}[k, 1]$.

Definition 3. *For a given ranked alphabet* Σ*, the* forest concatenation $;$ *is inductively defined as follows.*

(i) *For every* $n, m \in \mathbb{N}$ *and* $t \in \Sigma^{\circledast}[n, m]$*, let* $t; \langle\rangle_m = \langle\rangle_n$*,*
(ii) *for every* $n, m \in \mathbb{N}$ *(with* $m \geq 1$*), every* $t_1, \ldots, t_m \in \Sigma^{\circledast}[n, 1]$*, and every* $i \in \{1, \ldots, m\}$*, let* $\langle t_1, \ldots, t_m\rangle; \pi_i^m = t_i$*,*
(iii) *for every* $n, m, k \in \mathbb{N}$*, every* $s \in \Sigma^{\circledast}[n, m]$*, every* $t \in \Sigma^{\circledast}[m, k]$*, and every* $\sigma \in \Sigma_k$*, let* $s; (t\sigma) = (s; t)\sigma$*,*
(iv) *for every* $n, m, k \in \mathbb{N}$ *(with* $k \geq 2$*), every* $s \in \Sigma^{\circledast}[n, m]$*, and every* $t_1, \ldots, t_k \in \Sigma^{\circledast}[m, 1]$*, let* $s; \langle t_1, \ldots, t_k\rangle = \langle s; t_1, \ldots, s; t_k\rangle$.

Example 3. The concatenation of s and t above is $s; t = \langle\langle\pi_2^3, \pi_2^3\rangle\sigma, \langle\pi_1^3, \langle\rangle_3\alpha\rangle\sigma\rangle\sigma \in \Sigma^{\circledast}[3, 1]$, which is the forest r shown in Fig. 2.

One can easily show that the forest concatenation is associative and that for every $n, m \in \mathbb{N}$ and $s \in \Sigma^{\circledast}[n, m]$ we have that $\langle\pi_1^n, \ldots, \pi_n^n\rangle; s = s = s; \langle\pi_1^m, \ldots, \pi_m^m\rangle$. Hence, we have a category (that we also denote by $\Sigma^{\circledast}$) whose objects are the non-negative integers, the terminal object is 0, the cartesian product is given by the usual addition, and the π_i^n are the projections. I will write ε_n for $\langle\pi_1^n, \ldots, \pi_n^n\rangle \in \Sigma^{\circledast}[n, n]$ if $n \geq 1$ (resembling the empty word ε in the free monoid) and ε_0 for $\langle\rangle_0 \in \Sigma^{\circledast}[0, 0]$.

3 Recognizable Forest Languages

One can define recognizability either via automata or via the inverse image of a morphism into a finite structure. In our case this would be a cartesian category with countably many objects but with finite hom-sets. Usually there is a straightforward translation between such a morphism and a *deterministic* finite-state automaton. Our case is no different in that respect. But many costructions on automata, for example concatenation, require the non-deterministic model. Thus, we define the recognizable forest languages via *(non-deterministic) finite-state forest automata* (*fsfa*) which accept those languages. Morally, an fsfa is the same as a *finite-state tree automaton* (*fsta*), see e.g. [11], but technically, there is a subtle difference: fsfa have initial as well as final states, whereas fsta do only have final states. On the other hand, usual *finite-state automata* (*fsa*) over words do also have initial as well as final states. Thus, forest automata regain a symmetry which was lost for tree automata. In the definitions that follow, we will stay as close as possible to the theory of fsa for word languages [20]. The main difference is that there is no longer a single state transition relation (or state transition function), but a family of such relations (or functions), one for each rank that occurs in the ranked alphabet Σ.

Definition 4. *Let* $n, m \in \mathbb{N}$. *A* (nondeterministic) finite-state (n,m) forest automaton *((n,m)-*fsfa*) is a tuple* $\mathfrak{A} = \langle Q, \Sigma, I, F, E\rangle$, *where* Q *is a finite set of* states, Σ *is a ranked alphabet,* $I = I_1 \times \ldots \times I_n$ *(with* $I_1, \ldots, I_n \subseteq Q$*) is the set of* initial state tuples, $F = F_1 \times \ldots \times F_m$ *(with* $F_1, \ldots, F_m \subseteq Q$*) is the set of* final state tuples, *and*

$$E = \{E_k \mid k \geq 0\}, \quad \textit{where} \quad E_k \subseteq Q^k \times \Sigma_k \times Q \quad \textit{(for every } k \geq 0\textit{)}$$

is the set of state transition relations. *Since* Σ *is finite, the relation* E_k *is empty for almost all* $k \in \mathbb{N}$.

In order to define the language accepted by an fsfa $\mathfrak{A}$, it is necessary to extend the relations E_k from symbols in Σ to forests in $\Sigma^{\circledast}$. For this let us define for every $k, l \geq 0$ a relation $E^{\mathfrak{A}}_{k,l} \subseteq (\mathfrak{P}(Q))^k \times \Sigma^{\circledast}[k,l] \times Q^l$ (where $\mathfrak{P}(Q)$ denotes the powerset of Q). Informally speaking, we have $(\langle Q_1, \ldots, Q_k\rangle, t, \langle p_1, \ldots, p_l\rangle) \in E^{\mathfrak{A}}_{k,l}$ iff the forest $t \in \Sigma^{\circledast}[k,l]$ can cause a transformation from the states in $\langle Q_1, \ldots, Q_k\rangle$ to the state tuple $\langle p_1, \ldots, p_l\rangle$. The formal definition is given inductively on the structure of the elements of $\Sigma^{\circledast}[k,l]$.

(i) For every $k \geq 0$ and $Q_1, \ldots, Q_k \subseteq Q$, we have $(\langle Q_1, \ldots, Q_k\rangle, \langle\rangle_k, \langle\rangle) \in E^{\mathfrak{A}}_{k,0}$, where $\langle\rangle_k \in \Sigma^{\circledast}[k,0]$ is the unique $(k,0)$-forest (cf. Def. 2) and $\langle\rangle \in Q^0$ is the empty tuple of states,

(ii) for all $k \geq 0$ and $Q_1, \ldots, Q_k \subseteq Q$ and $i \in \{1, \ldots, k\}$ and $q \in Q_i$, we have $(\langle Q_1, \ldots, Q_k\rangle, \pi^k_i, q) \in E^{\mathfrak{A}}_{k,1}$,

(iii) for all $k, k' \geq 0$ and $t \in \Sigma^{\circledast}[k,k']$ and $\sigma \in \Sigma_{k'}$ and $Q_1, \ldots, Q_k \subseteq Q$ and $p_1, \ldots, p_{k'}, q \in Q$, if $(\langle Q_1, \ldots, Q_k\rangle, t, \langle p_1, \ldots, p_{k'}\rangle) \in E^{\mathfrak{A}}_{k,k'}$ and $(\langle p_1, \ldots, p_{k'}\rangle, \sigma, q) \in E_{k'}$, then $(\langle Q_1, \ldots, Q_k\rangle, t\sigma, q) \in E^{\mathfrak{A}}_{k,1}$,

(iv) for all $k \geq 0$ and $l \geq 2$ and $Q_1, \ldots, Q_k \subseteq Q$ and $p_1, \ldots, p_l \in Q$ and $t_1, \ldots, t_l \in \Sigma^{\circledast}[k,1]$, if $(\langle Q_1, \ldots, Q_k\rangle, t_i, p_i) \in E^{\mathfrak{A}}_{k,1}$, for every $i \in \{1, \ldots, l\}$, then $(\langle Q_1, \ldots, Q_k\rangle, \langle t_1, \ldots, t_l\rangle, \langle p_1, \ldots, p_l\rangle) \in E^{\mathfrak{A}}_{k,l}$.

Definition 5. *Let $n, m \in \mathbb{N}$ and $\mathfrak{A} = \langle Q, \Sigma, I, F, E\rangle$ be an (n,m)-fsfa. Then $L(\mathfrak{A}) = \{\ t \in \Sigma^{\circledast}[n,m] \mid \exists \langle p_1, \ldots, p_m\rangle \in F\ .\ (\langle I_1, \ldots, I_n\rangle, t, \langle p_1, \ldots, p_m\rangle) \in E^{\mathfrak{A}}_{n,m}\ \}$ is the language* accepted *(or* recognized*) by $\mathfrak{A}$.*

Example 4. Let Σ be as before. Define the $(0,1)$-fsfa $\mathfrak{A} = \langle Q, \Sigma, I, F, E\rangle$ by $Q = \{p, f\}$ and $I = \emptyset$ and $F = \{f\}$ and

$$E_0 = \{(\langle\rangle, \alpha, p), (\langle\rangle, \alpha, f)\}\ , \quad E_1 = \{(\langle p\rangle, \gamma, p)\}\ , \quad E_2 = \{(\langle p, f\rangle, \sigma, f)\}\ .$$

Then $L(\mathfrak{A})$ is the set of all forests with the shape shown on the right of Fig. 1.

Definition 6. *A (n,m)-forest language $L \subseteq \Sigma^{\circledast}[n,m]$ is called* recognizable *if there is an (n,m)-fsfa $\mathfrak{A} = \langle Q, \Sigma, I, F, E\rangle$ with $L(\mathfrak{A}) = L$. The set of all recognizable (n,m)-forest languages over Σ is denoted by $\mathsf{Rec}(\Sigma^{\circledast})[n,m]$. Further,*

$$\mathsf{Rec}(\Sigma^{\circledast}) = \bigcup_{n,m \in \mathbb{N}} \mathsf{Rec}(\Sigma^{\circledast})[n,m]$$

is the set of all recognizable forest languages over Σ.

The concept of fsta is usually introduced by means of term rewriting systems [11,21]. It should be obvious that such an fsta is the same as a $(0,1)$-fsfa (Def. 4): a $(0,1)$-fsfa has no initial states, the set of final state tuples is reduced to a set of final states and the relations E_k contain exactly the same information as the rules of the term rewriting system of a tree automaton, since there is not much difference between the tuple $(\langle q_1, \ldots, q_k\rangle, \sigma, q) \in E_k$ and the rewrite rule $\sigma(q_1, \ldots, q_k) \to q$ (with $\sigma \in \Sigma_k$). This means that for a given Σ, the set $\mathsf{Rec}(\Sigma^{\circledast})[0,1]$ is isomorphic to the set of recognizable tree languages over Σ. Sometimes the discussion is casted in terms of tree languages over a ranked alphabet Σ and a finite set of variables X. In [11], the set of recognizable tree languages over Σ and X is denoted by $\mathsf{Rec}(\Sigma, X)$. If $n = |X|$ is the number of symbols in X, then $\mathsf{Rec}(\Sigma, X)$ is isomorphic to $\mathsf{Rec}(\Sigma^{\circledast})[n,1]$.

Definition 7. *An (n,m)-fsfa $\mathfrak{A} = \langle Q, \Sigma, I, F, E\rangle$ is called* deterministic *if $|I| = 1$ (i.e., there is only one input state tuple), and for every $k \geq 0$, every $\sigma \in \Sigma_k$ and every $q_1, \ldots, q_k, p_1, p_2 \in Q$, if $(\langle q_1, \ldots, q_k\rangle, \sigma, p_1) \in E_k$ and $(\langle q_1, \ldots, q_k\rangle, \sigma, p_2) \in E_k$, then $p_1 = p_2$ (i.e., the next state is uniquely determined). An (n,m)-fsfa $\mathfrak{A} = \langle Q, \Sigma, I, F, E\rangle$ is called* totally defined *if for every $k \geq 0$, every $\sigma \in \Sigma_k$ and $q_1, \ldots, q_k \in Q$, there is a $p \in Q$ such that $(\langle q_1, \ldots, q_k\rangle, \sigma, p) \in E_k$.*

For a deterministic and totally defined (n,m)-fsfa $\mathfrak{A}$, the relations $E_k \subseteq Q^k \times \Sigma_k \times Q$ are graphs of functions $\delta_k : Q^k \times \Sigma_k \to Q$. In this case the definitions can be simplified by the use of functions $\delta^{\mathfrak{A}}_{k,l} : Q^k \times \Sigma^{\circledast}[k,l] \to Q^l$ instead of the relations $E^{\mathfrak{A}}_{k,l} \subseteq (\mathfrak{P}(Q))^k \times \Sigma^{\circledast}[k,l] \times Q^l$. The functions $\delta^{\mathfrak{A}}_{k,l}$ define a functor from

the free cartesian category $\Sigma^{\circledast}$ to a cartesian category with finite hom-sets. More explicitely, for a determinisic $(0,1)$-fsfa, the function $\delta^{\mathfrak{A}}_{0,1} : \Sigma^{\circledast}[0,1] \to Q$ is a Σ-algebra homomorphism whose inverse image of $F \subseteq Q$ is $L(\mathfrak{A}) \subseteq \Sigma^{\circledast}[0,1]$, as it is in the definition of deterministic tree automata [11]. We have the following proposition, whose proof uses the well-known power set construction.

Proposition 1. *Let $n, m \in \mathbb{N}$ and $\mathfrak{A}$ be an (n,m)-fsfa. Then there exists a deterministic and totally defined (n,m)-fsfa $\mathfrak{A}'$ such that $L(\mathfrak{A}') = L(\mathfrak{A})$.*

In the view of this, the definition of $E^{\mathfrak{A}}_{n,m}$ could be replaced by the much simpler construction in the deterministic automaton. However, as in the case of automata over words, the construction in the proof of the Kleene theorem requires non-deterministic automata and a properly defined state transition relation, which is the reason for not using rewriting rules.

4 Closure Properties of Recognizable Forest Languages

As mentioned before, the set $\mathsf{Rec}(\Sigma^{\circledast})[n,1] \subseteq \mathfrak{P}(\Sigma^{\circledast}[n,1])$ of recognizable $(n,1)$-forest languages over Σ is isomorphic to the set $\mathsf{Rec}(\Sigma, X) \subseteq \mathfrak{P}(T_{\Sigma}(X))$, where $|X| = n$, of recognizable tree languages [11]. Hence, $\mathsf{Rec}(\Sigma^{\circledast})[n,1]$ is closed under the boolean operators intersection, union, and complement.

Proposition 2. *Let Σ be given and $n \in \mathbb{N}$. If $L_1, L_2 \in \mathsf{Rec}(\Sigma^{\circledast})[n,1]$, then $L_1^C = \Sigma^{\circledast}[n,1] \setminus L_1$ as well as $L_1 \cup L_2$ and $L_1 \setminus L_2$ are recognizable.*

Proposition 3. *Let Σ be a ranked alphabet and $n, m \in \mathbb{N}$. If $L_1, L_2 \in \mathsf{Rec}(\Sigma^{\circledast})[n,m]$, then $L_1 \cap L_2 \in \mathsf{Rec}(\Sigma^{\circledast})[n,m]$.*

It is important to notice, that the set $\mathsf{Rec}(\Sigma^{\circledast})[n,m]$ is in general not closed under union. However, the recognizable languages are closed under cartesian product. With this as a base, we can define another operation $\uplus$ which will take the place of the union.

Definition 8. *If $L_1 \subseteq \Sigma^{\circledast}[n,m_1]$ and $L_2 \subseteq \Sigma^{\circledast}[n,m_2]$, then*

$$L_1 \times L_2 = \{\, \langle t_1, \ldots, t_{m_1+m_2}\rangle \in \Sigma^{\circledast}[n, m_1+m_2] \mid \langle t_1, \ldots, t_{m_1}\rangle \in L_1 \text{ and } \langle t_{m_1+1}, \ldots, t_{m_1+m_2}\rangle \in L_2 \,\}$$

is the cartesian product *of L_1 and L_2.*

Proposition 4. *Let Σ be a ranked alphabet and $n, m \in \mathbb{N}$. Further, let $L \subseteq \Sigma^{\circledast}[n,m]$. Then $L \in \mathsf{Rec}(\Sigma^{\circledast})[n,m]$ iff there are $L_1, \ldots, L_m \in \mathsf{Rec}(\Sigma^{\circledast})[n,1]$, such that $L = L_1 \times \ldots \times L_m$.*

Note that this implies that $\mathsf{Rec}(\Sigma^{\circledast})[n,m] = \mathsf{Rec}(\Sigma^{\circledast})[n,1]^m$.

Definition 9. *Let Σ be a ranked alphabet, let $n, m, k \in \mathbb{N}$, and let $L_1 \subseteq \Sigma^{\circledast}[n,m]$ and $L_2 \subseteq \Sigma^{\circledast}[m,k]$. Then $L_1; L_2 = \{t_1; t_2 \mid t_1 \in L_1, t_2 \in L_2\}$ is the* naive concatenation *of L_1 and L_2.*

In the case that L_2 is a singleton, say $L_2 = \{\nu\}$, I will write $L_1;\nu$ instead of $L_1;\{\nu\}$, for notational convenience. Similarly, $\nu;L_2$ stands for $\{\nu\};L_2$.

Definition 10. *Let $L, L' \subseteq \Sigma^{\circledast}[n,m]$. Then*

$$L \uplus L' = (L;\pi_1^m \cup L';\pi_1^m) \times \ldots \times (L;\pi_m^m \cup L';\pi_m^m)$$

is the componentwise union *of L and L'. Similarly, for a family $\{L_i \mid i \in I\}$ of languages with $L_i \subseteq \Sigma^{\circledast}[n,m]$ for all $i \in I$, define*

$$\biguplus_{i\in I} L_i \;=\; \left(\bigcup_{i\in I} L_i;\pi_1^m\right) \times \ldots \times \left(\bigcup_{i\in I} L_i;\pi_m^m\right) \quad .$$

Proposition 5. *Let Σ be a ranked alphabet, and $n,m \in \mathbb{N}$. If $L, L' \in \mathsf{Rec}(\Sigma^{\circledast})[n,m]$, then $L \uplus L' \in \mathsf{Rec}(\Sigma^{\circledast})[n,m]$.*

Definition 11. *Let Σ be a ranked alphabet and $n,m,k \in \mathbb{N}$. Further, let $L \subseteq \Sigma^{\circledast}[n,m]$ and $t \in \Sigma^{\circledast}[m,k]$. Define the* concatenation $L{\cdot}t$ *of L and t by induction on t as follows:*

$$\begin{aligned}
L \cdot \langle\rangle_m &= \{\langle\rangle_n\} \quad , \\
L \cdot \pi_i^m &= L;\pi_i^m \quad \textit{(for every } i = 1,\ldots,m\textit{)}, \\
L \cdot (t'\sigma) &= (L \cdot t');\varepsilon_{k'}\sigma \quad \textit{(for every } k' \geq 0,\ t' \in \Sigma^{\circledast}[m,k']\textit{, and } \sigma \in \Sigma_{k'}\textit{)}, \\
L \cdot \langle t_1,\ldots,t_k\rangle &= L \cdot t_1 \times \ldots \times L \cdot t_k \quad \textit{(for every } t_1,\ldots,t_k \in \Sigma^{\circledast}[m,1]\textit{)}.
\end{aligned}$$

If $L' \subseteq \Sigma^{\circledast}[m,k]$, then $L \cdot L' = \biguplus_{t\in L'} L \cdot t$ is the concatenation *of L and L'.*

The construction of the forest concatenation might seem unnatural, but the usual tree concatenation [6,11] is defined in a similar way: Different occurrences of the same 0-ary symbol can be replaced by different trees. Although the recognizable forest languages are not closed under the naive concatenation, they are closed under concatenation. This means that also $\mathsf{Rec}(\Sigma^{\circledast})$ forms a cartesian category (see also [13]).

Proposition 6. *Let Σ be a ranked alphabet, and $n,m,k \in \mathbb{N}$.*
If $L_1 \in \mathsf{Rec}(\Sigma^{\circledast})[n,m]$ and $L_2 \in \mathsf{Rec}(\Sigma^{\circledast})[m,k]$, then $L_1 \cdot L_2 \in \mathsf{Rec}(\Sigma^{\circledast})[n,k]$.

For the proof we need the concept of *normalized* fsfa, where there is only one input state tuple and only one output state tuple and there are no transitions from a final state or into an initial state. Then, the two normalized fsfa for L_1 and L_2 are connected.

Now we can define a generalization of Kleene's star operation as a closure operation for the concatenation. For the concatenation operation of a free monoid this does not bring any problems. Note that this makes sense only for languages $L \subseteq \Sigma^{\circledast}[n,n]$, and we do not take the union of all L^z for $0 \leq z < \omega$ (as it is done for word languages) but use the operation $\biguplus$ instead.

Definition 12. *For $L \subseteq \Sigma^{\circledast}[n,n]$ define*

$$L^0 = \{\varepsilon_n\} = \{\langle \pi_1^n, \ldots, \pi_n^n \rangle\} \quad ,$$
$$L^{z+1} = L^z \cdot L \quad \text{for every } z \geq 0 \quad ,$$
$$L^\star = \biguplus_{z \geq 0} L^z \quad .$$

This star operation is a generalization of both, Kleene's star operation for word languages and the star operations for trees defined in [6].

Proposition 7. *If $L \in \mathsf{Rec}(\Sigma^{\circledast})[n,n]$, then $L^\star \in \mathsf{Rec}(\Sigma^{\circledast})[n,n]$.*

The basic idea of the proof is to construct the normalized fsfa $\mathfrak{A}'$ for L and identify initial and final states. However, note that the situation is not entirely trivial because in general $L(\mathfrak{A}') \neq L$. The reason for this is that the forests containing π_i^n are removed in $L(\mathfrak{A}')$ because otherwise $\mathfrak{A}'$ would not be normal. We have to construct another automaton $\mathfrak{A}''$, which reintroduces these forests.

5 Rational Forest Languages

The set of rational forest languages is the smallest set that contains the finite languages and that is closed under componentwise union, cartesian product, concatenation, and the star operation.

Definition 13. *Let Σ be a ranked alphabet. For every $n, m \in \mathbb{N}$, the set $\mathsf{Rat}(\Sigma^{\circledast})[n,m]$ is the smallest set such that*

- *$\emptyset \in \mathsf{Rat}(\Sigma^{\circledast})[n,m]$, and if $t \in \Sigma^{\circledast}[n,m]$, then $\{t\} \in \mathsf{Rat}(\Sigma^{\circledast})[n,m]$,*
- *if $L_1, L_2 \in \mathsf{Rat}(\Sigma^{\circledast})[n,m]$, then $L_1 \uplus L_2 \in \mathsf{Rat}(\Sigma^{\circledast})[n,m]$,*
- *if $L_1 \in \mathsf{Rat}(\Sigma^{\circledast})[n,m_1]$ and $L_2 \in \mathsf{Rat}(\Sigma^{\circledast})[n,m_2]$, then $L_1 \times L_2 \in \mathsf{Rat}(\Sigma^{\circledast})[n,m_1+m_2]$,*
- *if $L_1 \in \mathsf{Rat}(\Sigma^{\circledast})[n,k]$ and $L_2 \in \mathsf{Rat}(\Sigma^{\circledast})[k,m]$, then $L_1 \cdot L_2 \in \mathsf{Rat}(\Sigma^{\circledast})[n,m]$,*
- *if $L \in \mathsf{Rat}(\Sigma^{\circledast})[n,n]$, then $L^\star \in \mathsf{Rat}(\Sigma^{\circledast})[n,n]$.*

The set of all rational forest languages over Σ is defined as

$$\mathsf{Rat}(\Sigma^{\circledast}) = \bigcup_{n,m \in \mathbb{N}} \mathsf{Rat}(\Sigma^{\circledast})[n,m] \quad .$$

From the propositions in the previous section it follows that for every ranked alphabet Σ and $n, m \in \mathbb{N}$ we have that $\mathsf{Rec}(\Sigma^{\circledast})[n,m] \subseteq \mathsf{Rat}(\Sigma^{\circledast})[n,m]$. The following theorem says that the converse is also true.

Theorem 1. $\mathsf{Rat}(\Sigma^{\circledast})[n,m] = \mathsf{Rec}(\Sigma^{\circledast})[n,m]$.

In the proof of this theorem, the rational expression is constructed inductively on the size of the fsfa. The construction is inspired by Kleene's original construction [1], but it is quite different from the one used by Thatcher and Wright [6].

In particular, note that *a priori* there is no relation between $\mathsf{Rat}(\Sigma^{\circledast})[n,1]$ and the rational languages of [6]. Hence, Theorem 1 is not a consequence of the result in [6]. But from Theorem 1 and [6] it follows that $\mathsf{Rat}(\Sigma^{\circledast})[n,1]$ coincides with the rational languages of [6].

Example 5. For the forest language recognized by the automaton in Example 4 we can obtain the rational expression depicted in Fig. 3:

$$L = \{\langle\rangle_0\alpha\} \uplus \big(\{\langle\rangle_0\alpha\} \cdot \{\pi_1^1\gamma\}^\star \times \{\langle\rangle_0\alpha\}\big) \cdot \{\langle\pi_1^2, \langle\pi_1^2, \pi_2^2\rangle\sigma\rangle\}^\star \cdot \{\langle\pi_1^2, \pi_2^2\rangle\sigma\} \quad .$$

By using Notation 1, we can write this more concisely as

$$L = \{\alpha\} \uplus (\{\alpha\} \cdot \{\gamma\}^\star \times \{\alpha\}) \cdot \{\langle\pi_1^2, \sigma\rangle\}^\star \cdot \{\sigma\} \quad .$$

Now we can immediately derive the main theorem of this paper.

Theorem 2. *Let Σ be a ranked alphabet. Then* $\mathsf{Rat}(\Sigma^{\circledast}) = \mathsf{Rec}(\Sigma^{\circledast})$.

6 Conclusions and Future Work

The algebraic concepts used in this paper have been studied for a long time (e.g., [12,14,16,13]), but the construction of rational expression for the Kleene theorem (Theorem 2) in this paper has (up to my knowledge) not yet appeared in the literature. Apart from the independent interest of the result, the work is motivated by the following issues of future research.

- The original motivation for proposing this new approach comes from the desire to give algebraic characterizations for certain classes of tree transducers (e.g., [21,22]). Their counterparts for word languages, e.g., general seqential machines [23], have been algebraicly characterized by using Kleene's original theorem (see e.g. [24]). For tree transductions it is more difficult to find such characterizations because of the insufficient behaviour of the tree concatenation operation (see also [16]). The here proposed notion of rational expression allows to give an algebraic characterization of the transductions realized by sequential forest transducers [19], which are a generalization of a certain class of bottom-up tree transducers. In [16], Arnold and Dauchet encountered the same problem when studying bimorphisms for trees. Their solution uses *magmoides*. The structures $\Sigma^{\circledast}$ and $\mathsf{Rec}(\Sigma^{\circledast})$ discussed in this paper are magmoides (see also [15]).
- In [25], the result of [6] is generalized to trees over arbitrary semirings (as done in [4] for word automata). This raises the question whether also Theorem 2 can be proved for forests over arbitrary semirings.
- Another important problem is whether the result can be extended to unranked trees and hedges [10,9] which are important for practical applications [7,8]. The notion of cartesian category already provides the right algebraic structure. We only have to add additional objects (infinite ordinals) to the category of forests.

– More generally, it is now possible to generalize the notion of recognizable and rational languages from the free cartesian category to any cartesian category, in the same sense as it had been generalized from the free monoid to any monoid [2,24]. The question is in which cases Theorem 2 holds.
– The same question can be raised for the definability in monadic second order logic [6,26,7,9].

References

1. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, pp. 3–40. Princeton (1956)
2. Eilenberg, S.: Automata, Languages And Machines, vol. B. Academic Press, New York (1976)
3. Ochmański, E.: Regular behaviour of concurrent systems. Bull. Europ. Assoc. Theoret. Comput. Sci. (EATCS) 27, 56–67 (1985)
4. Schützenberger, M.P.: On the definition of a family of automata. Inform. And Control 4, 245–270 (1961)
5. Droste, M., Gastin, P.: The Kleene–Schützenberger theorem for formal power series in partially commuting variables. Information and Computation 153, 47–80 (1999)
6. Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. Math. Systems Theory 2, 57–81 (1968)
7. Neven, F.: Automata, logic, and XML. In: Bradfield, J.C. (ed.) CSL 2002 and EACSL 2002. LNCS, vol. 2471, pp. 2–26. Springer, Heidelberg (2002)
8. Martens, W., Niehren, J.: On the minimization of XML schemas and tree automata for unranked trees. J. Comput. Syst. Sci. 73(4), 550–583 (2007)
9. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007) (release October 12, 2007)
10. Courcelle, B.: On recognizable sets and tree automata. In: Nivat, M., Ait-Kaci, H. (eds.) Resolution of equations in algebraic structures, pp. 93–126. Academic Press, London (1989)
11. Gécseg, F., Steinby, M.: Tree Languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, Beyond Words, vol. 3, pp. 1–68. Springer, Heidelberg (1997)
12. Lawvere, F.W.: Functorial semantics of algebraic theories. In: Proceedings of the National Academy of Sciences, USA. Volume 50., National Academy of Sciences, Summary of PhD thesis, pp. 869–872, Columbia University (1963)
13. Bloom, S.L., Ésik, Z.: Iteration Theories. Springer, Heidelberg (1993)
14. Ésik, Z.: An axiomatization of regular forests in the language of algebraic theories with iteration. In: Gecseg, F. (ed.) FCT 1981. LNCS, vol. 117, pp. 130–136. Springer, Heidelberg (1981)
15. Ésik, Z., Weil, P.: Algebraic recognizability of regular tree languages. Theoretical Computer Science 340, 291–321 (2005)
16. Arnold, A., Dauchet, M.: Morphisms et bimorphisms d'arbres. Theoretical Computer Science 20, 33–93 (1982)
17. Bojańczyk, M.: Forest expressions. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 146–160. Springer, Heidelberg (2007)

18. Eder, E.: Properties of substitutions and unifications. Journal of Symbolic Computation 1(1), 31–46 (1985)
19. Straßburger, L.: Rational forest languages and sequential forest transducers. Master's thesis, Technische Universität Dresden (2000)
20. Perrin, D.: Finite Automata. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science. Formal Models and Semantics, vol. B, pp. 1–57. Elsevier Science Publishers, B.V., Amsterdam (1990)
21. Engelfriet, J.: Bottom-up and top-down tree transformations—a comparison. Mathematical Systems Theory 9(3), 198–231 (1975)
22. Fülöp, Z., Vogler, H.: Syntax-Directed Semantics: Formal Models Based on Tree Transducers. Springer, Heidelberg (1998)
23. Ginsburg, S., Rose, G.F.: A characterisation of machine mappings. Can. J. of Math. 18, 381–388 (1966)
24. Berstel, J.: Transductions and Context-Free Languages. Leitfäden der angewandten Mathematik und Mechanik LAMM, vol. 38. B.G. Teubner Stuttgart (1979)
25. Droste, M., Pech, C., Vogler, H.: A Kleene theorem for weighted tree automata. Theory Comput. Syst. 38(1), 1–38 (2005)
26. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Language Theory, vol. 3, pp. 389–456. Springer, Heidelberg (1997)

Determinization and Expressiveness of Integer Reset Timed Automata with Silent Transitions

P. Vijay Suman and Paritosh K. Pandya

Tata Institute of Fundamental Research, India

Abstract. ϵ-IRTA are a subclass of timed automata with ϵ moves (ϵ-TA). They are useful for modelling global sparse time base used in time-triggered architecture and distributed business processes. In a previous paper [1], the language inclusion problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ was shown to be decidable when $\mathcal{A}$ is an ϵ-TA and $\mathcal{B}$ is an ϵ-IRTA. In this paper, we address the determinization, complementation and ϵ-removal questions for ϵ-IRTA. We introduce a new variant of timed automata called GRTA. We show that for every ϵ-IRTA we can effectively construct a language equivalent 1-clock, deterministic GRTA with *periodic time guards* (but having no ϵ moves). The construction gives rise to at most a double exponential blowup in the number of locations. Finally, we show that every GRTA with periodic guards can be reduced to a language equivalent ϵ-IRTA with at most double the number of locations. Thus, ϵ-IRTA, periodic GRTA, and deterministic 1-clock periodic GRTA have the same expressive power and that they are all expressively complete with respect to the regular $\delta\checkmark$-languages. Equivalence of deterministic and nondeterministic automata also gives us that these automata are closed under the boolean operations.

1 Introduction

Timed automata (TA) are an extension of finite state automata with real-valued clocks. They have emerged as a standard theoretical model for real-time systems, and their formal properties have been well studied [2]. The expressive power of richer classes of timed automata such as timed automata with ϵ transitions (ϵ-TA) [3] and TA with periodic constraints [4] have also been explored. These extensions result in models which can elegantly specify periodic timed behaviours of the systems.

Integer Reset Timed Automata (IRTA) are a syntactic subclass of timed automata which restrict clock resets to integral time points. Additionally, if ϵ transitions are allowed, the resulting automata are called ϵ-IRTA. A recent paper showed that the notion of global but sparse time as used in time triggered architecture and distributed business processes can be naturally modeled in ϵ-IRTA (see [1]). The ϵ transitions are crucial in modelling periodic timed activities.

In the previous paper [1], it was shown that the language inclusion problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is decidable with EXPSPACE complexity if $\mathcal{A}$ is an ϵ-TA and $\mathcal{B}$ is an ϵ-IRTA. The decision procedure relies upon an effectively computable

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 728–739, 2009.

symbolic representation of the timed language L of ϵ-TA over alphabet Σ by an untimed regular language $f(L)$ over the extended alphabet $\Sigma \cup \{\delta, \checkmark\}$. The finite state automata recognizing the untimed symbolic $\delta\checkmark$-languages can be called $\delta\checkmark$-automata.

In this paper, we consider the questions of determinization, complementation and ϵ removal of ϵ-IRTA. Note that, in general, all these questions are unsolvable for ϵ-TA [2,5,3]. Though silent transitions are useful in specifying periodic properties, the use of silent transitions cannot be justified in the scenario of deterministic modeling. Hence, our aim is to characterize timed languages recognized by ϵ-IRTA in terms of some suitable ϵ-free and deterministic species of timed automata.

In the paper, we introduce a new variant of timed automata called *Generalized Reset Timed Automata* (GRTA) in which only the integer part of the clock is reset. Since the clock resets do not change the fractional parts of the clocks, the fractional values of all clocks remain equal in any configuration of a GRTA. The clocks measure the time since the last integral time point before it was reset. Moreover, to model periodic activity, we consider GRTA with periodic time guards and call the resulting model as *Per-GRTA*.

As our main result, we give a construction of a deterministic, 1-clock Per-GRTA $\mathcal{B}$ for any $\delta\checkmark$-automaton $\mathcal{A}$ such that $f(L(\mathcal{B})) = L(\mathcal{A})$. This is used to prove that ϵ-IRTA can be effectively reduced to a language equivalent 1-clock, deterministic Per-GRTA with at most double exponential blowup in the number of locations. It follows that ϵ-IRTA are closed under complementation. We also show that Per-IRTA are strictly less expressive than ϵ-IRTA. Hence, using the proposed GRTA as well as periodic time guards seem necessary for ϵ-free determinization of ϵ-IRTA.

As our second main result, we show that every Per-GRTA can be reduced to a language equivalent ϵ-IRTA with at most doubling of the number of locations. This result builds upon the techniques for reducing periodic timed automata to ϵ-TA given by Choffrut and Goldwurm [4].

Putting all the results together, we conclude that ϵ-IRTA and Per-GRTA are expressively equivalent and they are expressively complete for the class of regular $\delta\checkmark$-languages. Our reduction also shows that, expressively, ϵ-IRTA and Per-GRTA are contained within 1-clock, deterministic ϵ-TA. Moreover, equality of nondeterministic and deterministic automata in classes ϵ-IRTA and Per-GRTA implies that they are closed under boolean (intersection, union and complementation) operations. The paper also studies the expressive power of various subclasses of ϵ-IRTAand Per-GRTA. The relationships between these classes is depicted in Figure 1. We end the paper by showing that the question whether there exists an ϵ-IRTA which is language equivalent to a given ϵ-TA is undecidable. The question whether a given timed automaton is determinizable was already been shown to be undecidable [6].

Related Work. Finding determinizable and boolean closed sub-classes of timed automata has been an interesting quest. Alur *et al* [7] have shown that a subclass of timed automata called Event Recording Automata (ERA) can indeed be deter-

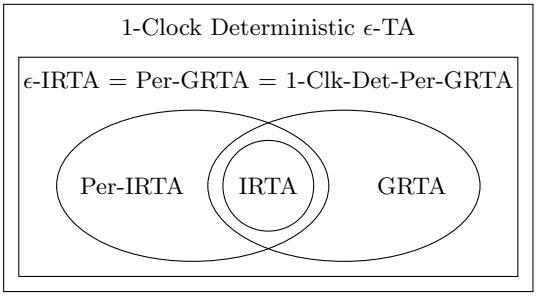


Fig. 1. Relationships between various classes. *1-Clk-Det-Per-GRTA* denotes 1-Clock, Deterministic GRTA with Periodic guards.

minized with one exponential blowup in the automaton size. Our previous work [1] showed that ϵ-IRTA and ERA are expressively incomparable. Ouaknine and Worrell [8] as well as Lasota and Walukiewicz [9] have defined 1-clock alternating timed automata which are boolean closed. Moreover the emptiness of these automata is decidable but with Non-primitive-recursive complexity. Laroussinie *et al* [10] have shown that emptiness checking for 1-clock TA is NLOGSPACE-complete although these automata are not closed under complementation. The language inclusion and universality of 1-clock timed automata has also been studied [11] and shown to be decidable with Non-primitive-recursive complexity.

2 Preliminaries

Definition 1 (Timed Word). *A finite timed word over Σ is defined as $\rho = (\sigma, \tau)$, where $\sigma = \sigma_1 \ldots \sigma_n$ is a finite sequence of symbols in Σ and $\tau = \tau_1 \ldots \tau_n$ is a finite monotone sequence of non-negative real numbers. τ_i represents the time stamp of the occurrence of the event σ_i.*

For convenience of presentation we assume a default initial time stamp $\tau_0 = 0$, prefixed to any sequence of time stamps $\tau = \tau_1 \ldots \tau_n$.

Definition 2 (Timed Automata). *A timed automaton $\mathcal{A}$ is a tuple (L, L_0, Σ, C, E, F) where (i) L is a finite set of locations, (ii) $L_0 \subseteq L$ is the set of initial locations, (iii) Σ is a finite set of symbols (called alphabet), (iv) C is a finite set of real valued clocks, (v) $E \subseteq L \times L \times \Sigma \times \Phi(C) \times 2^C$ is the set of transitions. An edge $e = (l, l', a, \varphi, \lambda)$ represents a transition from the source location l to the target location l' on input symbol a. The set $\lambda \subseteq C$ gives the set of clocks that are reset with the transition and, φ is a guard over C, and (vi)$F \subseteq L$ is the set of final locations.*

Let x represent a clock in C and k represent a natural number. $\Phi(C)$ is the set of constraints φ defined by $\varphi := x = k | x \leq k | x \geq k | x < k | x > k | \varphi \wedge \varphi$.

Definition 3 (Clock Interpretation). *Let C be the set of clocks. A clock interpretation $\nu : C \to \mathbb{R}_{\geq 0}$ maps each clock $x \in C$ to a non-negative real number.*

A state of $\mathcal{A}$ is a pair (l, ν) such that $l \in L$ and ν is a clock interpretation over C. The state space of $\mathcal{A}$ is $L \times \mathbb{R}_{\geq 0}^{|C|}$. The state of a timed automaton can change in 2 ways:

1. *Due to elapse of time*: for a state (l, ν) and a real-number $t \geq 0, (l, \nu) \xrightarrow{t} (l, \nu + t)$. This kind of transition is called a *timed transition.*
2. *Due to a location-switch*: for a state (l, ν) and an edge $(l, l', a, \varphi, \lambda)$ such that $\nu \models \varphi$, $(l, \nu) \xrightarrow{a} (l', \nu[\lambda := 0])$. We call such a transition, a Σ-transition.

Here $(\nu + t)(x) = \nu(x) + t$ and, $\nu[\lambda := 0](x) = 0$, $\forall x \in \lambda$, and remains unchanged $\forall x \in (C \backslash \lambda)$.

Definition 4 (Run, Word, Language). *A run r of a timed automaton is a sequence of alternating timed and Σ transitions: $(l_0, \nu_0) \xrightarrow{\tau_1} (l_0, \nu_1) \xrightarrow{e_1} (l_1, \nu_1') \xrightarrow{\tau_2 - \tau_1} (l_1, \nu_2) \cdots (l_{n-1}, \nu_{n-1}') \xrightarrow{\tau_n - \tau_{n-1}} (l_{n-1}, \nu_n) \xrightarrow{e_n} (l_n, \nu_n')$ with $l_0 \in L_0$ and ν_0 is such that $\nu_0(x) = 0$, $\forall x \in C$. The run r is accepting iff $l_n \in F$. Corresponding to each run, there is a timed word $(\sigma_1, \tau_1), (\sigma_2, \tau_2), \cdots, (\sigma_n, \tau_n)$ where σ_i is the label over the edge e_i, and τ_i is the time stamp of σ_i. A finite timed word $\rho = (\sigma, \tau)$ is accepted by $\mathcal{A}$ iff there exists an accepting run over $\mathcal{A}$, the word corresponding to which is ρ. The timed language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is defined as the set of all finite timed words accepted by $\mathcal{A}$.* □

We say that two automata are *equivalent* iff the (timed) languages accepted by them are the same. Let t be a non-negative real number. We use $int(t)$ and $fr(t)$ to denote respectively the integral part and the fractional part of t. We say that an edge is *resetting* if the set of clocks to be reset over it is nonempty.

Definition 5 (IRTA). *Integer reset timed automata are timed automata in which the guard over each resetting edge consists of an atomic constraint of the form $x = c$.*

In any run of an IRTA the clock resets happen only at integer time points.

Definition 6 (GRTA). *Generalized reset timed automata are a variant of timed automata wherein the clock resets are defined as follows. $\nu[\lambda := 0](x) = \nu(x) - int(\nu(x))$, $\forall x \in \lambda$, and remains unchanged $\forall x \in (C \backslash \lambda)$.*

Since resets do not perturb the fractional values of the clocks, in any configuration that arises in a GRTA the clocks have equal fractional parts. Note that GRTA is a generalization of IRTA.

Definition 7 (Periodic Clock Constraints). *An atomic periodic clock constraint is in either of the following forms: $\exists k \in \mathbb{N} : x \in [a + kp, b + kp]$, $\exists k \in \mathbb{N} : x \in (a + kp, b + kp]$, $\exists k \in \mathbb{N} : x \in [a + kp, b + kp)$, or $\exists k \in \mathbb{N} : x \in (a + kp, b + kp)$, where $a, b, p \in \mathbb{N}$.*

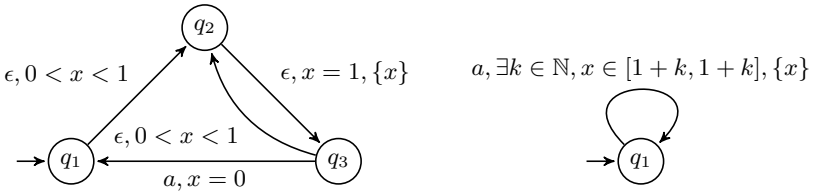


Fig. 2. An ϵ-IRTA and a Per-IRTA equivalent to it

Note that an aperiodic constraint can be expressed as a periodic constraint. For example, the constraint $x \geq 3$ is equivalent to the periodic constraint $\exists k \in \mathbb{N} : x \in [3+k, 4+k)$.

Definition 8 (Per-IRTA). *Per-IRTA are IRTA wherein the atomic clock constraints can be periodic and, the guard over each resetting edge consists an atomic constraint of the form* $\exists k \in \mathbb{N} : x \in [a+kp, a+kp]$.

In any run of a Per-IRTA, the clock resets happen only at integer time points.

Definition 9 (Per-GRTA). *Per-GRTA are GRTA wherein the atomic clock constraints can be periodic*

Definition 10 (ϵ-IRTA, ϵ-GRTA). *ϵ-IRTA and ϵ-GRTA are respectively generalizations of IRTA and GRTA, wherein transitions can have ϵ as label to designate silent moves. While a run can have ϵ-transitions, these symbols or their time stamp are not recorded in the timed word corresponding to the run (see [3]).*

Figure 2 shows an ϵ-IRTA and a Per-IRTA both of which accept the language $L_1 = \{\langle (a, \tau_1) \ldots (a, \tau_n) \rangle : \tau_i - \tau_{i-1}$ is an integer greater than $0\}$.

ϵ-IRTA formalize timed systems where the time stamps of events which occur in any open interval $(i, i+1)$ cannot be distinguished [1]. In this setting, we symbolically represent a timed word ρ over Σ by an untimed word $f(\rho)$ over $\Sigma \cup \{\delta, \checkmark\}$. In $f(\rho)$ a $\checkmark$ occurs at every integral time point and all the Σ events of ρ which occur at this time point immediately follow the symbol $\checkmark$ in order. Moreover, before every $\checkmark$ denoting the integral point i, there is a δ denoting the open interval $(i-1, i)$ and all events of ρ which occur within this open interval follow the symbol δ in order. Example 1 illustrates this representation. The formal definition of representation function f is given below. Firstly we define a $\delta\checkmark$-representation for a real number τ, and its extension for two real numbers $\tau_1 \leq \tau_2$.

Definition 11. *Let* $int(\tau) = k$.

$$dt(\tau) \triangleq \begin{cases} (\delta\checkmark)^k & \text{if } \tau \text{ is integral,} \\ (\delta\checkmark)^k\delta & \text{if } \tau \text{ is non-integral,} \end{cases}$$

Let $\tau_1 \leq \tau_2$ be two real numbers. Then $dte(\tau_1, \tau_2)$ is the $\delta\checkmark$-pattern that is to be right concatenated to $dt(\tau_1)$ to get $dt(\tau_2)$. □

For example, if $\tau_1 = 1.6$ and $\tau_2 = 2.7$, then $dt(\tau_1) = \delta\checkmark\delta$ while $dt(2.7) = \delta\checkmark\delta\checkmark\delta$. Therefore, $dte(\tau_1, \tau_2) = \checkmark\delta$.

Definition 12. *Given a timed word $\rho = (\sigma, \tau)$, the map $f(\rho)$ is defined as the untimed word $w_1\sigma_1 w_2\sigma_2 \ldots w_n\sigma_n$, where each w_i is $dte(\tau_{i-1}, \tau_i)$.* □

Example 1. For example, let $\rho_1 = \langle(a, 1.2), (b, 3.5), (c, 4), (d, 4.5), (e, 4.5), (f, 5.6), (g, 5.8))\rangle$, $\rho_2 = \langle(a, 0), (b, 0), (c, 0.5), (c, 0.6), (d, 2)\rangle$ and $\rho_3 = \langle (a, 0), (b, 0), (c, 0.3), (c, 0.7), (d, 2)\rangle$ be three timed words. Then, $f(\rho_1) = \delta\checkmark\delta a\checkmark\delta\checkmark\delta b\checkmark c\delta d e\checkmark\delta f g$ and $f(\rho_2) = f(\rho_3) = ab\delta cc\checkmark\delta\checkmark d$.

Definition 13 (f-Equivalence). *Two timed words ρ and ρ' are said to be f-equivalent, denoted by $\rho \cong \rho'$ iff $f(\rho) = f(\rho')$.* □

The central property of ϵ-IRTA is that its languages are closed under f-equivalence.

Theorem 1. *If $\mathcal{A}$ is an ϵ-IRTA and $\rho \cong \rho'$ then, $\rho \in L(\mathcal{A})$ iff $\rho' \in L(\mathcal{A})$ ([1]).*

3 $\delta\checkmark$-Regular Languages

A $\delta\checkmark$-word over Σ is a word over $\Sigma \cup \{\delta, \checkmark\}$ where ignoring the letters from Σ the δ and $\checkmark$ strictly alternate and the first such symbol (if any) must be δ. Moreover, the last letter of the word must be from Σ. For example, $a\delta bc\checkmark ab\delta c$ is a $\delta\checkmark$-word.

Definition 14. *Let $||$ be shuffle operator in the classic regular expression theory. The set of all $\delta\checkmark$-words over Σ is given by $(\Sigma^* \;||\; ((\delta\checkmark)^* + \delta(\checkmark\delta)^*)).\Sigma$. A regular set of $\delta\checkmark$-words is called a $\delta\checkmark$-regular language. A deterministic finite state automaton is called a $\delta\checkmark$-automaton iff the language accepted by it is $\delta\checkmark$-regular.*

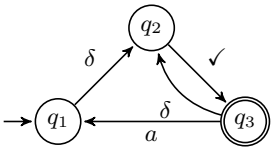


Fig. 3. A $\delta\checkmark$-automaton

Normalized $\delta\checkmark$-Automata. Let A be a given $\delta\checkmark$-automaton. We say that q_1 reaches q_2 by an extended time step of δ type, denoted $q_1 \overset{\delta}{\Longrightarrow} q_2$, if we have $q_1 \xrightarrow{u\delta v} q_2$ for some $u, v \in \Sigma^*$. Similarly, we can define extended time step of $\checkmark$ type $q_1 \overset{\checkmark}{\Longrightarrow} q_2$. A state q of automaton A is said to be $post(\delta)$ provided all incoming extended time steps are of δ type. A state is said to be of $post(\checkmark)$ type if all incoming extended time steps are of $\checkmark$ type. By convention the initial

state and all states reachable from it with only Σ-transitions are $post(\checkmark)$-type. In general, the automaton can have a state q with incoming time steps of both types (call such states mixed type). A $\delta\checkmark$-automaton is said to be *normalized* iff there are no mixed type states. It is easy to transform any $\delta\checkmark$-automaton to an equivalent normalized $\delta\checkmark$-automaton (See full version of this paper for details).

$\delta\checkmark$-Languages of ϵ-IRTA. The abstraction f mapping a timed word ρ into a $\delta\checkmark$-word $f(\rho)$ was defined in the previous section. This also maps a timed regular language L into a $\delta\checkmark$-language $f(L)$. The following results were proved in a previous paper [1].

Proposition 1. *For every $\mathcal{A} \in \epsilon$-TA we can effectively construct normalized $\delta\checkmark$-automaton $\mathcal{B}$ such that $L(B) = f(L(A))$. Thus, $f(L(\mathcal{A}))$ is $\delta\checkmark$-regular. Moreover, if $\mathcal{A}$ is an ϵ-IRTA, then $f^{-1}(f(L(A))) = L(A)$. (In general for ϵ-TA, we can only show that $f^{-1}(f(L(A))) \supseteq L(A)$).*

Complexity. If k is the maximum constant appearing in the constraints of $\mathcal{A} = (L, L_0, \Sigma, C, E, F)$ then, the number of states in $\mathcal{B}$ is upperbounded by $2^{4.|L|.(k+1)^{|C|+1}}$. We refer the reader to the original paper for details. Thus the reduction from ϵ-IRTA to normalized (deterministic) $\delta\checkmark$-automaton results in doubly exponential blowup in number of locations.

Proposition 2. *The following structural properties hold for a normalized $\delta\checkmark$-automaton $\mathcal{A}$.*

1. *$\mathcal{A}$ is bipartite with the partitions corresponding $post(\delta)$ and $post(\checkmark)$ states. The initial state is in $post(\checkmark)$ partition.*
2. *The Σ-transitions stay within the same partition. A δ-transition takes only a $post(\checkmark)$-type state to only a $post(\delta)$-type state and a $\checkmark$-transition takes only a $post(\delta)$-type state to only a $post(\checkmark)$-type state.*
3. *Let $Gr_{\delta\checkmark}(q)$ be the subgraph of $\mathcal{A}$ which consists of states reachable from a state q with only δ and $\checkmark$-transitions. Either $Gr_{\delta\checkmark}(q)$ is a simple path or is a simple path ending with a loop. Hence, a state cannot be part of two different pure $\delta\checkmark$-loops.* □

In the rest of the paper, we assume that the $\delta\checkmark$-automaton is always normalized.

4 Per-GRTA and ϵ-Removal

Our aim is to characterize timed languages recognized by ϵ-IRTA in terms of some suitable deterministic class of timed automata (See [2,4] for definition) which are also ϵ-free. One main use of the ϵ-transitions is to model periodic timed activity. Periodic timed constraints (See Definition 7) have been introduced as a direct construct for expressing periodic activities [4]. Hence Per-IRTA can be considered as a candidate for ϵ-free determinization of ϵ-IRTA. Unfortunately Lemma 1 below shows that the class Per-IRTA does not have the power to express all ϵ-IRTA languages.

In Section 2, a variant of timed automata with periodic constraints, called Per-GRTA, were introduced. In this section we show a reduction from ϵ-IRTA to equivalent 1-clock, deterministic Per-GRTA.

Lemma 1. *Per-IRTA* $\subsetneq$ *ϵ-IRTA.*

Proof. Consider the ϵ-IRTA $\mathcal{A}$ shown in Figure 4 (first automaton). $L(\mathcal{A}) = \{\langle(a,\tau_1),(b,\tau_2)\rangle : \tau_1$ and τ_2 are non-integral, and are separated by at least one integer$\}$. Assume to the contrary that there exists an IRTA with periodic clock constraints $\mathcal{B} = (L, L_0, \{a, b, \}, C, E, F)$ accepting $L(\mathcal{A})$. For $n \in \mathbb{N}$, let L_n represent the set of locations reachable from L_0 on the event a at time τ_1, where $\tau_1 \in (n, n+1)$. Note that since $\mathcal{A}$ is an IRTA, the subset $L_n \in 2^L$ is completely determined by the interval $(n, n+1)$. Also note that since the first time stamp of the words in $L(\mathcal{A})$ can be any non-integral time point, L_n is nonempty for each n. Let k be the maximum constant appearing in $\mathcal{B}$ (max. is calculated ignoring the periods). Since 2^L is finite and the number of integers is infinite, there exist k_1 and k_2 such that $k < k_1 < k_2$ and $L_{k_1} = L_{k_2}$. Let $\tau_1 \in (k_1, k_1+1)$ and $\tau_2 \in (k_2, k_2+1)$ be two randomly selected time stamps. Then we know that $\langle(a,\tau_1),(b,\tau_2)\rangle \in L(\mathcal{B})$. i.e., there exists an accepting run $(l_0,\nu_0) \xrightarrow{\tau_1} (l_0,\nu_1) \xrightarrow{e_1} (l_1,\nu_1) \xrightarrow{\tau_2-\tau_1} (l_1,\nu_2) \xrightarrow{e_2} (l_f,\nu_2)$ corresponding to this word, where $\forall x \in C$, $\nu_1(x) = \tau_1$ and $\nu_2(x) = \tau_2$. Note that none of the clocks would have been reset on e_1 or e_2 since they occur at a nonintegral time point. Since $L_{k_1} = L_{k_2}$ the modified run $(l_0,\nu_0) \xrightarrow{\tau_2} (l_0,\nu_2) \xrightarrow{e_1} (l_1,\nu_2) \xrightarrow{0} (l_1,\nu_2) \xrightarrow{e_2} (l_f,\nu_2)$ is also accepting. Hence the word $\langle(a,\tau_2),(b,\tau_2)\rangle \in L(\mathcal{B})$, which is a contradiction. □

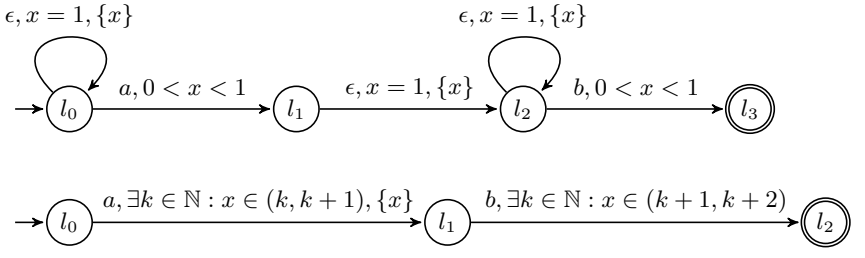


Fig. 4. An ϵ-IRTA and a Per-GRTA equivalent to it

4.1 Construction of 1-Clock Deterministic Per-GRTA

Let L be a $\delta\checkmark$-regular language and let $\mathcal{A}$ be the normalized $\delta\checkmark$-automaton accepting L. In this section we show how to construct a deterministic 1-clock Per-GRTA $\mathcal{B}$ from $\mathcal{A}$ which accepts the timed version of L. We first define *constraint* which decodes the timing information represented by $\delta\checkmark$-paths in $\mathcal{A}$.

Definition 15 (Time Distance). *Let π be a simple, pure $\delta\checkmark$-path from q_1 to q_2 in $Det(\mathcal{A}^{\checkmark}_{\delta})$. Let m be the number of $\checkmark$-transitions in π. Then,*

$$constraint(\pi) \triangleq \begin{cases} [m,m] & \text{if } q_2 \text{ is post}(\checkmark)\text{-type, and } q_2 \text{ is not in a pure } \delta\checkmark\text{-loop} \\ (m,m+1) & \text{if } q_2 \text{ is post}(\delta)\text{-type, and } q_2 \text{ is not in a pure } \delta\checkmark\text{-loop} \\ \exists k \in \mathbb{N}[m+pk, m+pk] & \text{if } q_2 \text{ is post}(\checkmark)\text{-type, and } q_2 \text{ is in a pure } \delta\checkmark\text{-loop with } p \checkmark\text{'s} \\ \exists k \in \mathbb{N}(m+pk, m+pk+1) & \text{if } q_2 \text{ is post}(\delta)\text{-type, and } q_2 \text{ is in a pure } \delta\checkmark\text{-loop with } p \checkmark\text{'s} \end{cases}$$

Construction: Let $\mathcal{A} = (Q, q_0, \Sigma \cup \{\delta, \checkmark\}, E, F)$ be the given normalized $\delta\checkmark$-automaton. We construct the deterministic 1-clock GRTA $\mathcal{B} = (L, l_0, \Sigma, \{x\}, E', F)$ as follows. Note that the number of locations in $\mathcal{B}$ is at most equal to the number of states in the $\mathcal{A}$.

- $L = \{q_0\} \cup \{q \in Q$: q has an incoming Σ-transition in $\mathcal{A}\}$. $l_0 = q_0$.
- There is an edge $(q_1, q_2, a, \varphi, \{x\}) \in E'$ iff there exists $q_3 \in Q$ such that (i) there is a simple, pure $\delta\checkmark$-path π from q_1 to q_3 in $\mathcal{A}$, (ii) the edge $(q_3, a, q_2) \in E$, and (iii) φ is defined as $x \in constraint(\pi)$. □

Figure 2 shows the Per-GRTA (which is in this case a Per-IRTA) constructed from the $\delta\checkmark$-automaton in Figure 3. In the following we prove that the resultant automaton $\mathcal{B}$ is deterministic and that $f(L(\mathcal{B})) = L(\mathcal{A})$.

Lemma 2. *The automaton $\mathcal{B}$ constructed above is deterministic.*

Proof. Let q be a location in $\mathcal{B}$ and let $e_1 = (q, q_1, a, \varphi_1, \{x\})$ and $e_2 = (q, q_2, a, \varphi_2, \{x\})$ be two edges out of q. Note that this is the case iff there is a simple, pure $\delta\checkmark$-path π_1 (π_2) from q to q_1' (q_2') such that $constraint(\pi_1) = [\varphi_1]$ ($constraint(\pi_2) = [\varphi_2]$) and there is an edge (q_1', a, q_1) ((q_2', a, q_2)) in $\mathcal{A}$. We assume w.l.o.g. that $q_1 \neq q_2$ and since $\mathcal{A}$ is deterministic this implies that $q_1' \neq q_2'$. We consider the case where both q_1' and q_2' are part of pure $\delta\checkmark$-loops. The argument is similar in the other cases. From Proposition 2 it follows that q_1' and q_2' are part of the same pure $\delta\checkmark$-loop. Hence $constraint(\pi_1)$ and $constraint(\pi_2)$ are periodic sets with the same period p but their offsets modulo p are distinct. Hence φ_1 and φ_2 are disjoint. □

Given a run, let $(l, \nu)^t$ denote that configuration (l, ν) arises in the given run at time t.

Lemma 3. *Let π be a pure $\delta\checkmark$-path in $\mathcal{A}$. There exists a superstep $q_1 \stackrel{\pi}{\Longrightarrow} q_2 \stackrel{e}{\longrightarrow} q_3$ where $e = (q_2, a, q_3)$, in $\mathcal{A}$ iff there exists a step $(q_1, \nu_1)^{\tau_1} \stackrel{\tau_2 - \tau_1}{\longrightarrow} (q_1, \nu_2)^{\tau_2} \stackrel{e'}{\longrightarrow} (q_3, \nu_3)^{\tau_2}$ where $e' = (q_1, q_3, a, \varphi, \{x\})$, in $\mathcal{B}$ such that $word(\pi) = dte(\tau_1, \tau_2)$.* □

Theorem 2. *$f(L(\mathcal{B})) = L(\mathcal{A})$. The number of locations in $\mathcal{B}$ is at most the number of states in $\mathcal{A}$.*

Proof. We use induction over steps of $\mathcal{B}$ and supersteps of $\mathcal{A}$. Using Lemma 3 we can establish that there exists an accepting run over r over a timed word $\rho = \langle(\sigma_1, \tau_1) \ldots (\sigma_n, \tau_n)\rangle$ in $\mathcal{B}$ iff there exists a corresponding accepting run r' over the word $\rho' = w_1\sigma_1 w_2\sigma_2 \ldots w_n\sigma_n$ in $\mathcal{A}$ such that (a) the states arising

after supersteps in r' are the locations of states arising after steps in r, and (b) $w_i = dte(\tau_{i-1}, \tau_i)$. From the definition of f it follows therefore that $f(\rho) = \rho'$.

Corollary 1. *For every ϵ-IRTA $\mathcal{A}$ we can effectively construct a language equivalent 1-clock, deterministic Per-GRTA $\mathcal{B}$ with at most double exponential blowup in number of locations.*

The corollary follows directly using Proposition 1 and Theorem 2. The size bound is an improvement over the triply exponential bound for determinization for the subclass IRTA given in [12]. Note that the reduction is via $\delta\checkmark$-automata and a direct translation from ϵ-IRTA to Per-GRTA seems nontrivial.

5 From Per-GRTA to ϵ-IRTA

In this section we show that for every Per-GRTA an equivalent ϵ-IRTA can be effectively constructed. Our construction is in two parts. Part 1 converts Per-GRTA into ϵ-GRTA making use of the technique of [4] originally introduced for showing that Per-TA is a subclass of ϵ-TA. Part 1 is carried out in two steps given below. In the second part the resulting ϵ-GRTA is reduced to an ϵ-IRTA.

Part 1, Step 1: The periodic guards can be put in canonical form such that there exists a common period p and each constraint consists of conjunctions of atomic constraints of the following form (where $0 \leq i < p$): $(i) x = i$, $(ii) x \in (i, i+1)$, $(iii) \exists k \in \mathbb{N} : x \in [i+kp, i+kp]$, $(iv) \exists k \in \mathbb{N} : x \in (i+kp, i+1+kp)$. Note that because of introduction of disjunction in making the guards canonical edges may have to be split into multiple edges (See [4] for the details and correctness of this construction).

Part 1, Step 2: For each clock x, a new clock $\overline{x}$ is introduced which keeps track of the value of x modulo p. Hence the periodic constraints over x could be replaced by aperiodic constraints over $\overline{x}$, of the following form: (i) $x = i$, (ii) $x \in (i, i+1)$. Additional transitions labelled with ϵ are introduced which reset $\overline{x}$ whenever $\overline{x} = p$. If an edge has two conjuncts of the form $x = i$ and $y \in (j, j+1)$ then, such a guard is not satisfiable as fractional parts of all clocks are equal, and such edges can be deleted. Hence the constraints over edges in the resultant automaton are either uniformly of the form $x = i$ or uniformly of the form $x \in (i, i+1)$. We call the former edges *integral* edges and the later kind of edges *nonintegral edges*.

Proposition 3. *Let $\mathcal{A}'$ be obtained after applying Step 1 and Step 2 to a given Per-GRTA $\mathcal{A}$. Then, $L(\mathcal{A}') = L(\mathcal{A})$.* □

Part 2: We intend to show that one can effectively construct an ϵ-IRTA $\mathcal{B}$ which is equivalent to $\mathcal{A}'$. The idea of the construction is to replace a resetting nonintegral edge by two consecutive edges. The first edge is an ϵ-transition which does the required resetting at the last integral time point (say t). The second transition is labeled by the same event as the original edge. The constraint over this edge makes sure that the transition occurs in the unit interval $(t, t+1)$. Let

φ be the constraint $\bigwedge_{x \in X} x \in (i_x, i_x + 1)$. Then $base(\varphi)$ represents the constraint $\bigwedge_{x \in X} x = i_x$ and $ext(\varphi)$ represents the constraint $\bigwedge_{x \in X} 0 < x < 1$.

Construction: Let $\mathcal{A}' = (L, L_0, \Sigma, C, E, F)$. Let E'' be the set of resetting nonintegral edges in E. The automaton $\mathcal{B} = (L', L_0, \Sigma, C, E', F)$ is defined as follows.

- $L' = L \cup \{l_e : e \in E''\}$.
- $E' = (E \setminus E'') \cup \{(l_1, l_e, \epsilon, base(\varphi), \lambda), (l_e, l_2, \sigma, ext(\varphi), \{\}):$ $e = (l_1, l_2, \sigma, \varphi, \lambda) \in E''\}$.

Lemma 4. *Let $e = (l_1, l_2, \sigma, \varphi, \lambda)$ be a resetting nonintegral edge in $\mathcal{A}'$. There exists a step $(l_1, \nu_1)^{\tau_1} \xrightarrow{\tau_2 - \tau_1} (l_1, \nu_2)^{\tau_2} \xrightarrow{e} (l_2, \nu_3)^{\tau_2}$ in $\mathcal{A}'$ iff there exists a double step $(l_1, \nu_1)^{\tau_1} \xrightarrow{i - \tau_1} (l_1, \mu_1)^{i} \xrightarrow{e_1} (l_e, \mu_2)^{i} \xrightarrow{\tau_2 - i} (l_e, \mu_3)^{\tau_2} \xrightarrow{e_2} (l_2, \nu_3)^{\tau_2}$ in $\mathcal{B}$, where $i = int(\tau_2)$, $e_1 = (l_1, l_e, \epsilon, base(\varphi), \lambda)$ and $e_2 = (l_e, l_2, \sigma, ext(\varphi), \{\})$.* □

Lemma 5. *$L(\mathcal{B}) = L(\mathcal{A}')$.*

Proof. We observe that (i) the edges which are not resetting nonintegral in $\mathcal{A}'$ are replicated in $\mathcal{B}$, and (ii) each new location l_e is not part of any edges except those from and to the source and target of e. Let ρ be a timed word. $\rho \in L(\mathcal{A}')$ iff there exists an accepting run for ρ over $\mathcal{A}'$. By induction on the number of steps/double steps and Lemma 4 it follows that such a run exists iff there exists an accepting run over $\mathcal{B}$ for ρ. Note that the ϵ-transitions are not accounted for in a timed word. □

Corollary 2. *It follows from Proposition 3 and Lemma 5 that given a Per-GRTA $\mathcal{A}$ an ϵ-IRTA $\mathcal{B}$ can be effectively constructed such that $L(\mathcal{A}) = L(\mathcal{B})$ with at most double the number of locations.* □

6 Expressiveness

Corollary 3. *From Corollaries 1 and Corollary 2, it follows that Per-GRTA = ϵ-IRTA.* □

The technique of Proposition 3 can be used to prove $\epsilon - Per - GRTA = \epsilon - GRTA = Per - GRTA$. The following Lemma and Figure 1 summarize the relationships between various classes of automata introduced in this paper.

Lemma 6. *(i) IRTA $\subsetneq$ Per-IRTA, (ii) GRTA $\not\subseteq$ Per-IRTA, (iii) IRTA $\subsetneq$ GRTA and (iv) Per-IRTA $\not\subseteq$ GRTA.* □

Theorem 3. *It is undecidable to determine whether for a given timed automaton there exists an ϵ-IRTA equivalent to it.*

Proof. We observe that the problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$?, where $\mathcal{A}$ is an ϵ-IRTA and $\mathcal{B}$ is a TA, is undecidable (follows from undecidability of universality of

timed automata). We reduce this problem to the above problem. We know that $L(\mathcal{A}) \subseteq L(\mathcal{B})$ iff $L(\mathcal{A} \cap \mathcal{B}) = L(\mathcal{A})$. Given an instance of $L(\mathcal{A}) \subseteq L(\mathcal{B})$?, we ask whether $A \cap B$ which is a timed automaton, is ϵ-IRTA-representable. If the answer is *No* then clearly $L(\mathcal{A} \cap \mathcal{B}) \subsetneq L(\mathcal{A})$. Hence $L(\mathcal{A}) \nsubseteq L(\mathcal{B})$. If the answer is *Yes* then we can check whether $L(\mathcal{A} \cap \mathcal{B}) \supseteq L(\mathcal{A})$ by the method of [1]. In this method, we can directly construct the $\delta\checkmark$-automaton for $f(L(\mathcal{A} \cap \mathcal{B}))$ without constructing an equivalent ϵ-IRTA. □

Essentially the same proof can be used to show that determining the existence of ϵ-IRTA/IRTA/GRTA/Per-GRTA equivalent to given TA/ϵ-TA is undecidable as well.

Acknowledgements. We thank Kamal Lodaya for his helpful comments.

References

1. Suman, P.V., Pandya, P.K., Krishna, S.N., Manasa, L.: Timed automata with integer resets: Language inclusion and expressiveness (LNCS 5215). In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 78–92. Springer, Heidelberg (2008)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
3. Berard, B., Petit, A., Gastin, P., Diekert, V.: Characterization of the expressive power of silent transitions in timed automata. Fundamenta Informaticae 36(2-3), 145–182 (1998)
4. Choffrut, C., Goldwurm, M.: Timed automata with periodic clock constraints. Journal of Automata, Languages and Combinatorics 5(4), 371–404 (2000)
5. Bouyer, P., Haddad, S., Reynier, P.A.: Undecidability results for timed automata with silent transitions. Research Report LSV-07-12, Laboratoire Spécification et Vérification, ENS Cachan, France, 22 pages (2007)
6. Finkel, O.: Undecidable problems about timed automata. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 187–199. Springer, Heidelberg (2006)
7. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: a determinizable class of timed automata. Theoretical Computer Science 211(1–2), 253–273 (1999)
8. Ouaknine, J., Worrell, J.: On the decidability of metric temporal logic. In: LICS 2005: Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science, Washington, DC, USA, pp. 188–197. IEEE Computer Society, Los Alamitos (2005)
9. Lasota, S., Walukiewicz, I.: Alternating timed automata. ACM Trans. Comput. Logic 9(2), 1–27 (2008)
10. Laroussinie, F., Markey, N., Schnoebelen, P.: Model checking timed automata with one or two clocks. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 387–401. Springer, Heidelberg (2004)
11. Ouaknine, J., Worrell, J.: On the language inclusion problem for timed automata: Closing a decidability gap. In: LICS 2004: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, Washington, DC, USA, pp. 54–63. IEEE Computer Society, Los Alamitos (2004)
12. Suman, P.V., Pandya, P.K., Krishna, S.N., Manasa, L.: Determinization of timed automata with integral resets. Research Report TIFR-PPVS-GM-2007/4, TIFR (2007)

One-Clock Deterministic Timed Automata Are Efficiently Identifiable in the Limit

Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen

Delft University of Technology
{S.E.Verwer,M.M.deWeerdt,C.Witteveen}@tudelft.nl

Abstract. We study the complexity of identifying (learning) timed automata in the limit from data. In previous work, we showed that in order for timed automata to be efficiently identifiable in the limit, it is necessary that they are deterministic and that they use at most one clock. In this paper, we show this is also sufficient: we provide an algorithm that identifies one-clock deterministic timed automata efficiently in the limit.

1 Introduction

Timed automata [1] (TAs) are finite state models that model timed events using an explicit notion of time. They can be used to model and reason about real-time systems [2]. In practice, however, the knowledge required to completely specify a TA model is often insufficient. An alternative is to try to induce the specification of a TA from observations. This approach is also known as *inductive inference*. The idea behind inductive inference is that it is often easier to find examples of the behavior of a real-time system than to specify the system in a direct way. Inductive inference then provides a way to find a TA model that characterizes the (behavior of the) real-time system that produced these examples.

In our case, we can monitor the occurrences of the events in a real-time system. This results in a set of labeled (positive and negative) time stamped event sequences. The exact problem we want to solve is to find the TA model that (most likely) produced this data. This is called *TA identification*. Naturally, we want to solve this problem in an efficient way, i.e., without requiring exponential amounts of space or time.

Unfortunately, identifying a TA efficiently is difficult due to the fact that the identification problem for non-timed deterministic finite state automata (DFAs) from a finite data set is already NP-complete [3]. This property easily generalizes to the problem of identifying a TA (by setting all time values to 0). Thus, unless $P = NP$, a TA cannot be identified efficiently from finite data. Even more troublesome is the fact that the DFA identification problem from finite data cannot even be approximated within any polynomial [4]. Hence (since this also generalizes), the TA identification problem from finite data is also inapproximable. However, both of these results require that the input for the identification problem is finite. While in normal decision problems this is very natural, in an identification problem the amount of input data is somewhat arbitrary: more

A.H. Dediu, A.M. Ionescu, and C. Martín-Vide (Eds.): LATA 2009, LNCS 5457, pp. 740–751, 2009.

data can be sampled if necessary. Therefore, it makes sense to study the behavior of an identification process when it is given more and more data. The framework that studies this behavior is called *identification in the limit* [5].

The identifiability of many interesting models (or languages) have been studied within the framework of learning in the limit. For instance, DFAs have been shown to be efficiently identifiable in the limit from labeled examples [6], but non-deterministic finite state automata (NFAs) have been shown not to be efficiently identifiable in this way [7]. Because of this, we restrict our attention to deterministic timed automata (DTAs). Unfortunately, in previous work we have shown that DTAs in general cannot be identified efficiently in the limit [8]. The argument we used was based on the existence of DTAs such that languages of these DTAs only contain examples of length exponential in the size of the DTA. Therefore, if we intend to identify the full class of DTAs, we will sometimes require an exponential amount of data in order to converge to the correct DTA. However, this argument no longer holds if the DTAs contain at most one timed component, known as a *clock*. In fact, in the same work, we even proved that the length of the shortest string in the symmetric difference of two one-clock DTAs (1-DTAs) can always be bounded by a polynomial. DTAs with this property are called *polynomially distinguishable*. This is a very important property for identification purposes because to identify a model is basically to distinguish models from each other based on examples. We have shown that this is a necessary requirement for efficient identification in the limit [8].

In this paper, we provide an algorithm that identifies 1-DTAs efficiently in the limit. The proof of convergence of this algorithm is based on the polynomial distinguishability of 1-DTAs. The algorithm is an important step in the direction of being able to identify real-time systems efficiently. To the best of our knowledge, our result is the first positive efficiency result for identifying timed automata. Moreover, we do not know of any other algorithm that identifies the complete structure of a timed automaton, including the clock resets.

The paper is organized as follows. We start with a brief introduction to 1-DTAs (Section 2), and a formal explanation of efficient identifiability in the limit (Section 3). We then describe our algorithm for identifying 1-DTAs (Section 4), and give the sketch of a proof that it converges to the correct 1-DTA efficiently in the limit(Section 5). We end our paper with some conclusions (Section 6).

2 Deterministic One-Clock Timed Automata

A *timed automaton* (TA) [1] is an automaton that accepts (or generates) strings with event-time value pairs, called timed strings. A *timed string* τ over a finite set of symbols Σ is a sequence $(a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$ of symbol-time value pairs $(a_i, t_i) \in \Sigma \times \mathbb{N}$.[1] Each time value t_i in a timed string represents the time passed until the occurrence of symbol a_i since the occurrence of the previous symbol a_{i-1}. We use $\tau_i = (a_1, t_1) \dots (a_i, t_i)$ to denote the prefix of length i of τ.

[1] Sometimes $\mathbb{R}$ is used as a time domain for TAs. However, for identification of TAs $\mathbb{N}$ is sufficient since, in practice, we always measure time using finite precision.

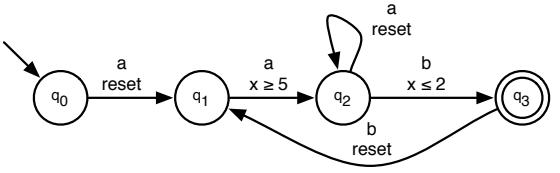


Fig. 1. A deterministic one-clock timed automaton. The start state q_0 is denoted by an arrow pointing to it from nowhere. The final state q_3 has two circles instead of one. The arrows represent transitions. The labels, clock guards, and clock resets are specified for every transition. When no guard is specified, the guard is always satisfied.

In TAs, timing conditions are added using a finite set X of *clocks* and one *clock guard* on every transition. For the purpose of this paper, i.e., identifying one-clock deterministic timed automata, we focus on the case where the TAs are *deterministic* and contain at most a *single clock*. Such a TA is called a *deterministic one-clock timed automata* (1-DTA). In TAs, *valuation* mappings $v : X \to \mathbb{N}$ are used to obtain the value of a clock $x \in X$. Since we use only a single clock x, we use v instead of $v(x)$ to denote the value of x. Every transition δ in a 1-TA contains a boolean value known as a *reset value*. When a transition δ fires (or occurs), and the reset value of δ is set to true, then the value of x is set to 0, i.e., $v := 0$. In this way, x is used to record the time since the occurrence of some specific event. Clock guards are then used to change the behavior of the 1-DTA depending on the value of x. A clock guard $g = c \leq x \leq c'$ is a boolean constraint on the value of x, where $c \in \mathbb{N}$ and $c' \in \mathbb{N} \cup \{\infty\}$ are constants.[2] A valuation v is said to *satisfy* a clock guard g, if whenever each occurrence of x in g is replaced by v the resulting statement is true. A deterministic one-clock timed automaton is defined as follows:

Definition 1. *A* deterministic one-clock timed automaton *(1-DTA) is a tuple* $\mathcal{A} = \langle Q, x, \Sigma, \Delta, q_0, F \rangle$*, where* Q *is a finite set of states,* x *is a clock,* Σ *is a finite set of symbols,* Δ *is a finite set of transitions,* q_0 *is the start state, and* $F \subseteq Q$ *is a set of final states.*

A transition $\delta \in \Delta$ *is a tuple* $\langle q, q', a, g, r \rangle$*, where* $q, q' \in Q$ *are the source and target states,* $a \in \Sigma$ *is a symbol, called the transition label,* g *is a clock guard, and* $r \in \{true, false\}$ *is a reset value. Since* $\mathcal{A}$ *is deterministic, for every source state* $q \in Q$*, every label* $a \in \Sigma$*, and every valuation* $v \in \mathbb{N}$*, there exists at most a single transition* $\langle q, q', a, g, r \rangle \in \Delta$ *such that* v *satisfies* g*.*

Figure 1 shows an example of a 1-DTA. Like a DFA, a 1-DTA moves from state to state by firing *state transitions* $\delta \in \Delta$ that connect these states. However, in addition, a 1-DTA is capable of remaining in the same state q for a while. While doing so, the valuation v of its clock x increases. We call this action of staying in the same state a time transition. A *time transition* of t time units increases the valuation v of x by t, i.e. $v := v + t$. One can view a time transition as moving

[2] Since we use the natural numbers to represent time open ($x < c$) and closed ($x \leq c$) one-clock timed automata are equivalent.

from one *timed state* (q, v) to another timed state $(q, v + t)$ while remaining in the same untimed state q. The length t of time transitions corresponds to the time values in a timed string. More precisely, the occurrence of a timed symbol (a, t) in a timed string means that before firing a transition δ labeled with a, the 1-DTA makes a time transition of t time units. After such a time transition, δ can only fire if its guard g is satisfied by $v + t$. The exact behavior of a 1-DTA is defined by what is called a *run* of a 1-DTA:

Definition 2. *A (finite)* run *of a 1-DTA* $\mathcal{A} = \langle Q, x, \Sigma, \Delta, q_0, F\rangle$ *over a timed string* $\tau = (a_1, t_1)\dots(a_n, t_n)$ *is a sequence of timed states and transitions* $(q_0, v_0) \xrightarrow{t_1} (q_0, v_0 + t_1) \xrightarrow{a_1} (q_1, v_1) \dots (q_{n-1}, v_{n-1}) \xrightarrow{t_n} (q_{n-1}, v_{n-1} + t_n) \xrightarrow{a_n} (q_n, v_n)$, *such that for all* $1 \leq i \leq n$ *there exists a transition* $\delta_i = \langle q_{i-1}, q_i, a_i, g, r\rangle \in \Delta$ *such that* $v_{i-1} + t_i$ *satisfies* g, $v_0 = 0$, *and* $v_i = 0$ *if* $r = true$, $v_i = v_{i-1} + t_i$ *otherwise.*

In a run the subsequence $(q_i, v_i + t) \xrightarrow{a_{i+1}} (q_{i+1}, v_{i+1})$ represents a state transition like in a finite automaton without time. When this occurs, the timed string τ is said to *fire* a transition δ with valuation $v_i + t$ to another timed state (q_{i+1}, v_{i+1}). The time transitions of a 1-DTA are represented by $(q_i, v_i) \xrightarrow{t_{i+1}} (q_i, v_i + t_{i+1})$. We say that a timed string τ *reaches* a timed state (q, v) in a TA $\mathcal{A}$ if there exist two time values $t \leq t'$ such that $(q, v') \xrightarrow{t'} (q, v' + t')$ occurs somewhere in the run of $\mathcal{A}$ over τ and $v = v' + t$. If a timed string reaches a timed state (q, v) in $\mathcal{A}$ for some valuation v, it also reaches the untimed state q in $\mathcal{A}$. A timed string *ends* in the last (timed) state it reaches, i.e., (q_n, v_n) or q_n. Notice that timed states in 1-DTAs are similar to the (untimed) states in DFAs: at any point during the run of a timed string τ, the state τ ends in depends on the current timed state and the remaining suffix of τ. A timed string τ is *accepted* by a 1-DTA $\mathcal{A}$ if τ ends in a final state, i.e., if $q_n \in F$. The set of all strings τ that are accepted by $\mathcal{A}$ is called the *language* $L(\mathcal{A})$ of $\mathcal{A}$.

Example 1. Consider the TA $\mathcal{A}$ of Fig. 1. The run of $\mathcal{A}$ over the timed string $\tau = (a, 5)(a, 6)(a, 2)(b, 2)$ is given by: $(q_0, 0) \xrightarrow{5} (q_0, 5) \xrightarrow{a} (q_1, 0) \xrightarrow{6} (q_1, 6) \xrightarrow{a} (q_2, 6) \xrightarrow{2} (q_2, 8) \xrightarrow{a} (q_2, 0) \xrightarrow{2} (q_2, 2) \xrightarrow{b} (q_3, 2)$. Since q_3 is a final state, it holds that $\tau \in L(\mathcal{A})$. Note that q_3 cannot be reached directly after reaching q_2 from q_1: the clock guard to q_2 is satisfied by a valuation v greater than 4, while the guard of the transition to q_3 requires v to be less than 3.

3 Efficient Identification in the Limit

An identification process tries to find (learn) a model that explains a set of observations (data). The ultimate goal of such a process is to find a model equivalent to the actual concept that was responsible for producing the observations, called the *target concept*. In our case, we try to find a 1-DTA model $\mathcal{A}$ that is equivalent to a target language L_t, i.e., $L(\mathcal{A}) = L_t$. If this is the case, we say that L_t is identified correctly. We try to find this model using *labeled data*: an *input sample*

S for L_t is a pair of finite sets of positive examples $S_+ \subseteq L_t$ and negative examples $S_- \subseteq L_t^{\mathrm{C}} = \{\tau \mid \tau \notin L_t\}$. We modify the non-strict set-inclusion operators for input samples such that they operate on the positive and negative examples separately, for example if $S = (S_+, S_-)$ and $S' = (S'_+, S'_-)$ then $S \subseteq S'$ means $S_+ \subseteq S'_+$ and $S_- \subseteq S'_-$. In addition, by $\tau \in S$ we mean $\tau \in S_+ \cup S_-$.

The input of our 1-DTA identification problem is a pair of finite sets. Unfortunately, as we already mentioned in the introduction, the 1-DTA identification problem is inapproximable. Because of this, we study the behavior of a 1-DTA identification process that is given more and more data. The framework for studying such a process is called *identification in the limit* [5]. In this framework, we do not regard the complexity for any possible input sample S, but for any possible target language L_t. For every target language L_t, there exist many possible input samples S. An identification process A is called *efficient in the limit* (from polynomial time and data) if for any target language L_t, A requires time polynomial in the size of any input sample S for L_t, and if the smallest input sample S such that A converges to L_t can be bounded by a polynomial in the size of L_t. The size of a target language L_t is defined as the size of a smallest model for L_t. Efficient identifiability in the limit can be proved by showing the existence of polynomial *characteristic sets* [7].

Definition 3. *A* characteristic set S_{cs} *of a target language L_t for an identification algorithm A is an input sample (S_+, S_-) for L_t such that:*

- *given S_{cs} as input, algorithm A identifies L_t, i.e., A returns an automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L_t$, and*
- *given any input sample $S' \supseteq S_{cs}$ as input, algorithm A still identifies L_t.*

Definition 4. *A class of automata C is* efficiently identifiable in the limit *if there exist two polynomials p and q, and an algorithm A such that:*

- *given an input sample of size $n = \sum_{\tau \in S} |\tau|$, A runs in time bounded by $p(n)$,*
- *and for every target language $L_t = L(\mathcal{A})$, $\mathcal{A} \in C$, there exists a characteristic set S_{cs} of L_t for A of size bounded by $q(|\mathcal{A}|)$.*

In previous work [8], we showed that DTAs in general are not efficiently identifiable. The argument we used was based on the fact that there exists DTAs such that languages of these DTAs only contain examples (strings) of length exponential in the size of the DTA. This was used to prove that DTAs are not *polynomially distinguishable*:

Definition 5. *A class C of automata is* polynomially distinguishable *if there exists a polynomial function p, such that for any $\mathcal{A}, \mathcal{A}' \in C$ with $L(\mathcal{A}) \neq L(\mathcal{A}')$, there exists a $\tau \in (L(\mathcal{A}) \cup L(\mathcal{A}')) \setminus (L(\mathcal{A}) \cap L(\mathcal{A}'))$, such that $|\tau| \leq p(|\mathcal{A}| + |\mathcal{A}'|)$.*

We have also shown that polynomial distinguishability is a necessary requirement for efficient identification [8]. Because of this, we cannot identify DTAs efficiently. In addition, we have proved that *1-DTAs are polynomially distinguishable*. Based on this result, we conjectured that 1-DTAs might be efficiently identifiable in

the limit. In the following sections, we prove this conjecture by first describing an algorithm for identifying 1-DTAs. We then prove the convergence of this algorithm, i.e., that it identifies 1-DTAs efficiently in the limit. This proof is based on the polynomial distinguishability of 1-DTAs.

4 Identifying 1-DTAs Efficiently in the Limit

In this section, we describe our ID_1DTA algorithm for identifying 1-DTAs from an input sample S. The main value of this algorithm is that:

- given any input sample S, ID_1DTA returns in polynomial time a 1-DTA $\mathcal{A}$ that is consistent with S, i.e., such that $S_+ \subseteq L(\mathcal{A})$ and $S_- \subseteq L(\mathcal{A})^{\mathrm{C}}$,
- and if S contains a characteristic subsample S_{cs} for some target language L_t, then ID_1DTA returns a correct 1-DTA $\mathcal{A}$, i.e., such that $L(\mathcal{A}) = L_t$.

In other words, ID_1DTA identifies 1-DTAs efficiently in the limit. Note that, in a 1-DTA identification problem, the size of the 1-DTA is not predetermined. Hence, our algorithm has to identify the complete structure of a 1-DTA, including states, transitions, clock guards, and resets. Our algorithm identifies this structure one transition at a time: it starts with an empty 1-DTA $\mathcal{A}$, and whenever an identified transition requires more states or additional transitions, these will be added to $\mathcal{A}$. In this way, ID_1DTA builds the structure of $\mathcal{A}$ piece by piece. Since we claim that ID_1DTA identifies 1-DTAs efficiently, i.e., from polynomial time and data, we require that, for any input sample S for any target language L_t, the following four properties hold for this identification process:

Property 1. Identifying a single transition δ requires time polynomial in the size of S *(polynomial time per δ)*.

Property 2. The number of such transition identifications is polynomial in the size of S *(convergence in polynomial time)*.

Property 3. For every transition δ, there exists an input sample S_{cs} of size polynomial in the size of the smallest 1-DTA for L_t such that when included in S, S_{cs} guarantees that δ is identified correctly *(polynomial data per δ)*.

Property 4. The number of such correct transition identifications that are required to return a 1-DTA $\mathcal{A}$ with $L(\mathcal{A}) = L_t$ is polynomial in the size of the smallest 1-DTA for L_t *(convergence from polynomial data)* .

With these four properties in mind, we develop our ID_1DTA algorithm for the efficient identification of 1-DTAs. Pseudo code of this algorithm is shown in Algorithm 1. In this section, we use an illustrative example to show how this algorithm identifies a single transition, and to give some intuition why the algorithm satisfies these four properties. In the next section, we prove that our algorithm indeed satisfies these four properties and thus prove that it identifies 1-DTAs efficiently in the limit.

Example 2. Suppose that after having identified a few transitions, our algorithm has constructed the (incomplete) 1-DTA $\mathcal{A}$ from Figure 2. Furthermore, suppose

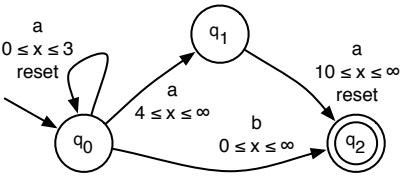


Fig. 2. A partially identified 1-DTA. The transitions from state q_0 have been completely identified. State q_1 only has one outgoing transition. State q_2 has none.

that S contains the following timed strings: $\{(a,4)(a,6),(a,5)(b,6),(b,3)(a,2),(a,4)(a,1)(a,3),(a,4)(a,2)(a,2)(b,3)\} \subseteq S_+$ and $\{(a,3)(a,10),(a,4)(a,2)(a,2),(a,4)(a,3)(a,2)(b,3),(a,5)(a,3)\} \subseteq S_-$. Our algorithm has to identify a new transition δ of $\mathcal{A}$ using information from S. There are a few possible identifiable transitions: state q_1 does not yet contain any transitions with label b, or with label a and valuations smaller than 9, and state q_2 does not yet contain any transitions at all. Our algorithm first makes a choice which transition to identify, i.e., it selects the source state, label, and valuations for a new transition. Then our algorithm actually identifies the transition, i.e., it uses S in order to determine the target state, clock guard, and reset of the transition.

As can be seen from the example, the first problem our algorithm has to deal with is to determine which transition to identify. Our algorithm makes this decision using a fixed predetermined order (independent of the input sample). The order used by our algorithm is very straightforward: first a state q is selected in the order of identification (first identified first), second a transition label l is selected in alphabetic order, and third the highest possible upper bound c' for a clock guard in this state-label combination is chosen. This fixed order makes it easier to prove the existence of characteristic sets (satisfying property 3). In our example, our algorithm will try to identify a transition $\delta = \langle q, q', l, c \leq x \leq c', r\rangle$, where $q = q_1$, $l = a$, and $c' = 9$ (since there exists a transition with a clock guard that is satisfied by a valuation $v = 10$) are all fixed. Thus, our algorithm only needs to identify: (i) the target state q', (ii) the lower bound of the clock guard c, and (iii) the clock reset r.

Note that fixing q, a, and c' in this way does not influence which transitions will be identified by our algorithm. Since we need to identify a transition with these values anyway, it only influences the order in which these transitions are identified. We now show how our algorithm identifies c, r, and q'.

The lower bound c. Our algorithm first identifies the lower bound c of the clock guard g of δ. The smallest possible lower bound for g is the smallest reachable valuation $v_{\min}$ in q (q_1 in the example). This valuation $v_{\min}$ is equal to the smallest lower bound of a transition with q as target. In the example, $v_{\min}$ is 4. Thus, the lower bound c has to be a value with the set $\{c \mid v_{\min} \leq c \leq c'\}$. One approach for finding c would be to try all possible values from this set and pick the best one. However, since time values are encoded in binary in the input sample S, iterating over such a set is exponential in the size of these time values,

Algorithm 1. Efficiently learning 1-DTAs from polynomial data: ID_1DTA

Require: An input sample $S = (S_+, S_-)$ for a language L_t, with alphabet Σ
Ensure: $\mathcal{A}$ is a 1-DTA consistent with S, i.e. $S_+ \subseteq L(\mathcal{A})$ and $S_- \subseteq L(\mathcal{A})^{\mathrm{C}}$, in addition, if it holds that $S_{cs} \subseteq S$, then $L(\mathcal{A}) = L_t$
  $\mathcal{A} := \langle Q = \{q_0\}, x, \Sigma, \Delta = \emptyset, q_0, F = \emptyset \rangle$
  if S_+ contains the empty timed string λ **then** set $F := \{q_0\}$
  while there exist a reachable timed state (q, v) and a symbol a for which there exists no transition $\langle q, q', a, g, r \rangle \in \Delta$ such that v satisfies g **do**
    for all states $q \in Q$ and symbols $a \in \Sigma$ **do**
      $v_{\min} := \min\{v \mid (q, v)$ is reachable $\}$
      $c' := \max\{v \mid \neg\exists \langle q, q', a, g, r \rangle \in \Delta$ such that v satisfies $g\}$
      while $v_{\min} \leq c'$ **do**
        create a new transition $\delta := \langle q, q' := 0, a, g := v_{\min} \leq x \leq c', r \rangle$, add δ to Δ
        $V := \{v \mid \exists \tau \in S : \tau$ fires δ with valuation $v\}$
        $r :=$ true and $c_1 :=$ lower_bound$(\delta, V \cup \{v_{\min}\}, \mathcal{A}, S)$
        $r :=$ false and $c_2 :=$ lower_bound$(\delta, V \cup \{v_{\min}\}, \mathcal{A}, S)$
        if $c_1 \leq c_2$ **then** set $r :=$ true and $g := c_1 \leq x \leq c'$
        else set $r :=$ false and $g := c_2 \leq x \leq c'$
        for every state $q'' \in Q$ (first identified first) **do**
          $q' := q''$
          if consistent$(\mathcal{A}, S)$ is true **then break else** $q' := 0$
        end for
        if $q' = 0$ **then**
          create a new state q'', set $q' := q''$, and add q' to Q
          if $\exists \tau \in S_+$ such that τ ends in q' **then** set $F := F \cup \{q'\}$
        end if
        $c' := \min\{v \mid v$ satisfies $g\} - 1$
      end while
    end for
  end while

i.e., it is exponential in the size of S (contradicting property 1). This is why our algorithm only tries those time values that are actually used by timed strings from S. We determine these in the following way. We first set the lower bound of g to be $v_{\min}$. There are now examples in S that fire δ. The set of valuations V that these examples use to fire δ are all possible lower bounds for g, i.e., $V := \{v \mid \exists \tau \in S : \tau$ fires δ with valuation $v\}$. In our example, we have that $\{(a,4)(a,1)(a,3)\} \subseteq S_+$ and $\{(a,5)(a,3),(a,4)(a,2)(a,2)\} \subseteq S_-$. In this case, $V = \{4+1=5, 5+3=8, 4+2=6\}$. Since for every time value in V there exists at least one timed string in S for every such time value, iterating over this set is polynomial in the size of S (satisfying property 1).

From the set $V \cup \{v_{\min}\}$ our algorithm selects the smallest possible consistent lower bound. A lower bound is consistent if the 1-DTA resulting from identifying this bound is consistent with the input sample S. A 1-DTA $\mathcal{A}$ is called *consistent* if S contains no positive example that inevitably ends in the same state as a negative example, i.e., if the result $\mathcal{A}$ can still be such that $S_+ \in L(\mathcal{A})$ and $S_- \in L(\mathcal{A})^{\mathrm{C}}$ (satisfying property 4). Whether $\mathcal{A}$ is consistent with S is checked

by testing whether there exist no two timed strings $\tau \in S_+$ and $\tau' \in S_-$ that reach the same timed state (possibly after making a partial time transition) and afterwards their suffixes are identical. We use consistent to denote this check. This check can clearly be done in polynomial time (satisfying property 1). Our algorithm finds the smallest consistent lower bound by trying every possible lower bound $c \in V \cup \{v_{\min}\}$, and testing whether the result is consistent. We use lower_bound to denote this routine. This routine ensures that at least one timed string from S will fire δ, and hence that our algorithm only identifies a polynomial amount of transitions (satisfying property 2). In our example, setting c to 5 makes $\mathcal{A}$ inconsistent since now both $(a,4)(a,1)(a,3)$ and $(a,4)(a,2)(a,2)$ reach $(q',6)$, where q' is any possible target for δ, and afterwards they have the same suffix $(a,2)$. However, setting c to 6 does not make $\mathcal{A}$ inconsistent. Since 6 is the smallest value in $V \cup \{v_{\min}\}$ greater than 5, $c = 6$ is the smallest consistent lower bound for g.

Our main reason for selecting the smallest consistent lower bound for g is that this selection can be used to force our algorithm to make the correct identification (required by property 3). Suppose that if our algorithm identifies c^*, and if all other identifications are correct, then the result $\mathcal{A}$ will be such that $L(\mathcal{A}) = L_t$. Hence, our algorithm should identify c^*. In this case, there always exist examples that result in an inconsistency when our algorithm selects any valuation smaller than c^*. The reason is that an example that fires δ with valuation $c^* - 1$ should actually fire a different transition, to a different state, or with a different reset value. Hence, the languages after firing these transitions are different. Therefore, there would exist two timed strings $\tau \in L_t$ and $\tau' \in L_t^{\mathrm{C}}$ (that can be included in S) that have identical suffixes after firing δ with valuations c^* and $c^* - 1$ respectively. Moreover, any pair of string that fire δ with valuations greater or equal to c^* cannot lead to an inconsistency since their languages after firing δ are the same.

The reset r. After having identified the lower bound c of the clock guard g of δ, our algorithm needs to identify the reset r of δ. One may notice that the identification of g depends on whether δ contains a clock reset or not: the value of r determines the valuations that are reached by timed strings after firing δ (the clock can be reset to 0), hence this value determines whether $\mathcal{A}$ is consistent after trying a particular lower bound for g. In our example, $(a,4)(a,1)(a,3)$ and $(a,4)(a,1)(a,2)$ reach $(q',1)$ and $(q',0)$ respectively before their suffixes are identical if $r = \text{true}$. Because of this, our algorithm identifies the clock reset r of δ at the same time it identifies its clock guard g. The method it uses to identify r is very simple: first set $r = \text{true}$ and then find the smallest consistent lower bound c_1 for g, then set $r = \text{false}$ and find another such lower bound c_2 for g. The value of r is set to true if and only if the lower bound found with this setting is smaller than the other one, i.e., iff $c_1 \leq c_2$. There always exist timed strings that ensure that the smallest consistent lower bound for g when the clock reset is set incorrectly is larger than when it is set correctly (satisfying property 3). In our example the timed strings that ensure this are $(a,4)(a,2)(a,2)(b,3) \in S_+$ and $(a,4)(a,3)(a,2)(b,3) \in S_-$. Because these examples reach the same valuations

in state q' only if the clock is reset, they create an inconsistency when r is set to true. In general, such strings always exists since the difference of 1 time value is sufficient for such an inconsistency: a difference of 1 time value can always be the difference between later satisfying and not satisfying some clock guard.[3]

The target state q'. Having identified both the clock guard and the reset of δ, our algorithm still needs to identify the target state q' of δ. Since we need to make sure that our algorithm is capable of identifying any possible transition (required by property 3), we need to try all possible settings for q', and in order to make it easier to prove the existence of a characteristic set (required by property 3), we do so in a fixed order. The order our algorithm uses is the order in which our algorithm identified the states, i.e., first q_0, then the first additional identified state, then the second, and so on. The target state for δ is set to be the first consistent target state in this order. In our example, we just try state q_0, then state q_1, and finally state q_3. When none of the currently identified states result in a consistent 1-DTA $\mathcal{A}$, the target is set to be a new state. This new state is set to be a final state only if there exists a timed string in S_+ that ends in it. It should be clear that since the languages after reaching different states are different, there always exist timed strings that ensure that our algorithm identifies the correct target (satisfying property 3). In our example, there exist no timed strings that make $\mathcal{A}$ inconsistent when our algorithm tries the first state (state q_0), and hence our algorithm identifies a transition $\langle q_1, q_0, a, 6 \leq x \leq 9, \text{false}\rangle$.

This completes the identification of δ and (possibly) q'. This identification of a single transition δ essentially describes the main part of our algorithm (see Algorithm 1). However, we still have to explain how our algorithm iterates over the transitions it identifies. The algorithm consists of a main loop that iterates in a fixed order over the possible source states and labels for new transitions. For every combination of a source state q and a label a, our algorithm first sets two values: $v_{\min}$ and c'. The first is the smallest reachable valuation in q. The second is the fixed upper bound of the delay guard of a new transition. Because our model is deterministic, this is set to be the largest reachable valuation for which there exists no transition with q as source state and a as label. After identifying a transition δ with these values, our algorithm updates c' to be one less than the lower bound of the clock guard of δ. If c is still greater than $v_{\min}$, there are still transitions to identify for state q and label a. Thus, our algorithm iterates and continues this iteration until c' is strictly less than $v_{\min}$. Our main reason for adding this additional iteration is that it makes it easier to prove the convergence of our algorithm (property 4). The main loop of our algorithm continuously identifies new transitions and possibly new target states until there are no more new transitions to identify, i.e., until there exists a transition for every reachable timed state in $\mathcal{A}$. This is necessary because identifying a transition δ can create new identifiable transitions. This happens when the smallest reachable

[3] This holds unless the clock guard can only be satisfied by a single valuation v, i.e., unless $g = c \leq x \leq c$. However, in this case any setting for r is correct since both can lead to results such that $L(\mathcal{A}) - L_t$.

valuation $v_{\min}$ in some state is decreased, or when a new state is identified, by the identification of δ.

5 Properties of the Algorithm

We described an algorithm for the identification of 1-DTAs. We claim now, and argued in the previous section, that the algorithm converges efficiently to a correct 1-DTA, i.e., that it identifies 1-DTAs efficiently in the limit. In this section, we show that the four properties mentioned in the previous section hold. The combination of these properties satisfies all the constraints required for efficient identification in the limit (Definition 4), and hence shows that 1-DTAs are efficiently identifiable (Theorem 1). We only give sketches of proofs due to space restrictions.

Proposition 1. *ID_1DTA is a polynomial time algorithm (properties* 1 *and* 2*).*

Proof. (sketch) Identifying a single transition can be done in polynomial time as argued in the previous section. In addition, every transition is guaranteed to be fired by at least one timed string from S. Hence, our algorithm will stop after identifying a polynomial amount of transitions. The proposition follows. □

Lemma 1. *There exist polynomial characteristic sets of the transitions of 1 - DTAs for ID_1DTA (property* 3*).*

Proof. (sketch) As shown by the example in the previous section, we can always find a polynomial amount of timed strings such that our algorithm identifies the correct transition δ. In addition, we can bound each of these strings by a polynomial since 1-DTAs are polynomially distinguishable. Moreover, because the order in which our algorithm identifies transitions is independent of S, it is impossible to add additional examples to S such that our algorithm will no longer identify δ. This proves the lemma. □

Lemma 2. *ID_1DTA converges after identifying a polynomial amount of transitions (property* 4*).*

Proof. (sketch) Since 1-DTAs are polynomially distinguishable, any state can be reached by timed strings τ of polynomial length. Hence, the main loop will be run at most $|\tau|$ times before the smallest reachable valuation in any state of $\mathcal{A}_t$ can be reached. Once the smallest reachable valuation can be reached, every transition can be identified. In a single run of the main loop, at most $|\mathcal{A}_t|$ new correct transitions can be identified. Thus, our algorithm identifies at most $|\tau| \times |\mathcal{A}_t|$ transitions before every transition can be identified. Hence, by Lemma 1, every transition of $\mathcal{A}_t$ can be identified correctly after identifying $|\tau| \times |\mathcal{A}_t|$ transitions. □

Theorem 1. *1-DTAs are efficiently identifiable in the limit.*

Proof. By Proposition 1 and Lemma 2, if all the examples from Lemma 1 are included in S, our algorithm returns a 1-DTA $\mathcal{A}$ such that $L(A) = L_t$ in polynomial time and from polynomial data. We conclude that Algorithm 1 identifies 1-DTAs efficiently in the limit. □

6 Conclusions

In this paper we described an algorithm that identifies 1-DTAs efficient in the limit. This algorithm is an important step in the direction of being able to identify timed systems efficiently. To the best of our knowledge, our result is the first positive efficiency result for identifying timed automata. The fact that 1-DTAs can be identified efficiently has important consequences for anyone interested in identifying timed systems. Most importantly, it is a reason to model timed systems with 1-DTAs. However, when 1-DTAs are too restrictive, our result is still useful because identification algorithms for other DTAs could identify the class of 1-DTAs efficiently. This is a desirable property since 1-DTAs are efficiently identifiable. For instance, in related work, a query learning algorithm is described for identifying event recording automata (ERAs) [9]. However, the used query learning algorithm requires an exponential amount of queries (data). It would be interesting to either adapt the timed query learning algorithm to the class of 1-DTAs, or to show that the algorithm does in fact identify 1-DTAs efficiently. This could result in an efficient query learning algorithm for timed systems.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126, 183–235 (1994)
2. Larsen, K.G., Petterson, P., Yi, W.: Uppaal in a nutschell. International journal on software tools for technology transfer 1(1-2), 134–152 (1997)
3. Gold, E.M.: Complexity of automaton identification from given data. Information and Control 37(3), 302–320 (1978)
4. Pitt, L., Warmuth, M.: The minimum consistent dfa problem cannot be approximated within and polynomial. In: Annual ACM Symposium on Theory of Computing, pp. 421–432. ACM, New York (1989)
5. Gold, E.M.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)
6. Oncina, J., Garcia, P.: Inferring regular languages in polynomial update time. In: Pattern Recognition and Image Analysis. Series in Machine Perception and Artificial Intelligence, vol. 1, pp. 49–61. World Scientific, Singapore (1992)
7. de la Higuera, C.: Characteristic sets for polynomial grammatical inference. Machine Learning 27(2), 125–138 (1997)
8. Verwer, S., de Weerdt, M., Witteveen, C.: Polynomial distinguishability of timed automata. In: Clark, A., Coste, F., Miclet, L. (eds.) ICGI 2008. LNCS, vol. 5278, pp. 238–251. Springer, Heidelberg (2008)
9. Grinchtein, O., Jonsson, B., Pettersson, P.: Inference of event-recording automata using timed decision trees. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 435–449. Springer, Heidelberg (2006)

Author Index